

FM Meets AI

Equipping AI Agents with Formal-Methods Tools

web: <https://www.sosy-lab.org/research/fm-tools>

Dirk Beyer, Thomas Lemberger, Henrik Wachowitz
LMU Munich, Germany

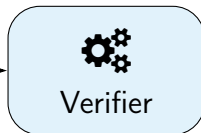
2026-10-24, ISoLA 2026, Kos, Greece



Automatic Software Verification

Program + Specification

```
int main() {  
    int a = foo();  
    int b = bar(a);  
    assert(a == b);  
}
```



TRUE: specification holds
+ witness with invariants

UNKNOWN: out of resources

FALSE: bug found
+ witness with error path

Status on Verifiers

- ▶ From lack of verifiers to plentitude
- ▶ 74 verification tools evaluated in SV-COMP 2026 [1, 2]

Competitions in Software Verification and Testing

Mature research area, and there are tool competitions:

- ▶ RERS: off-site, tools, free-style [3]
- ▶ SV-COMP: off-site, automatic tools, controlled [4]
- ▶ Test-Comp: off-site, automatic tools, controlled [5]
- ▶ VerifyThis: on-site, interactive, teams [6]

(alphabetic order)

SV-COMP (Automatic Tools 2012)



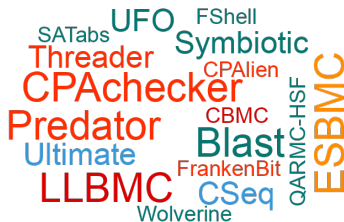
A word cloud of automatic tools from the SV-COMP 2012 competition. The tools are arranged in a roughly circular pattern. The colors of the text are: FShell (black), QARMC-HSF (orange), LLBMC (orange), SATabs (teal), Predator (blue), Blast (blue), ESBMC (teal), CPAchecker (red), and Wolverine (teal).

FShell
QARMC-HSF
LLBMC
SATabs
Predator
Blast
ESBMC
CPAchecker
Wolverine

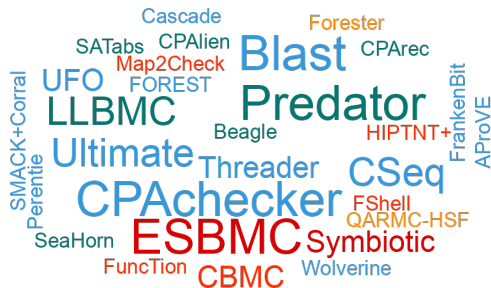
SV-COMP (Automatic Tools 2013, cumulative)



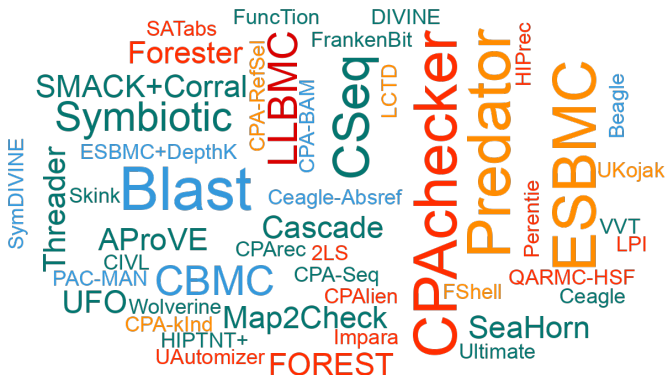
SV-COMP (Automatic Tools 2014, cumulative)



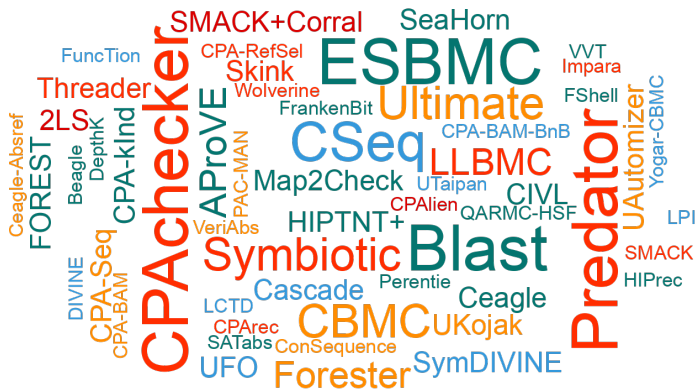
SV-COMP (Automatic Tools 2015, cumulative)



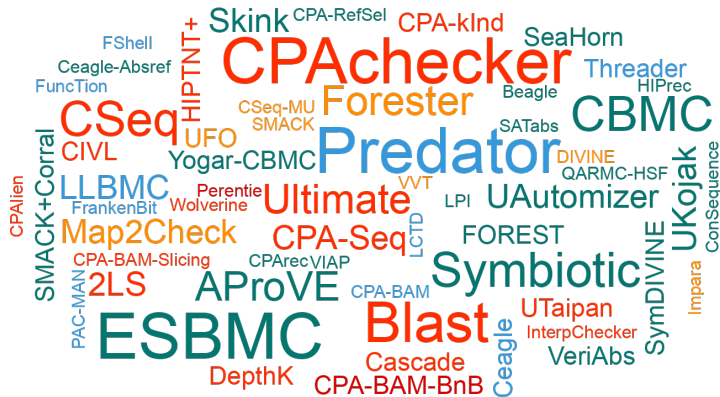
SV-COMP (Automatic Tools 2016, cumulative)



