

Mopsa-C at SV-Comp 2024

Raphaël Monat, Marco Milanese, Francesco Parolini,
Jerôme Boillot, Abdelraouf Ouadjaout, Antoine Miné



Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer



Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer

Different properties

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



Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer

Different properties

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

Multiple languages

- ▶ C
- ▶ Python (+C)
- ▶ μ OCaml
- ▶ Michelson



Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer

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)
- ▶ μ OCaml
- ▶ Michelson



Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer

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)
- ▶ μ OCaml
- ▶ Michelson

Contributors (2018–2024)

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

Our approach

- 1 Analyze the target program with Mopsa

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 safe

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 safe
 - **No?** restart with a more precise analysis configuration

↪ 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 safe
 - **No?** restart with a more precise analysis configuration

↪ Mopsa returns unknown or times out when a property is not verified.

Max. Conf.	Tasks proved correct	Tasks yielding timeout
1	6995	368
2	7775 (+780)	717 (+349)
3	8197 (+422)	2954 (+2237)
4	8257 (+60)	3527 (+573)
5	8400 (+143)	9532 (+6005)

Improvements made in 2024

- ▶ Dynamic memory allocation

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme
 - Decreasing iterations added in that case

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme
 - Decreasing iterations added in that case
 - Rewriting specific cases into loops (improves precision)

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme
 - Decreasing iterations added in that case
 - Rewriting specific cases into loops (improves precision)
- ▶ Libc stubs

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme
 - Decreasing iterations added in that case
 - Rewriting specific cases into loops (improves precision)
- ▶ Libc stubs
 - Precise handling of `memset` of constant size

Improvements made in 2024

- ▶ Dynamic memory allocation
 - Based on the recency abstraction
 - Now avoids summarization during unrollings
- ▶ Integer abstractions
 - Constant exclusion domain
 - Simplification of expressions with overflows of Boillot and Feret
- ▶ Goto-based loops
 - AST-based iterations (compared to CFG), special fixpoint scheme
 - Decreasing iterations added in that case
 - Rewriting specific cases into loops (improves precision)
- ▶ Libc stubs
 - Precise handling of **memset** of constant size
 - **NULL** pointer synthesis from contiguous block of 0 bytes.

Our results – SoftwareSystems track

Category	Prop.	tasks	Mopsa'23	Mopsa'24	Best score (2024)	
AWS	R	197	32	36	137	Symbiotic
coreutils	M	140	0	0	0	–
coreutils	N	30	0	4	4	Mopsa
BusyBox	N	54	4	8	8	Mopsa
DDL	R	2442	3174	3476	3476	Mopsa
DDLL	R	8	10	14	14	Mopsa
DDL	M	141	0	8	71	Bubaak-SpLit
other	R	22	0	10	10	Mopsa
other	M	34	0	12	12	Mopsa
uthash	R	138	0	192	228	Bubaak*, Symbiotic
uthash	M	138	0	96	204	Bubaak*, Symbiotic
uthash	N	114	0	204	204	Mopsa

- ▶ Inject invariants in the program and verify this new program

- ▶ Inject invariants in the program and verify this new program
- ▶ Similar to Metaval's approach, with less changes on the original program

Beyer and Spiessl. "MetaVal: Witness Validation via Verification". CAV (2) 2020

- ▶ Inject invariants in the program and verify this new program
- ▶ Similar to Metaval's approach, with less changes on the original program
- ▶ Handy new YAML format!

Beyer and Spiessl. "MetaVal: Witness Validation via Verification". CAV (2) 2020

A word on witness validation

- ▶ Inject invariants in the program and verify this new program
- ▶ Similar to MetaVal's approach, with less changes on the original program
- ▶ Handy new YAML format!
- ▶ GoblinT has a much smarter approach

Beyer and Spiessl. "MetaVal: Witness Validation via Verification". CAV (2) 2020

Saan, Schwarz, Erhard, Seidl, Tilscher, and Vojdani. "Correctness Witness Validation by Abstract Interpretation". VCMAl 2024

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)
- ▶ Commitment to soundness, 20 verdict fixes

Strengths & Weaknesses

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)
- ▶ Commitment to soundness, 20 verdict fixes

Weaknesses

Strengths & Weaknesses

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)
- ▶ Commitment to soundness, 20 verdict fixes

Weaknesses

- ▶ Fixed sequence of configurations

Strengths & Weaknesses

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)
- ▶ Commitment to soundness, 20 verdict fixes

Weaknesses

- ▶ Fixed sequence of configurations
- ▶ Unable to provide counterexamples

Strengths & Weaknesses

Strengths

- ▶ Scalability (even for DeviceDriversLinux-Large)
- ▶ Commitment to soundness, 20 verdict fixes

Weaknesses

- ▶ Fixed sequence of configurations
- ▶ Unable to provide counterexamples
- ▶ Not competitive outside *SoftwareSystems*: array segmentation, partitioning?

Milanese and Miné. “Generation of Violation Witnesses by Under-Approximating Abstract Interpretation”.

VMCAI 2024

Mopsa-C at SV-Comp 2024

Raphaël Monat, Marco Milanese, Francesco Parolini,
Jerôme Boillot, Abdelraouf Ouadjaout, Antoine Miné