Mopsa at the Software Verification Competition

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné
`rmonat.fr`

SV-Comp
24 April 2023

Mopsa

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

[1] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

[1] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

- explore new designs, ease dev.

[1] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

- explore new designs, ease dev.
- support multiple languages

[1] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

- explore new designs, ease dev.
- support multiple languages
- support multiple properties

[1] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

- explore new designs, ease dev.
- support multiple languages
- support multiple properties
- loosely couple abstractions

[1] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019

Overview of Mopsa

Modular Open Platform for Static Analysis [1]

`gitlab.com/mopsa/mopsa-analyzer`

Goals

- explore new designs, ease dev.
- support multiple languages
- support multiple properties
- loosely couple abstractions

Contributors (2018–2023)

- Antoine Miné
- Abdelraouf Ouadjaout
- Raphaël Monat
- David Delmas
- Guillaume Bau
- Milla Valnet
- Matthieu Journault

[1] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019

Current public analyses in Mopsa

Based on abstract interpretation

Only proves programs correct

Semantic property

Runtime error detection

Current public analyses in Mopsa

Based on abstract interpretation

Only proves programs correct

Semantic property

Runtime error detection

$\simeq$ 50,000 lines of OCaml code

Current public analyses in Mopsa

Based on abstract interpretation

Only proves programs correct

Semantic property

Runtime error detection

$\simeq$ 50,000 lines of OCaml code

Language	Benchmark	Max. LoC	$\simeq$ Time	Selectivity
C[2]	Coreutils	550	20s	99.8%
	Juliet	340,000	2.5h	98.9%

[2] Ouadjaout and Miné. “A Library Modeling Language for the Static Analysis of C Programs”. SAS 2020

Current public analyses in Mopsa

Based on abstract interpretation

Only proves programs correct

Semantic property

Runtime error detection

$\simeq$ 50,000 lines of OCaml code

Language	Benchmark	Max. LoC	$\simeq$ Time	Selectivity
C[2]	Coreutils	550	20s	99.8%
	Juliet	340,000	2.5h	98.9%
Python[3]	PyPerformance	1,792	1.3m	99.2%
	PathPicker	2,560	3.0m	99.2%

$\frac{\text{\# safe operations}}{\text{\# operations}}$

[2] Ouadjaout and Miné. "A Library Modeling Language for the Static Analysis of C Programs". SAS 2020

[3] Monat, Ouadjaout, and Miné. "Static Type Analysis by Abstract Interpretation of Python Programs". ECOOP 2020

Current public analyses in Mopsa

Based on abstract interpretation

Only proves programs correct

Semantic property

Runtime error detection

$\simeq$ 50,000 lines of OCaml code

Language	Benchmark	Max. LoC	$\simeq$ Time	Selectivity
C[2]	Coreutils	550	20s	99.8%
	Juliet	340,000	2.5h	98.9%
Python[3]	PyPerformance	1,792	1.3m	99.2%
	PathPicker	2,560	3.0m	99.2%
Python+C[4]	ahocorasick	4,800	1.0m	98.0%
	bitarray	5,700	4.6m	94.6%

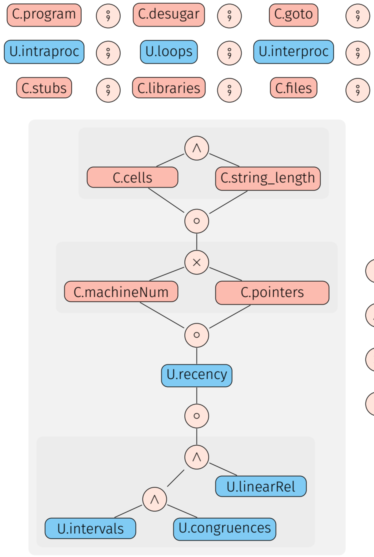
$\frac{\text{\# safe operations}}{\text{\# operations}}$

[2]Ouadjaout and Miné. “A Library Modeling Language for the Static Analysis of C Programs”. SAS 2020

[3]Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020

[4]Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021

Example configuration in Mopsa



⨾ Sequence

∧ Reduced product

× Cartesian product

∘ Composition

Mopsa at SV-Comp

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!

