Mopsa-C at SV-Comp 2024

Raphaël Monat, Marco Milanese, Francesco Parolini, Jerôme Boillot, Abdelraouf Ouadjaout, Antoine Miné
A General Overview of Mopsa

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)
▶ µOCaml
▶ Michelson

Contributors (2018–2024)
▶ G. Bau
▶ J. Boillot
▶ D. Delmas
▶ A. Fromherz
▶ M. Milanese
▶ A. Miné
▶ R. Monat
▶ A. Ouadjaout
▶ M. Journault
▶ F. Parolini
▶ M. Valnet
A General Overview of Mopsa

Modular Open Platform for Static Analysis
gitlab.com/mopsa/mopsa-analyzer

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

Different properties

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

Multiple languages

- C
- Python (+C)
- μOCaml
- Michelson
# A General Overview of Mopsa

## Modular Open Platform for Static Analysis

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

<table>
<thead>
<tr>
<th>Different properties</th>
<th>Specificities</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Runtime error detection</td>
<td></td>
</tr>
<tr>
<td>▶ Portability (patch, endianness)</td>
<td></td>
</tr>
<tr>
<td>▶ Non-exploitability</td>
<td></td>
</tr>
<tr>
<td>▶ Modular abstractions, loose coupling</td>
<td></td>
</tr>
<tr>
<td>▶ Optimized for relational domains</td>
<td></td>
</tr>
<tr>
<td>▶ Ease dev.: interactive engine, hooks</td>
<td></td>
</tr>
</tbody>
</table>

## Multiple languages

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

Contributors (2018–2024):
- G. Bau
- J. Boillot
- D. Delmas
- A. Fromherz
- M. Milanese
- A. Miné
- R. Monat
- A. Ouadjaout
- M. Journault
- F. Parolini
- M. Valnet
A General Overview of Mopsa

**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)
- µOCaml
- Michelson

### Contributors (2018–2024)
- G. Bau
- J. Boillot
- D. Delmas
- A. Fromherz
- M. Milanese
- A. Miné
- R. Monat
- M. Journault
- F. Parolini
- M. Valnet
- A. Ouadjaout
Our approach

1. Analyze the target program with Mopsa

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)
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

<table>
<thead>
<tr>
<th>Max. Conf.</th>
<th>Tasks proved correct</th>
<th>Tasks yielding timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6995</td>
<td>368</td>
</tr>
<tr>
<td>2</td>
<td>7775 (+780)</td>
<td>717 (+349)</td>
</tr>
<tr>
<td>3</td>
<td>8197 (+422)</td>
<td>2954 (+2237)</td>
</tr>
<tr>
<td>4</td>
<td>8257 (+60)</td>
<td>3527 (+573)</td>
</tr>
<tr>
<td>5</td>
<td>8400 (+143)</td>
<td>9532 (+6005)</td>
</tr>
</tbody>
</table>
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

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)
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

Max. Conf. Tasks proved correct Tasks yielding timeout

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6995</td>
<td>368</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7775 (+780)</td>
<td>717 (+349)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8197 (+422)</td>
<td>2954 (+2237)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8257 (+60)</td>
<td>3527 (+573)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8400 (+143)</td>
<td>9532 (+6005)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

$\leadsto$ 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.

<table>
<thead>
<tr>
<th>Max. Conf.</th>
<th>Tasks proved correct</th>
<th>Tasks yielding timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6995</td>
<td>368</td>
</tr>
<tr>
<td>2</td>
<td>7775 (+780)</td>
<td>717 (+349)</td>
</tr>
<tr>
<td>3</td>
<td>8197 (+422)</td>
<td>2954 (+2237)</td>
</tr>
<tr>
<td>4</td>
<td>8257 (+60)</td>
<td>3527 (+573)</td>
</tr>
<tr>
<td>5</td>
<td>8400 (+143)</td>
<td>9532 (+6005)</td>
</tr>
</tbody>
</table>
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 \texttt{memset} of constant size
  - NULL pointer synthesis from contiguous block of 0 bytes.

Boillot and Feret. "Symbolic Transformation of Expressions in Modular Arithmetic". SAS 2023
Improvements made in 2024

- Dynamic memory allocation
  - Based on the recency abstraction

- 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.
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

---

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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

---

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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)

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
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.

Boillot and Feret. “Symbolic Transformation of Expressions in Modular Arithmetic”. SAS 2023
Our results – *SoftwareSystems* track

<table>
<thead>
<tr>
<th>Category</th>
<th>Prop.</th>
<th>tasks</th>
<th>Mopsa’23</th>
<th>Mopsa’24</th>
<th>Best score (2024)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>R</td>
<td>197</td>
<td>32</td>
<td>36</td>
<td>137 Symbiotic</td>
</tr>
<tr>
<td>coreutils</td>
<td>M</td>
<td>140</td>
<td>0</td>
<td>0</td>
<td>0 _</td>
</tr>
<tr>
<td>coreutils</td>
<td>N</td>
<td>30</td>
<td>0</td>
<td>4</td>
<td>4 Mopsa</td>
</tr>
<tr>
<td>BusyBox</td>
<td>N</td>
<td>54</td>
<td>4</td>
<td>8</td>
<td>8 Mopsa</td>
</tr>
<tr>
<td>DDL</td>
<td>R</td>
<td>2442</td>
<td>3174</td>
<td>3476</td>
<td>3476 Mopsa</td>
</tr>
<tr>
<td>DDDL</td>
<td>R</td>
<td>8</td>
<td>10</td>
<td>14</td>
<td>14 Mopsa</td>
</tr>
<tr>
<td>DDL</td>
<td>M</td>
<td>141</td>
<td>0</td>
<td>8</td>
<td>71 Bubaak-SpLit</td>
</tr>
<tr>
<td>other</td>
<td>R</td>
<td>22</td>
<td>0</td>
<td>10</td>
<td>10 Mopsa</td>
</tr>
<tr>
<td>other</td>
<td>M</td>
<td>34</td>
<td>0</td>
<td>12</td>
<td>12 Mopsa</td>
</tr>
<tr>
<td>uthash</td>
<td>R</td>
<td>138</td>
<td>0</td>
<td>192</td>
<td>228 Bubaak*, Symbiotic</td>
</tr>
<tr>
<td>uthash</td>
<td>M</td>
<td>138</td>
<td>0</td>
<td>96</td>
<td>204 Bubaak*, Symbiotic</td>
</tr>
<tr>
<td>uthash</td>
<td>N</td>
<td>114</td>
<td>0</td>
<td>204</td>
<td>204 Mopsa</td>
</tr>
</tbody>
</table>
A word on witness validation

- Inject invariants in the program and verify this new program
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

---

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!

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”. VCMAI 2024
Strengths & Weaknesses

Strengths

- Scalability (even for DeviceDriversLinux-Large)

Weaknesses

- Fixed sequence of configurations
- Unable to provide counterexamples
- Not competitive outside SoftwareSystems: array segmentation, partitioning?
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
## Strengths & Weaknesses

<table>
<thead>
<tr>
<th>Strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Scalability (even for DeviceDriversLinux-Large)</td>
</tr>
<tr>
<td>▶ Commitment to soundness, 20 verdict fixes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weaknesses</th>
</tr>
</thead>
</table>
## Strengths & Weaknesses

### Strengths

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

### Weaknesses

- Fixed sequence of configurations
## Strengths

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

## Weaknesses

- Fixed sequence of configurations
- Unable to provide counterexamples

---

Milanese and Miné. “Generation of Violation Witnesses by Under-Approximating Abstract Interpretation”. VMCAI 2024
## 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