Mopsa at the Software Verification Competition

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné
rmonat.fr

SV-Comp
24 April 2023
Mopsy
Overview of Mopsa

Modular Open Platform for Static Analysis

gitlab.com/mopsa/mopsa-analyzer

1Journault, 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**

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 Open Platform for Static Analysis**

[gitlab.com/mopsa/mopsa-analyzer](http://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 Open Platform for Static Analysis

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

Goals

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

---

1Journault, 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**

[gitlab.com/mopsa/mopsa-analyzer](https://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 Open Platform for Static Analysis

[Link to GitLab repository: 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

Overview of Mopsa

**Modular Open Platform for Static Analysis**

[gitlab.com/mopsa/mopsa-analyzer](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

<table>
<thead>
<tr>
<th>Based on abstract interpretation</th>
<th>Semantic property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only proves programs correct</td>
<td>Runtime error detection</td>
</tr>
</tbody>
</table>

≃ 50,000 lines of OCaml code

Language Benchmark Max. LoC

<table>
<thead>
<tr>
<th>Language</th>
<th>Benchmark</th>
<th>Max. Lines</th>
<th>Time</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Coreutils</td>
<td>550</td>
<td>20s</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>Juliet</td>
<td>340,000</td>
<td>2.5h</td>
<td>98.9%</td>
</tr>
<tr>
<td>Python</td>
<td>PyPerformance</td>
<td>1,792</td>
<td>1.3m</td>
<td>99.2%</td>
</tr>
<tr>
<td></td>
<td>PathPicker</td>
<td>2,560</td>
<td>3.0m</td>
<td>99.2%</td>
</tr>
<tr>
<td>Python+C</td>
<td>ahocorasick</td>
<td>4,800</td>
<td>1.0m</td>
<td>98.0%</td>
</tr>
<tr>
<td></td>
<td>bitarray</td>
<td>5,700</td>
<td>4.6m</td>
<td>94.6%</td>
</tr>
</tbody>
</table>

# safe operations

2


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
Current public analyses in Mopsa

<table>
<thead>
<tr>
<th>Based on abstract interpretation</th>
<th>Semantic property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only proves programs correct</td>
<td>Runtime error detection</td>
</tr>
</tbody>
</table>

≅ 50,000 lines of OCaml code

# safe operations
# operations


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

Monat, Ouadjaout, and Miné. "A Multilanguage Static Analysis of Python Programs with Native C Extensions". SAS 2021
Current public analyses in Mopsa

Based on abstract interpretation  
Only proves programs correct

Semantic property  
Runtime error detection

≈ 50,000 lines of OCaml code

<table>
<thead>
<tr>
<th>Language</th>
<th>Benchmark</th>
<th>Max. LoC</th>
<th>≃ Time</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C^2$</td>
<td>Coreutils</td>
<td>550</td>
<td>20s</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>Juliet</td>
<td>340,000</td>
<td>2.5h</td>
<td>98.9%</td>
</tr>
</tbody>
</table>

Current public analyses in Mopsa

<table>
<thead>
<tr>
<th>Based on abstract interpretation</th>
<th>Semantic property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only proves programs correct</td>
<td>Runtime error detection</td>
</tr>
</tbody>
</table>

≃ 50,000 lines of OCaml code

<table>
<thead>
<tr>
<th>Language</th>
<th>Benchmark</th>
<th>Max. LoC</th>
<th>≃ Time</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>C²</td>
<td>Coreutils</td>
<td>550</td>
<td>20s</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>Juliet</td>
<td>340,000</td>
<td>2.5h</td>
<td>98.9%</td>
</tr>
<tr>
<td>Python³</td>
<td>PyPerformance</td>
<td>1,792</td>
<td>1.3m</td>
<td>99.2%</td>
</tr>
<tr>
<td></td>
<td>PathPicker</td>
<td>2,560</td>
<td>3.0m</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