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

Suboptimal strategy

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

Suboptimal strategy

- Task: decide if a property holds on a program

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

Suboptimal strategy

- Task: decide if a property holds on a program
  But Mopsa analyzes full programs and detects <u>all</u> runtime errors

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

Suboptimal strategy

- Task: decide if a property holds on a program

  But Mopsa analyzes full programs and detects <u>all</u> runtime errors

  $\Longrightarrow$ We could at least add slicing

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

Suboptimal strategy

- Task: decide if a property holds on a program

  But Mopsa analyzes full programs and detects <u>all</u> runtime errors

  $\implies$ We could at least add slicing
- New analyses restart from scratch

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized
2. + string-length domain, medium structs initialized

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized
2. + string-length domain, medium structs initialized
3. + polyhedra with static packing

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized
2. + string-length domain, medium structs initialized
3. + polyhedra with static packing
4. + congruences & widening tweaks: thresholds, delay

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized
2. + string-length domain, medium structs initialized
3. + polyhedra with static packing
4. + congruences & widening tweaks: thresholds, delay

Conf.	✓		🕒	
1	5695		279	
2	6433	(+738)	365	(+86)
3	6885	(+452)	1844	(+1479)
4	6909	(+24)	2009	(+165)

21220 tasks in total, 12636 correctness tasks

Portfolio of analyses used

Increasingly precise analyses

1. Intervals, small structs initialized
2. + string-length domain, medium structs initialized
3. + polyhedra with static packing
4. + congruences & widening tweaks: thresholds, delay

Conf.	✓		🕒	
1	5695		279	
2	6433	(+738)	365	(+86)
3	6885	(+452)	1844	(+1479)
4	6909	(+24)	2009	(+165)

21220 tasks in total, 12636 correctness tasks

Mopsa validates 54% of correct tasks (61% for overall winner, UAutomizer).

Mopsa's Results

For our first participation, we competed in

- *ReachSafety*,

Mopsa’s Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,

Mopsa’s Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,
- *NoOverflows*

Mopsa's Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,
- *NoOverflows* ranking 6/19,

Mopsa’s Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,
- *NoOverflows* ranking 6/19,
- *SoftwareSystems*

Mopsa's Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,
- *NoOverflows* ranking 6/19,
- *SoftwareSystems* bronze medal!

Mopsa's Results

For our first participation, we competed in

- *ReachSafety*,
- *MemorySafety*,
- *NoOverflows* ranking 6/19,
- *SoftwareSystems* bronze medal!

Results in the *SoftwareSystems* category

Verifier	Bubaak	CPAchecker	Goblint	Mopsa	Symbiotic	Ultimate
Proved correct	291	**1,651**	1,256	1,610	942	1,423
Proved incorrect	143	59	0	0	84	2
CPU Time (s)	2,000,000	730,000	800,000	580,000	**400,000**	1,400,000
Rank	2	6	10	3	**1**	7

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

- Uncomplete.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

- Uncomplete.
- Lack of precision on small, intricate programs.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

- Uncomplete.
- Lack of precision on small, intricate programs.
- Trivial witness generation:

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

- Uncomplete.
- Lack of precision on small, intricate programs.
- Trivial witness generation:
  - 96.4% are validated.

Strengths and Weaknesses

- Scalability. Answers in 98.5% of the cases.
- Soundness. Fixed 164 task definitions.
- Good precision for an abstract interpreter.

- Uncomplete.
- Lack of precision on small, intricate programs.
- Trivial witness generation:
  - 96.4% are validated.
  - Difficulties with interprecodural encoding[5]?

[5] Saan. Witness Generation for Data-flow Analysis. 2020

Benefits of participation

- Fun! (up-to exhaustion)

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist
- Interaction and comparison with other tools from a broad community

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist
- Interaction and comparison with other tools from a broad community
- Better understanding of the benchmarks

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist
- Interaction and comparison with other tools from a broad community
- Better understanding of the benchmarks
  - Becoming a de facto standard

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist
- Interaction and comparison with other tools from a broad community
- Better understanding of the benchmarks
  - Becoming a de facto standard
  - Always ongoing benchmark curation

Benefits of participation

- Fun! (up-to exhaustion)
- Good time for software improvements
  - 20 issues fixed
  - We already have a 2024 feature wishlist
- Interaction and comparison with other tools from a broad community
- Better understanding of the benchmarks
  - Becoming a de facto standard
  - Always ongoing benchmark curation
- Brings new research questions

Conclusion

Conclusion

Mopsa as a stable academic static analyzer,

Conclusion

Mopsa as a stable academic static analyzer,
able to analyze C and Python programs,

Conclusion

Mopsa as a stable academic static analyzer,
able to analyze C and Python programs,
competing with cutting-edge verifiers.

Conclusion

Mopsa as a stable academic static analyzer,
able to analyze C and Python programs,
competing with cutting-edge verifiers.

Looking forward to the next editions!

Conclusion

Mopsa as a stable academic static analyzer,
able to analyze C and Python programs,
competing with cutting-edge verifiers.

Looking forward to the next editions!

Some SV-Comp related research questions

- Best configuration to analyze a given program under resource constraints

Conclusion

Mopsa as a stable academic static analyzer,
able to analyze C and Python programs,
competing with cutting-edge verifiers.

Looking forward to the next editions!

Some SV-Comp related research questions

- Best configuration to analyze a given program under resource constraints
- Targeting falsification tasks: synergy with symbolic execution, or backward analysis

Mopsa at the Software Verification Competition
Questions

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné
`rmonat.fr`

SV-Comp
24 April 2023