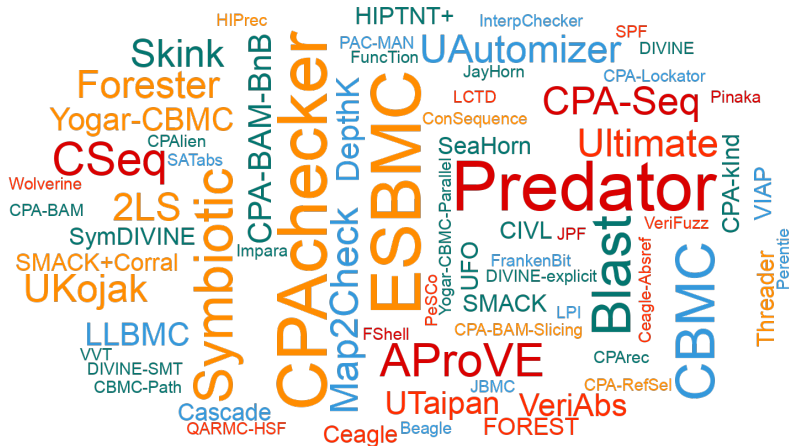
SV-COMP (Automatic Tools 2017, cumulative)



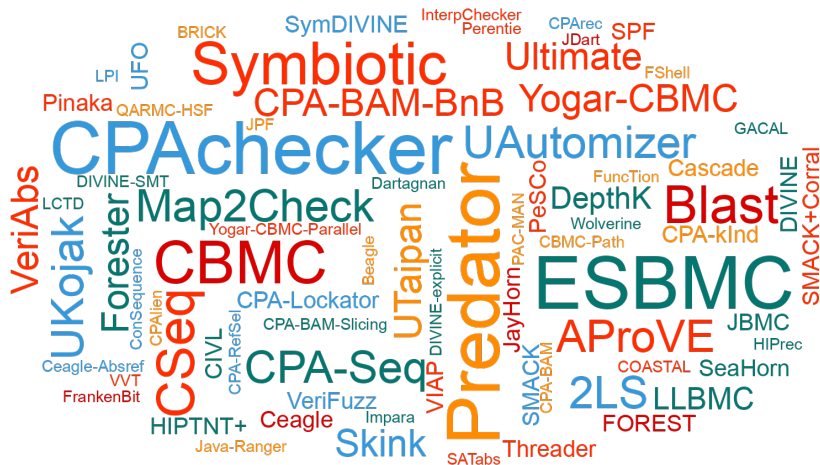
SV-COMP (Automatic Tools 2018, cumulative)



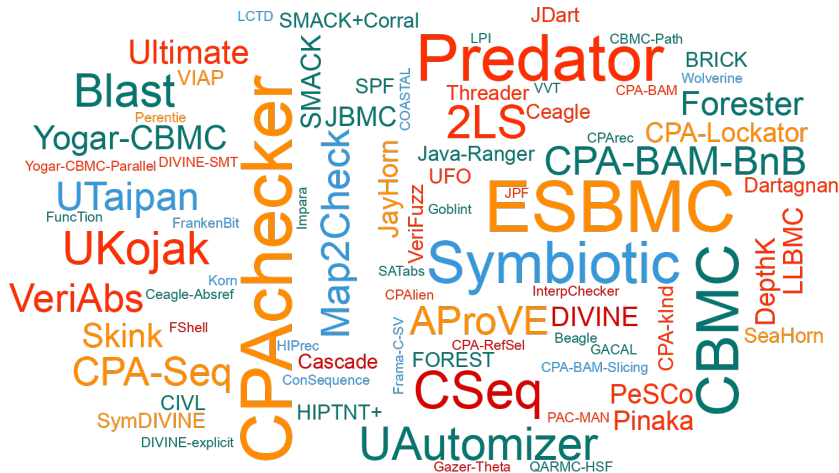
SV-COMP (Automatic Tools 2019, cumulative)



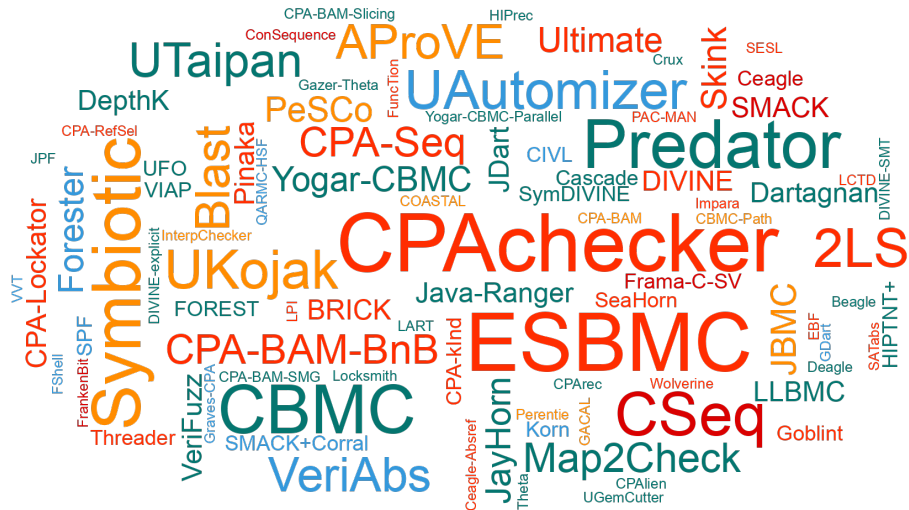
SV-COMP (Automatic Tools 2020, cumulative)



