Mopsa-C at SV-Comp 2026

Marco Milanese, Raphaël Monat,
Abdelraouf Ouadjaout, Antoine Miné

rmonat.fr

SV-Comp @ ETAPS
13 April 2026

A General Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis

`gitlab.com/mopsa/mopsa-analyzer` `opam install mopsa`

A General Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis

`gitlab.com/mopsa/mopsa-analyzer` `opam install mopsa`

Different properties

- Runtime error detection
- Regression (patch, endianness)
- Non-exploitability

Modular Open Platform for Static Analysis

`gitlab.com/mopsa/mopsa-analyzer` `opam install mopsa`

Different properties

- Runtime error detection
- Regression (patch, endianness)
- Non-exploitability

Multiple languages

- C
- Python (+C)
- μOCaml
- Michelson

A General Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis

`gitlab.com/mopsa/mopsa-analyzer` `opam install mopsa`

Different properties

- Runtime error detection
- Regression (patch, endianness)
- Non-exploitability

Specificities

- Modular abstractions, loose coupling
- Built with relational domains in mind
- Ease dev.: interactive engine, hooks

Multiple languages

- C
- Python (+C)
- μOCaml
- Michelson

Modular Open Platform for Static Analysis

`gitlab.com/mopsa/mopsa-analyzer` `opam install mopsa`

Different properties

- Runtime error detection
- Regression (patch, endianness)
- Non-exploitability

Specificities

- Modular abstractions, loose coupling
- Built with relational domains in mind
- Ease dev.: interactive engine, hooks

Multiple languages

- C
- Python (+C)
- μOCaml
- Michelson

Contributors (2018–2026)

- G. Bau
- J. Boillot
- D. Delmas
- A. Fromherz
- M. Milanese
- A. Miné
- R. Monat
- A. Ouadjaout
- M. Journault
- F. Parolini
- M. Valnet
- T. Goalard

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe
   - **No?** ensure we found a counterexample with sanitizers

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe
   - **No?** ensure we found a counterexample with sanitizers
   - **Unknown?** restart with a more precise analysis configuration

Adapting Mopsa to SV-Comp's Framework

Sequential portfolio loop

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe
   - **No?** ensure we found a counterexample with sanitizers
   - **Unknown?** restart with a more precise analysis configuration

2026 Portfolio

1-2 Forward, over-approx. analyses using intervals

3 Backward, under-approx. analysis
   - Starting from alarms of the first analysis.
   - Experimental implementation

4-7 Forward, over-approx. analyses, with relational domains and partitioning

Improvements made in 2026

- Termination analysis

Improvements made in 2026

- Termination analysis
  - Introduces loop counters

Improvements made in 2026

- Termination analysis
  - Introduces loop counters
  - Check they are bounded

Improvements made in 2026

- Termination analysis
  - Introduces loop counters
  - Check they are bounded
  - Extended to handle goto-based loops too

Improvements made in 2026

- Termination analysis
  - Introduces loop counters
  - Check they are bounded
  - Extended to handle goto-based loops too
- Boolean condition normalization (improved precision)

Improvements made in 2026

- Termination analysis
  - Introduces loop counters
  - Check they are bounded
  - Extended to handle goto-based loops too
- Boolean condition normalization (improved precision)
- Refined handling of backward gotos

Improvements made in 2026

- Termination analysis
  - Introduces loop counters
  - Check they are bounded
  - Extended to handle goto-based loops too
- Boolean condition normalization (improved precision)
- Refined handling of backward gotos
- Removed spurious overflow alarms in variable-length arrays (by comparing to Goblint!)

Backward, under-approximating analysis results

Category	|false tasks|	Validated	Unconfirmed	Best |false|, verifier	
ReachSafety	3076	137	36	1743	CPAchecker
NoOverflows	3680	1473	92	2847	Symbiotic
MemSafety	2156	1035	7	2042	Symbiotic
SoftwareSystems	1568	15	4	324	Symbiotic

2026 Improvements

Subcategory	Prop.	|tasks|	Mopsa'25	Mopsa'26	Best score, verifier (2026)	
Loops	R	764	386	491	982	aise
Recursive	R	174	60	85	150	Symbiotic
Arrays	M	221	124	179	356	UAutomizer
Juliet	M	3271	2530	3788	4709	CPAchecker
Main	N	1986	2138	2384	2763	UParalizer
Heap	T	201	_	258	352	PROTON
Other	T	1630	_	1576	2184	PROTON
Busybox	N	65	8	14	14	Mopsa
Other	R	70	12	22	35	ESBMC-kind
Other	M	49	14	18	18	Mopsa
DDL	T	215	_	32	201	UAutomizer
Uthash	T	32	_	36	54	PROTON

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Weaknesses

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Weaknesses

- Experimental backward analysis

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Weaknesses

- Experimental backward analysis
- Incorrectness witnesses not validated (SV-Sanitizer validator?)

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Weaknesses

- Experimental backward analysis
- Incorrectness witnesses not validated (SV-Sanitizer validator?)
- No support of concurrent programs, recursive functions

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Weaknesses

- Experimental backward analysis
- Incorrectness witnesses not validated (SV-Sanitizer validator?)
- No support of concurrent programs, recursive functions
- Fixed sequence of configurations