3 Monat, Ouadiaout, 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

≃ 50,000 lines of OCaml code

<table>
<thead>
<tr>
<th>Language</th>
<th>Benchmark</th>
<th>Max. LoC</th>
<th>≃ Time</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>C²</td>
<td>Coreutils</td>
<td>550</td>
<td>20s</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>Juliet</td>
<td>340,000</td>
<td>2.5h</td>
<td>98.9%</td>
</tr>
<tr>
<td>Python³</td>
<td>PyPerformance</td>
<td>1,792</td>
<td>1.3m</td>
<td>99.2%</td>
</tr>
<tr>
<td></td>
<td>PathPicker</td>
<td>2,560</td>
<td>3.0m</td>
<td>99.2%</td>
</tr>
<tr>
<td>Python+C⁴</td>
<td>ahocorasick</td>
<td>4,800</td>
<td>1.0m</td>
<td>98.0%</td>
</tr>
<tr>
<td></td>
<td>bitarray</td>
<td>5,700</td>
<td>4.6m</td>
<td>94.6%</td>
</tr>
</tbody>
</table>

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

- C.program
- C.desugar
- C.goto
- U.intraproc
- U.loops
- U.interproc
- C.stubs
- C.libraries
- C.files
- ∧ C.cells
- C.string_length
- ◦ × C.machineNum
- ◦ U.recency
- ◦ Composition
- ◦ Reduced product
- × Cartesian product
- △ Sequence
- U.intervals
- U.congruences
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!

Suboptimal strategy

Task: decide if a property holds on a program

But Mopsa analyzes full programs and detects all runtime errors

We could at least add slicing

New analyses restart from scratch
### 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
### 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 all runtime errors... We could at least add slicing... New analyses restart from scratch...
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 all 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 all runtime errors
  → 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 all runtime errors
  ➔ 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

<table>
<thead>
<tr>
<th></th>
<th>Mopsa validates correct tasks</th>
<th>Overall winner, UAutomizer validating correct tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5695</td>
<td>61%</td>
</tr>
<tr>
<td>2</td>
<td>6433 (+738)</td>
<td>54%</td>
</tr>
<tr>
<td>3</td>
<td>6885 (+452)</td>
<td>54%</td>
</tr>
<tr>
<td>4</td>
<td>6909 (+24)</td>
<td>54%</td>
</tr>
</tbody>
</table>

21,220 tasks in total, 12,636 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

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

<table>
<thead>
<tr>
<th>Conf.</th>
<th>✔</th>
<th>✕</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5695</td>
<td>279</td>
</tr>
<tr>
<td>2</td>
<td>6433 (+738)</td>
<td>365 (+86)</td>
</tr>
<tr>
<td>3</td>
<td>6885 (+452)</td>
<td>1844 (+1479)</td>
</tr>
<tr>
<td>4</td>
<td>6909 (+24)</td>
<td>2009 (+165)</td>
</tr>
</tbody>
</table>

21220 tasks in total, 12636 correctness tasks

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

<table>
<thead>
<tr>
<th>Conf.</th>
<th>✓</th>
<th>🔴</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5695</td>
<td>279</td>
</tr>
<tr>
<td>2</td>
<td>6433 (+738)</td>
<td>365 (+86)</td>
</tr>
<tr>
<td>3</td>
<td>6885 (+452)</td>
<td>1844 (+1479)</td>
</tr>
<tr>
<td>4</td>
<td>6909 (+24)</td>
<td>2009 (+165)</td>
</tr>
</tbody>
</table>

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*
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

- \textit{ReachSafety},
- \textit{MemorySafety},
- \textit{NoOverflows} ranking 6/19,
- \textit{SoftwareSystems} bronze medal!
Mopsa’s Results

For our first participation, we competed in

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

<table>
<thead>
<tr>
<th>Verifier</th>
<th>Bubaak</th>
<th>CPAchecker</th>
<th>Goblint</th>
<th>Mopsa</th>
<th>Symbiotic</th>
<th>Ultimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proved correct</td>
<td>291</td>
<td>1,651</td>
<td>1,256</td>
<td>1,610</td>
<td>942</td>
<td>1,423</td>
</tr>
<tr>
<td>Proved incorrect</td>
<td>143</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>84</td>
<td>2</td>
</tr>
<tr>
<td>CPU Time (s)</td>
<td>2,000,000</td>
<td>730,000</td>
<td>800,000</td>
<td>580,000</td>
<td>400,000</td>
<td>1,400,000</td>
</tr>
<tr>
<td>Rank</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
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 interpredural encoding

---

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