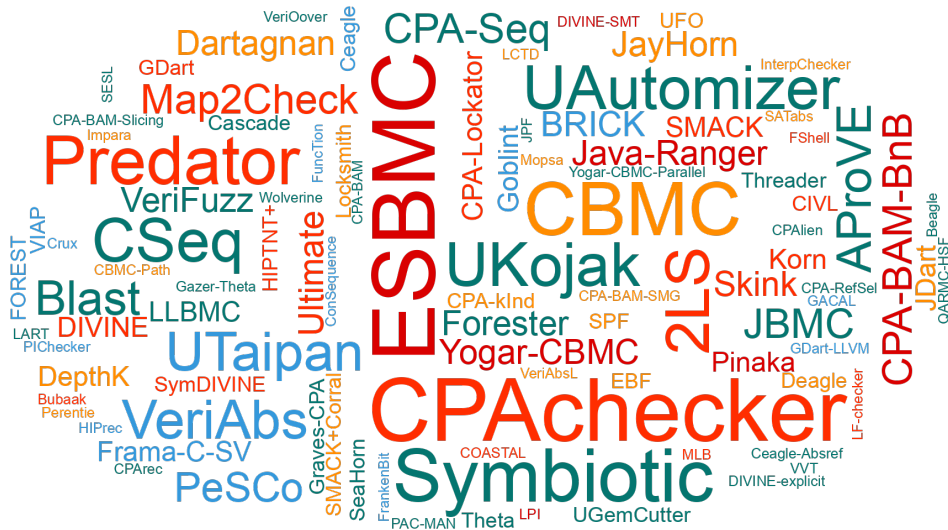
SV-COMP (Automatic Tools 2021, cumulative)



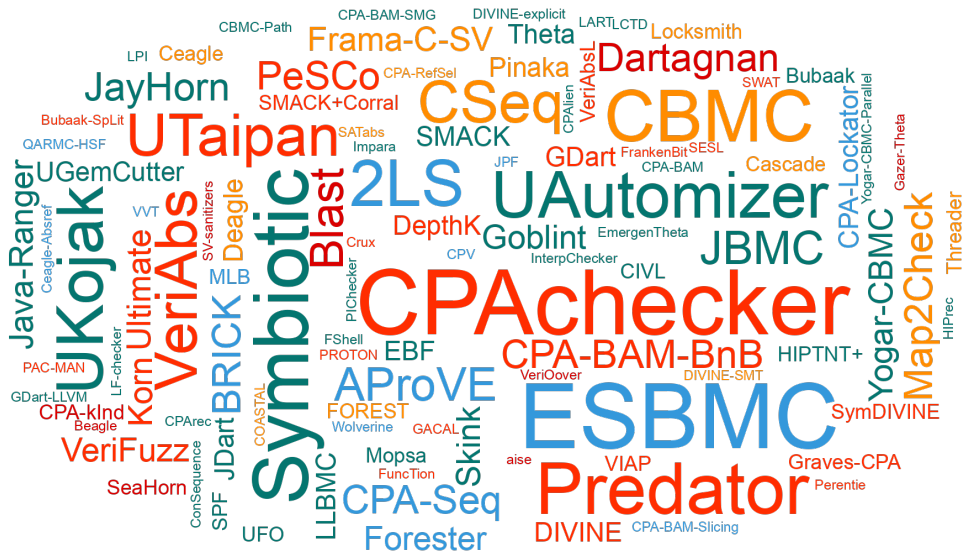
SV-COMP (Automatic Tools 2022, cumulative)



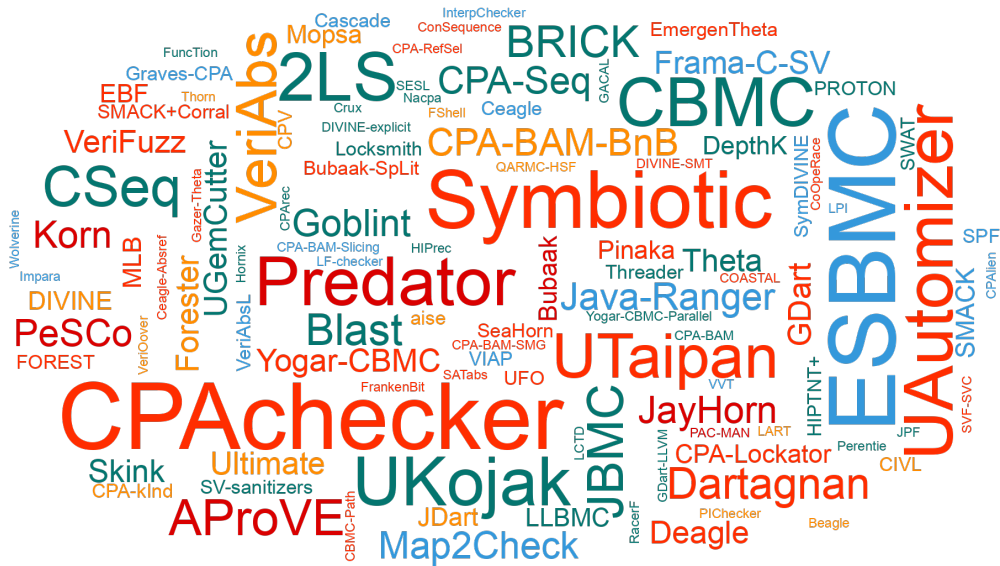
SV-COMP (Automatic Tools 2023, cumulative)



