Mopsa-C at SV-Comp 2025

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné

`rmonat.fr`

SV-Comp
1 April 2025

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
- Portability (patch, endianness)
- Non-exploitability

A General Overview of Mopsa

Modular Open Platform for Static Analysis

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

Different properties

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

Specificities

- Modular abstractions, loose coupling
- Optimized for relational domains
- Ease dev.: interactive engine, hooks

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
- Portability (patch, endianness)
- Non-exploitability

Specificities

- Modular abstractions, loose coupling
- Optimized for relational domains
- Ease dev.: interactive engine, hooks

Multiple languages

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

Contributors (2018–2025)

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

Adapting Mopsa to SV-Comp’s Framework

Our approach

1. Analyze the target program with Mopsa

Adapting Mopsa to SV-Comp's Framework

Our approach

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

Our approach

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

Adapting Mopsa to SV-Comp's Framework

Our approach

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?** restart with a more precise analysis configuration

Adapting Mopsa to SV-Comp's Framework

Our approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
    - **Yes?** finished, program is <u>safe</u>
    - **No?** restart with a more precise analysis configuration

$\rightsquigarrow$ Mopsa returns unknown or times out when a property is not verified.

Adapting Mopsa to SV-Comp's Framework

Our approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is <u>safe</u>
   - **No?** restart with a more precise analysis configuration

$\rightsquigarrow$ Mopsa returns unknown or times out when a property is not verified.

Max. Conf.	Tasks proved correct		Tasks reaching 900s timeout	
1	7002		389	
2	7743	(+741)	970	(+581)
3	8489	(+746)	3377	(+2407)
4	8660	(+171)	5378	(+2001)
5	8933	(+273)	8440	(+3062)

Improvements made in 2025

- Heuristic Autosuggestions

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds
- Memory deallocation

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds
- Memory deallocation
- Sound bitfield support

Our 2025 improvements

Category	Prop.	|tasks|	Mopsa'24	Mopsa'25	Best score, verifier (2025)	
Hardness	R	4012	432	518	7426	SVF-SVC [CMY25]
Heap	R	240	190	226	314	PredatorHP [PSV20]
Loops	R	774	298	376	1031	AISE [WC24; YZJ25]
Recursive	R	160	12	60	150	UTaipan [Die+23]
Heap	M	247	40	154	331	PredatorHP [PSV20]
Juliet	M	3271	2224	2530	4709	CPAchecker [Bai+24]
LinkedLists	M	134	58	96	220	PredatorHP [PSV20]
Main	N	1989	1920	2138	2756	UAutomizer [Hei+23]
AWS	R	341	36	76	326	Bubaak [CH23; MC25]
DDL	R	2420	3476	3602	3602	Mopsa
uthash	M	192	96	108	246	Bubaak* [CH23; MC25; CR24]
uthash	N	162	204	300	300	Mopsa

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

- Unable to provide counterexamples yet

ongoing work by Marco Milanese [MM24]

Strengths & Weaknesses

Strengths

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

Weaknesses

- Unable to provide counterexamples yet

  ongoing work by Marco Milanese [MM24]
- Fixed sequence of configurations

Preprocessing Complex Programs

- `mopsa-build`

Preprocessing Complex Programs

- `mopsa-build`
  - Records compiler/linker calls and their options

Preprocessing Complex Programs

- `mopsa-build`
    - Records compiler/linker calls and their options
    - Creates a compilation database

Preprocessing Complex Programs

- `mopsa-build`
  - Records compiler/linker calls and their options
  - Creates a compilation database

$\rightsquigarrow$ `mopsa-build make` drop-in replacement for `make`

Preprocessing Complex Programs

- `mopsa-build`
  - Records compiler/linker calls and their options
  - Creates a compilation database

$\rightsquigarrow$ `mopsa-build make` drop-in replacement for `make`

- `mopsa-c` leverages the compilation database

```
mopsa-c mopsa.db -make-target=fmt
```

Preprocessing Complex Programs

- `mopsa-build`
  - Records compiler/linker calls and their options
  - Creates a compilation database

$\leadsto$ `mopsa-build make` drop-in replacement for `make`

- `mopsa-c` leverages the compilation database

```
mopsa-c mopsa.db -make-target=fmt
```

- Option to generate a single, preprocessed file `-c-preprocess-and-exit`
  Used to generate preprocessed `coreutils` for SV-Comp.

References – I

References

[Bai+24] Daniel Baier et al. **“CPAchecker 2.3 with Strategy Selection - (Competition Contribution)”.** In: TACAS (3). Vol. 14572. Lecture Notes in Computer Science. Springer, 2024, pp. 359–364.

[CH23] Marek Chalupa and Thomas A. Henzinger. **“Bubaak: Runtime Monitoring of Program Verifiers - (Competition Contribution)”.** In: TACAS (2). Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 535–540.

References – II

[CMY25] C. McGowan, M. Richards, and Y. Sui. **“SVF-SVC: Software Verification Using SVF (Competition Contribution)”.** In: Proc. TACAS. LNCS. Springer, 2025.

[CR24] Marek Chalupa and Cedric Richter. **“Bubaak-SpLit: Split what you cannot verify (Competition contribution)”.** In: TACAS (3). Vol. 14572. Lecture Notes in Computer Science. Springer, 2024, pp. 353–358.

[Die+23] Daniel Dietsch et al. **“Ultimate Taipan and Race Detection in Ultimate - (Competition Contribution)”.** In: TACAS (2). Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 582–587.

[Hei+23] Matthias Heizmann et al. **“Ultimate Automizer and the CommuHash Normal Form - (Competition Contribution)”.** In: TACAS (2). Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 577–581.

References – III

[MC25] M. Chalupa and C. Richter. “BUBAAK: Dynamic Cooperative Verification (Competition Contribution)”. In: Proc. TACAS. LNCS. Springer, 2025.

[MM24] Marco Milanese and Antoine Miné. “Generation of Violation Witnesses by Under-Approximating Abstract Interpretation”. In: VMCAI. Springer, 2024.

[PSV20] Petr Peringer, Veronika Soková, and Tomás Vojnar. “PredatorHP Revamped (Not Only) for Interval-Sized Memory Regions and Memory Reallocation (Competition Contribution)”. In: TACAS (2). Vol. 12079. Lecture Notes in Computer Science. Springer, 2020, pp. 408–412.

References – IV

[WC24] Zhen Wang and Zhenbang Chen. **"AISE: A Symbolic Verifier by Synergizing Abstract Interpretation and Symbolic Execution (Competition Contribution)".** In: TACAS (3). Vol. 14572. Lecture Notes in Computer Science. Springer, 2024, pp. 347–352.

[YZJ25] Y. Lin, Z. Chen, and J. Wang. **"AISE v2.0: Combining Loop Transformations (Competition Contribution)".** In: Proc. TACAS. LNCS. Springer, 2025.