

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

2026-09-30, with rtsrg@BME 2026, Budapest, Hungary



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

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

But: AI-generated code can be **subtly wrong** — violating safety properties, missing corner cases, relying on implicit assumptions that break in production.

Formal-Methods Tools Could Help ...

- ▶ Verifiers and test generators give *rigorous* answers:
Verify if code meets a specification
- ▶ AI agents could now *invoke* these powerful tools
- ▶ More than 60 verifiers and 20 test generators are maintained by the formal-methods community

Formal-Methods Tools Could Help ...

- ▶ Verifiers and test generators give *rigorous* answers:
Verify if code meets a specification
- ▶ AI agents could now *invoke* these powerful tools
- ▶ More than 60 verifiers and 20 test generators are maintained by the formal-methods community

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

Why Tools Are Hard to Use

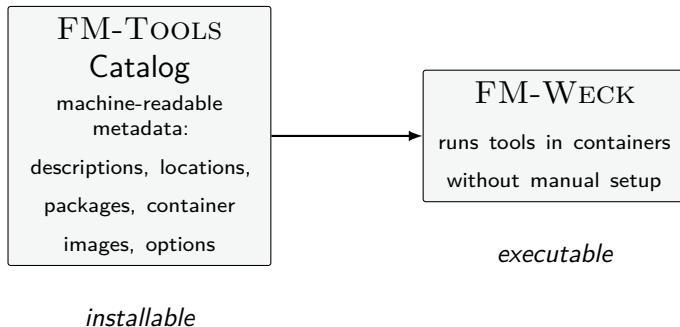
Each formal-methods tool requires:

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

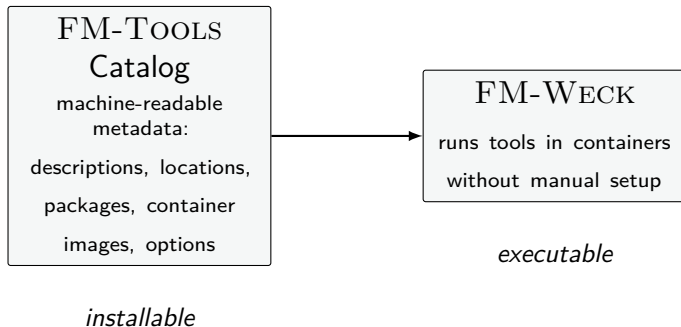
And, on top of that:

- ▶ *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

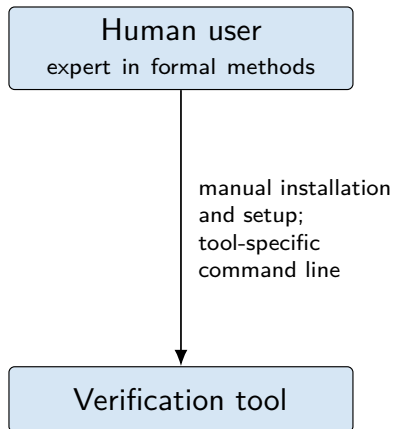
Concretely, we contribute:

- ▶ 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

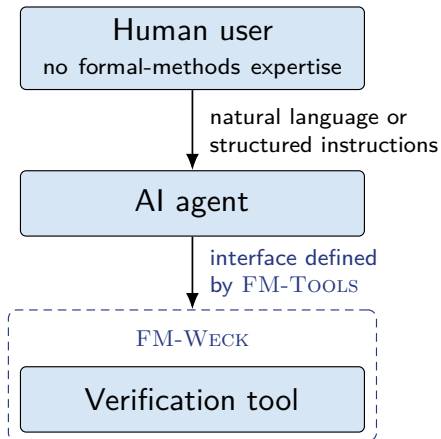
Structured data also *reduces token consumption* vs. scraping web pages.

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 [1]
3. **Execution:** run tools in an automated, containerized environment: FM-WECK [2]



The FM-TOOLS Catalog and FM-WECK

FM-Tools Catalog [1]

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

FM-Weck [2] — 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

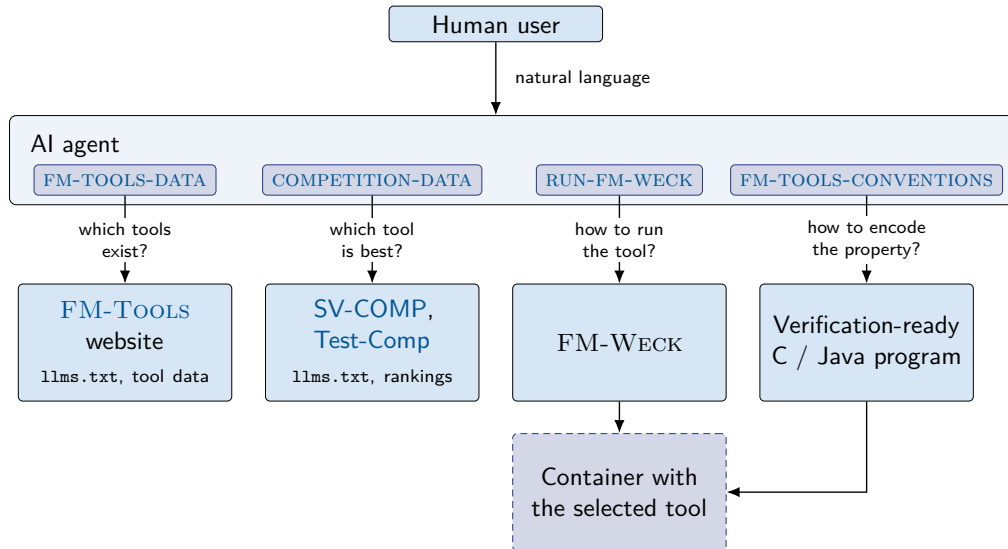
How to Connect an AI Agent to Tools and Data

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 [3] 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/llms.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 [1]
- ▶ and the competition results of SV-COMP [4] and Test-Comp [5]
- ▶ tools are invoked via FM-WECK [2] 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.: 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
- [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., 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
- [4] 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
- [5] 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