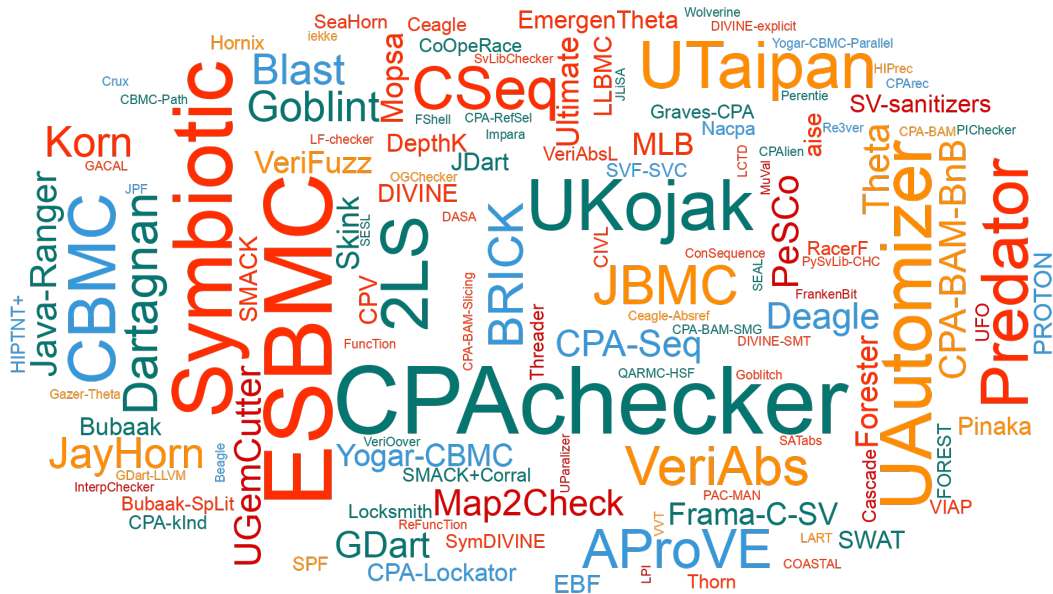
SV-COMP (Automatic Tools 2024, cumulative)



SV-COMP (Automatic Tools 2025, cumulative)

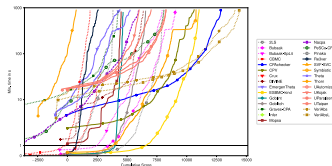


SV-COMP (Automatic Tools 2026, cumulative)



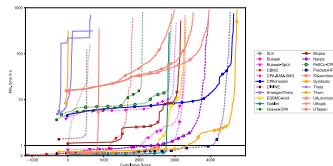
Different Strengths

C.ReachSafety



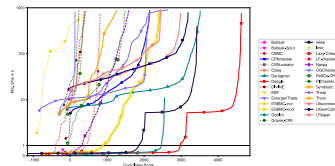
1. CPAchecker
2. ESBMC-kind
3. Symbiotic

C.MemSafety



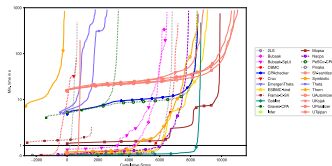
1. Symbiotic
2. CPAchecker
3. UAutomizer

C.Concurrency



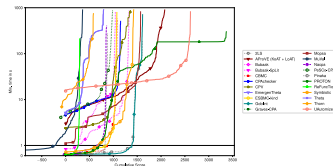
1. Deagle
2. Dartagnan
3. iekke

C.NoOverflows



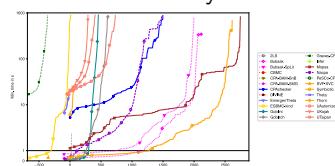
1. UParalizer
2. UAutomizer
3. UTaipan

C.Termination



1. PROTON
2. UAutomizer
3. AProVE (KoAT + LoAT)

C.SoftwareSystems



1. Mopsa
2. Symbiotic
3. CPAchecker

SV-COMP 2026 [2]: <https://sv-comp.sosy-lab.org/2026/results>

Different Techniques

Tool	Abstract Interpretation	Algorithm Selection	ARG-Based Analysis	Automata-Based Analysis	Bit-Precise Analysis	Bounded Model Checking	CEGAR	Concurrency Support	Constrained Horn Clauses	Explicit-Value Analysis	Floating-Point Arithmetics	Interpolation	k-Induction	Lazy Abstraction	Numeric Interval Analysis	Portfolio	Predicate Abstraction	Property-Directed Reach.	Ranking Functions	Separation Logic	Shape Analysis	Symbolic Execution	Targeted Input Generation	Task Translation
AISE																						✓		
AProVE																			✓			✓		
BRICK						✓	✓	✓							✓							✓		
BUBAAK [⊗]					✓			✓								✓			✓			✓		
BUBAAK-SPLIT [⊗]		✓		✓	✓			✓					✓			✓			✓			✓		
CBMC [⊗]					✓	✓		✓														✓		
COASTAL [⊗]																					✓			
CONCURRENTW2T								✓																
CoOpERACE ^{meta}		✓						✓								✓								
CPA-BAM-BnB [⊗]		✓	✓		✓		✓		✓			✓	✓	✓			✓				✓	✓		
CPACHECKER	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		

Algorithms and techniques used by the participating tools

Excerpt from Table 8 of the [competition report SV-COMP 2026 \[2\]](#)

Example CPAchecker [24]: Many Concepts

- ▶ Included Concepts:
 - ▶ CEGAR [7] Interpolation [8, 9]
 - ▶ Configurable Program Analysis [10, 11]
 - ▶ Adjustable-block encoding [12]
 - ▶ Conditional model checking [13]
 - ▶ Verification witnesses [14, 15]
 - ▶ Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- ▶ Available analyses approaches:
 - ▶ Predicate abstraction [16, 12, 11, 17]
 - ▶ IMPACT algorithm [18, 19, 9]
 - ▶ Bounded model checking [20, 9]
 - ▶ k-Induction [21, 9]
 - ▶ IC3/Property-directed reachability [22]
 - ▶ Explicit-state model checking [8]
 - ▶ Interpolation-based model checking [23]

Insights from Software Model Checking

- ▶ Verifiers have different strengths
- ▶ There are plenty of tools
- ▶ $\Rightarrow$ Cooperative Verification Approaches

Cooperative Verification — Think big!

- ▶ Introduce a new level!
- ▶ Current tools should become "low level" components (engines)
- ▶ Construct combinations
- ▶ Clear Interfaces
via, e.g., Conditions, Witnesses, Test Suites
- ▶ Success: SAT, SMT (common interfaces, usable as libraries)
- ▶ See also: Little Engines [25], Evidential Tool Bus [26]

Cooperative Verification — Think big!

- ▶ Introduce a new level!
- ▶ Current tools should become "low level" components (engines)
- ▶ Construct combinations
- ▶ Clear Interfaces
via, e.g., Conditions, Witnesses, Test Suites
- ▶ Success: SAT, SMT (common interfaces, usable as libraries)
- ▶ See also: Little Engines [25], Evidential Tool Bus [26]

New actors: **AI agents** generate code and can orchestrate tools.

AI Agents Write a Lot of Our Code

AI agents:

- ▶ Now everyday part of software development
- ▶ Draft, refactor, and debug code
- ▶ Speed was unthinkable a few years ago
- ▶ No extensive onboarding for using libraries and languages

Formal-Methods Tools Can Help ...

- ▶ Verifiers and test generators give *rigorous* answers:
Verify if code meets a specification
- ▶ AI agents could now *invoke* these powerful tools
- ▶ Many such tools exist: the verifiers we have seen,
plus 21 test generators (Test-Comp 2026 [27])

Formal-Methods Tools Can Help ...

- ▶ Verifiers and test generators give *rigorous* answers:
Verify if code meets a specification
- ▶ AI agents could now *invoke* these powerful tools
- ▶ Many such tools exist: the verifiers we have seen,
plus 21 test generators (Test-Comp 2026 [27])

... but formal-methods tools have been **out of reach** for AI agents *and* non-expert developers alike.

Why Tools Are Hard to Use

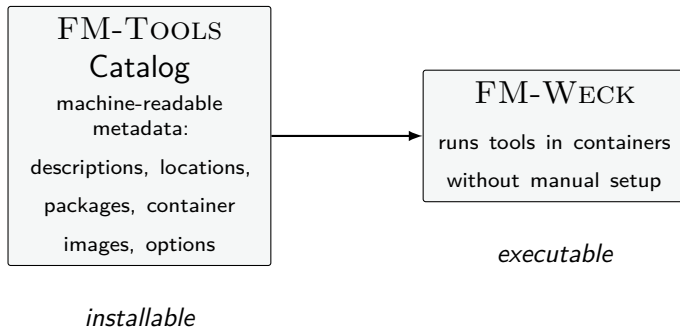
Each formal-methods tool requires:

- ▶ a specific runtime environment
(Linux distribution, JDK version, LLVM version, . . .)
- ▶ its own extensive command-line interface
- ▶ specifications in a particular format

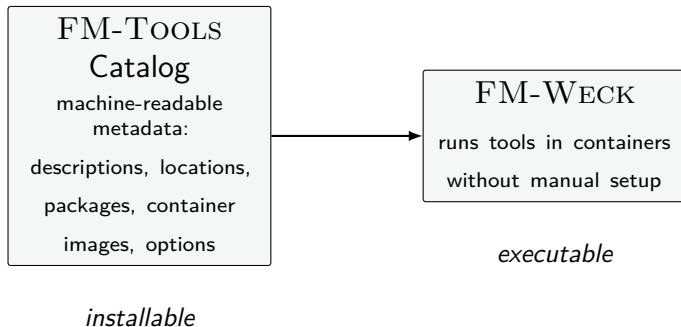
And, on top of that:

- ▶ *Setting up* a single verifier requires substantial expertise
- ▶ *Choosing the right* verifier for a given task is a research question of its own

What the Community Already Built



What the Community Already Built



Tools are *installable* and *executable* in a uniform way — but **not yet discoverable and usable** for an AI agent with no prior knowledge of formal methods.

The use of software has entered a new era:

- ▶ software that was classically written for human users
- ▶ should now be discoverable and usable by AI agents as well
- ▶ the integration and usage must be effective and efficient
- ▶ some software will be written for AI agents alone

Contribution

A set of new **agent skills** that closes this gap.

They let AI agents (e.g., Claude Code):

1. **list** the available formal-methods tools
2. **select** the best-ranked tool for the task
3. **invoke** it on a given input program
4. **interpret** the result — all without manual setup

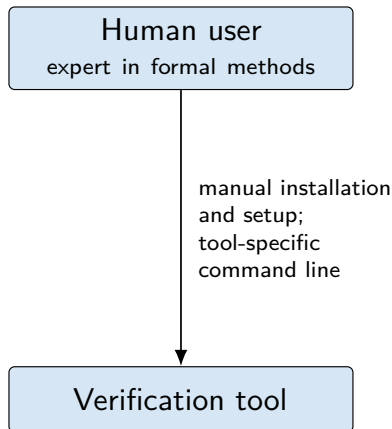
Concretely, we contribute:

- ▶ structured, authoritative FM-TOOLS data accessible via a skill and `llms.txt` files,
- ▶ structured competition results from SV-COMP and Test-Comp (no web scraping),
- ▶ automatic, containerized tool selection and execution via FM-WECK

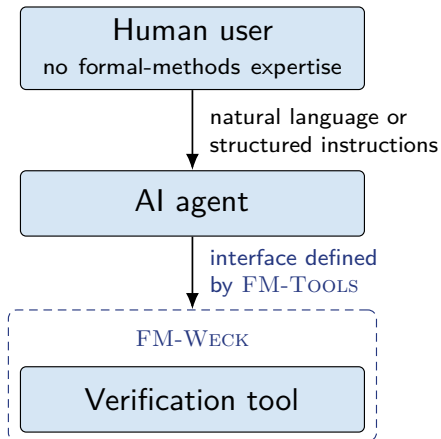
Structured data also *reduces token consumption* vs. scraping web pages.

Accessing a Verification Tool: Classic vs. with an AI Agent

Classic usage



Usage with an AI agent



FM-TOOLS defines the interface between agent and tool.
FM-WECK encapsulates the tool for installation and execution.

Three Core Concepts

1. **Off-the-shelf components:** treat verifiers, testers, and solvers as binary components to build upon
2. **Conservation:** keep tool information and usage in a central catalog: the FM-TOOLS Catalog [28]
3. **Execution:** run tools in an automated, containerized environment: FM-WECK [29]



The FM-TOOLS Catalog and FM-WECK

FM-Tools Catalog [28]

- ▶ YAML data files per tool: project, maintainers, parameters
- ▶ per version: DOI of the archive + the expected runtime environment

FM-Weck [29] — a unified command-line tool that

- ▶ downloads, installs, and runs any catalog tool
- ▶ executes it in a *container* (no host dependencies)
- ▶ is inspired by package managers (pipx, npx)

fm-weck

run

cpachecker:4.0

example.c

How to Connect an AI Agent to Tools and Data

Three established mechanisms:

- ▶ **Agent skill**: an instruction file with a short YAML front matter;
loaded *when its description matches the task*;
tells the agent *how* to use what it already has
- ▶ **MCP server** (Model Context Protocol):
a running server exposing typed JSON-RPC methods the agent can call
- ▶ **llms.txt**: a text file at a website root giving an LLM
a curated overview of the site

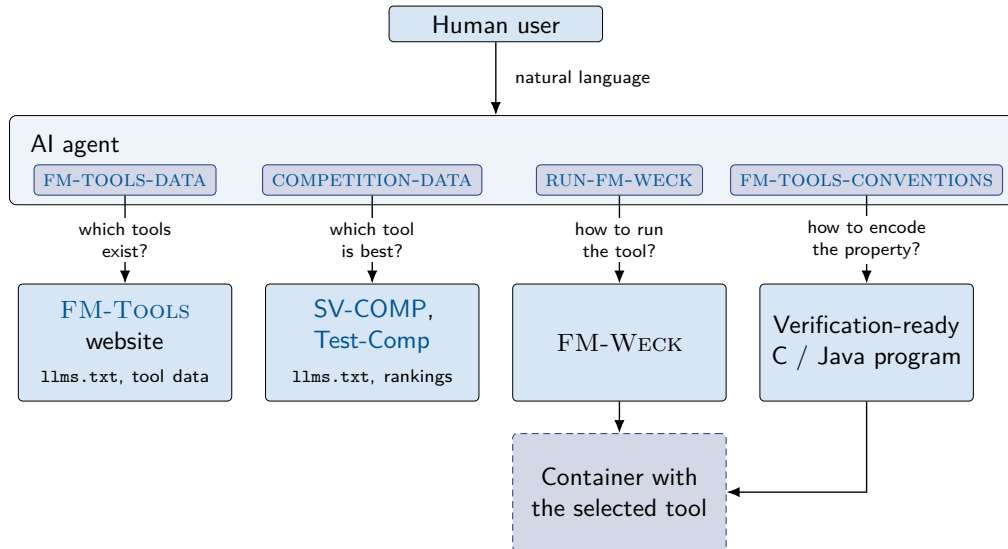
How to Connect an AI Agent to Tools and Data

Three established mechanisms:

- ▶ **Agent skill**: an instruction file with a short YAML front matter;
loaded *when its description matches the task*;
tells the agent *how* to use what it already has
- ▶ **MCP server** (Model Context Protocol):
a running server exposing typed JSON-RPC methods the agent can call
- ▶ **llms.txt**: a text file at a website root giving an LLM
a curated overview of the site

We choose **skills**: more lightweight, easier to provide, extend, maintain, and install — no extra server to run.

Four New Agent Skills



Each skill connects the agent to the data or the execution engine that answers one question.

Skill 1: FM-TOOLS-DATA

Gives the agent step-by-step instructions to retrieve tool information from the FM-TOOLS Catalog (via `llms.txt` and YAML files).

name: fm-tools-data

description: "Catalog lookup for metadata about formal methods tools listed at `fm-tools.sosy-lab.org`: maintainers, input languages, supported properties, container images, literature, competition history. Trigger when the user asks WHAT a verifier supports, HOW it works, or WHERE to find it. Also for LIST or COMPARE by technique, framework, or input language. Do NOT trigger for rankings/scores (competition-data), running a verifier (`run-fm-weck`), or general FM concepts that do not require catalog data."

Skill 2: COMPETITION-DATA

Makes *competition results* usable for data-driven tool selection.

- ▶ Currently SV-COMP and Test-Comp
- ▶ Auxiliary files we provide on the competition websites:
 - ▶ CSV files as an alternative to the HTML result tables,
 - ▶ one *summary Markdown* file per competition: per category, the top-three participants by score
 - ▶ an `llms.txt` pointing to all of these (plus rules and task definitions)
- ▶ Turns the agent into a *recommender system* using the latest competition data

Example: match the user's input program to the closest category, then read the top-three participants for that category.

Skill 3: RUN-FM-WECK

Instructs the agent how to call FM-WECK to run a catalog tool.

name: run-fm-weck

description: "Use this skill when the user explicitly mentions fm-weck, asks to run a named formal software verifier (CPAchecker, Ultimate Automizer, ESBMC, CBMC, or similar) on a program, asks which tools or versions fm-weck supports, or asks to verify a program without naming a specific tool. In the last case, use the competition-data skill to map the program's property to a competition and category and select the best-ranked tool before proceeding. Do NOT trigger for general static analysis (linters, sanitizers, Valgrind) or formal methods discussion unrelated to running a tool."

Skill 3: RUN-FM-WECK — Structure

Short sections, each answering one question the agent must address:

- ▶ *Overview*: what FM-WECK does, at a high level
- ▶ *Prerequisites*: system requirements
- ▶ *Installation*: a decision tree picking the first method that works — try `pipx` first (isolated install), fall back to a virtual environment
Goal: avoid polluting the user's global Python packages
- ▶ *Basic usage*: one concrete FM-WECK example
- ▶ *Tool selection*: if no tool is named, use COMPETITION-DATA to find a fitting tool

Skill 4: FM-TOOLS-CONVENTIONS

Equips agents with the input conventions expected by formal-methods tools.

- ▶ Conventions established by the SV-COMP and Test-Comp communities (not obvious to an AI agent), e.g., DIMACS CNF for SAT solvers or JML for Java deductive verifiers.
- ▶ The skill describes how to:
 - ▶ encode *non-deterministic input* in C and Java,
 - ▶ encode *assumptions*,
 - ▶ encode *assertions*.

Setup: Installing the Skills

The skills follow the *agent-skills convention* supported by all major AI agents.
Maintained at:

<https://gitlab.com/sosy-lab/benchmarking/fm-tools/-/tree/main/skills> Install
with one command:

```
npx skills install \  
  https://gitlab.com/sosy-lab/benchmarking/fm-tools
```

The `llms.txt` files are served from the official SV-COMP and Test-Comp websites and are used automatically by agents that access those sites.

Four Use Cases

We exercise each use case *without* skills, then *with* skills, to show the added value (agent: Claude Code, model Sonnet 4.6):

1. Get information about a *specific tool*.
2. Check a C program for *memory safety*.
3. Check *auto-generated* Java code.
4. Act as a *research assistant* for literature.

Full conversations: Zenodo [30] and <https://www.sosy-lab.org/research/fm-tools>

Use Case 1: Information About a Specific Tool

Task: Who maintains CPACHECKER, and what is the latest version?

Without skill

- ▶ several web searches,
- ▶ fetch data from the CPACHECKER repository,
- ▶ then answers correctly.

~40k tokens in, 1.2k out
total cost \$0.18

With fm-tools-data

- ▶ loads the skill,
- ▶ retrieves version + maintainers directly from the catalog.

~9k tokens in, 1.1k out
total cost \$0.093

Same correct answer — more reliable (no web search) and $\approx$ half the cost.

Use Case 2: Memory Safety of a C Program

Without skills: checks for installed verifiers, finds none, falls back to compiler checks (e.g. AddressSanitizer), then concludes “memory safe” — without a verifier.

With skills (excerpt):

```
> Verify the memory-safety of my list program.  
* Skill(run-fm-weck)      \ Successfully loaded skill  
* Skill(competition-data) \ Successfully loaded skill  
* Fetch(https://sv-comp.sosy-lab.org/11ms.txt)  
* Fetch(.../2026/results/.../summary.md)  
* The most fitting category for a doubly-linked circular  
  list is C.valid-memsafety.LinkedList, where PredatorHP  
  ranked 1st in SV-COMP 2026 with a score of 220. [...]
```

Runs PREDATORHP (top-ranked) and reports correctly, with the confidence of the best available tool.

Use Case 3: Checking Auto-Generated Code

Task: an algorithm generated in Java — strengthen trust by formulating and checking pre-/postconditions.

Without skills

- ▶ writes a separate Dafny program to confirm the idea,
- ▶ user must clarify: check the Java code directly,
- ▶ writes JML annotations,
- ▶ downloads OPENJML, installs prerequisites,
- ▶ after several tries, verifies.

With skills

- ▶ FM-TOOLS-CONVENTIONS: use correct constructs for non-deterministic inputs + pre-/postconditions
- ▶ RUN-FM-WECK + COMPETITION-DATA: pick a verifier (here SWAT) and run it
- ▶ verifies correctly (spec via `verifierAssert()`)

Use Case 4: Research Assistant

Task: find related work on test generation for C programs — best test generators in the last Test-Comp, plus maintainers and literature.

Without skills

- ▶ broad web search,
- ▶ partially correct, but also partially outdated and incomplete.

~220k in, 9.4k out
total cost \$0.61

With skills

- ▶ COMPETITION-DATA for Test-Comp results,
- ▶ FM-TOOLS-DATA for reliable tool info.

~88k in, 12k out
total cost \$0.45

Similar-or-better results, lower cost, and up to date.

Conclusion

We equipped AI agents to **access, select, and execute** formal-methods tools without manual setup:

- ▶ agent skills expose the structured FM-TOOLS Catalog [28]
- ▶ and the competition results of SV-COMP [31] and Test-Comp [32]
- ▶ tools are invoked via FM-WECK [29] in an isolated container

Benefits: data-driven tool selection and
reduced token consumption vs. scraping HTML

Agents can also run SAT solvers such as KISSAT through FM-WECK



<https://www.sosy-lab.org/research/fm-tools>

References I

- [1] Beyer, D., Podelski, A.: Software model checking: 20 years and beyond. In: Principles of Systems Design. pp. 554–582. LNCS 13660, Springer (2022). doi:10.1007/978-3-031-22337-2_27
- [2] Beyer, D., Strejček, J.: Evaluating software verifiers for C, Java, and SV-LIB (Report on SV-COMP 2026). In: Proc. TACAS (2). pp. 461–502. LNCS 16506, Springer (2026). doi:10.1007/978-3-032-22749-2_23
- [3] Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Proc. ISO LA. pp. 608–614. LNCS 7609, Springer (2012). doi:10.1007/978-3-642-34026-0_45
- [4] Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). doi:10.1007/978-3-642-28756-5_38
- [5] Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019). doi:10.1007/978-3-030-17502-3_11
- [6] Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012: A program verification competition. STTT 17(6), 647–657 (2015). doi:10.1007/s10009-015-0396-8
- [7] 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). doi:10.1145/876638.876643
- [8] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). doi:10.1007/978-3-642-37057-1_11
- [9] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning 60(3), 299–335 (2018). doi:10.1007/s10817-017-9432-6

References II

- [10] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). doi:10.1007/978-3-540-73368-3_51
- [11] Beyer, D., Henzinger, T.A., Théoduloz, G.: Program analysis with dynamic precision adjustment. In: Proc. ASE. pp. 29–38. IEEE (2008). doi:10.1109/ASE.2008.13
- [12] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010). <https://dl.acm.org/doi/10.5555/1998496.1998532>
- [13] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). doi:10.1145/2393596.2393664
- [14] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). doi:10.1145/2786805.2786867
- [15] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). doi:10.1145/2950290.2950351
- [16] 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). doi:10.1109/FMCAD.2009.5351147
- [17] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). doi:10.1145/2491411.2491429

References III

- [18] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). doi:10.1007/11817963_14
- [19] Beyer, D., Wendler, P.: Algorithms for software model checking: Predicate abstraction vs. IMPACT. In: Proc. FMCAD. pp. 106–113. FMCAD (2012). <https://ieeexplore.ieee.org/document/6462562>
- [20] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). doi:10.1007/978-3-540-24730-2_15
- [21] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). doi:10.1007/978-3-319-21690-4_42
- [22] Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). doi:10.1007/978-3-030-45190-5_1
- [23] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning **69**(1), 5 (2025). doi:10.1007/s10817-024-09702-9
- [24] Beyer, D., Keremoglu, M.E.: CPACHECKER: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). doi:10.1007/978-3-642-22110-1_16
- [25] Shankar, N.: Little engines of proof. In: Proc. FME. pp. 1–20. LNCS 2391, Springer (2002). doi:10.1007/3-540-45614-7_1
- [26] Cruanes, S., Hamon, G., Owre, S., Shankar, N.: Tool integration with the Evidential Tool Bus. In: Proc. VMCAI. pp. 275–294. LNCS 7737, Springer (2013). doi:10.1007/978-3-642-35873-9_18

References IV

- [27] Beyer, D.: Evaluating tools for automatic software testing (Report on Test-Comp 2026). In: Proc. FASE. pp. 449–468. LNCS 16504, Springer (2026). doi:10.1007/978-3-032-22774-4_23
- [28] Beyer, D.: Find, use, and conserve tools for formal methods. In: Proc. Festschrift Podelski 65th Birthday. pp. 75–91. LNCS 14765, Springer (2026). doi:10.1007/978-3-032-13711-1_5
- [29] 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
- [30] Beyer, D., Lemberger, T., Wachowitz, H.: Conversation records for the ISOLA 26 paper: FM meets AI: Equipping AI agents with formal-methods tools. Zenodo (2026). doi:10.5281/zenodo.20319312
- [31] Beyer, D., Strejček, J.: Improvements in software verification and witness validation: SV-COMP 2025. In: Proc. TACAS (3). pp. 151–186. LNCS 15698, Springer (2025). doi:10.1007/978-3-031-90660-2_9
- [32] 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