Easing implementation & maintenance of academic static analyzers

Raphaël Monat, Abdelraouf Ouadjaout & Antoine Miné

rmonat.fr

CSV
6 June 2024

Inria

Introduction

Motivation

Academic research around static analysis

Academic research around static analysis

Ideal analyzer

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding
- Sound, precise and scalable

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding
- Sound, precise and scalable

Implementation hurdles

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding
- Sound, precise and scalable

Implementation hurdles

- Debugging time-consuming

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding
- Sound, precise and scalable

Implementation hurdles

- Debugging time-consuming
- Maintenance necessary to build upon previous work

Academic research around static analysis

Ideal analyzer

- Eases research:
  - Implementation
  - Experimental evaluation
  - Onboarding
- Sound, precise and scalable

Implementation hurdles

- Debugging time-consuming
- Maintenance necessary to build upon previous work

$\Longrightarrow$ Aiming for lowest possible implementation & maintenance costs

This talk

Since 2017 Development of the Mopsa static analysis platform

This talk

Since 2017 Development of the Mopsa static analysis platform

This talk shares our experience and approach in Mopsa

This talk

Since 2017 Development of the Mopsa static analysis platform

This talk shares our experience and approach in Mopsa

Afterwards continue the conversation and increase sharing of practices

This talk

Since 2017 Development of the Mopsa static analysis platform

This talk shares our experience and approach in Mopsa

Afterwards continue the conversation and increase sharing of practices

⚠ Some things might be folklore.

Introduction

Mopsa

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

`gitlab.com/mopsa/mopsa-analyzer`

Goals: explore new designs, ease development of (relational) analyses

[1] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. 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 development of (relational) analyses

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

[1] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. 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 development of (relational) analyses

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

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

Modular Open Platform for Static Analysis[1]

gitlab.com/mopsa/mopsa-analyzer

Goals: explore new designs, ease development of (relational) analyses

One AST to rule them all

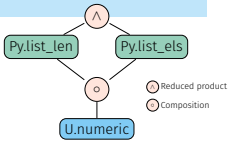
- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation



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

Works around Mopsa

Languages

C, Python

Multilanguage Python+C

WIP: Michelson, OCaml, ...

Works around Mopsa

Languages

C, Python

Multilanguage Python+C

WIP: Michelson, OCaml, ...

Properties

- **Absence of RTEs**
- Patch analysis
- Endianness portability
- Non-exploitability

Works around Mopsa

Languages

C, Python

Multilanguage Python+C

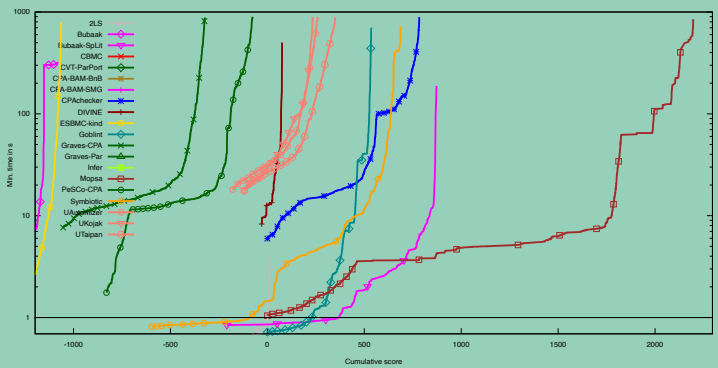
WIP: Michelson, OCaml, ...

Properties

- **Absence of RTEs**
- Patch analysis
- Endianness portability
- Non-exploitability

Software Verification Competition

We won the “SoftwareSystems” track of SV-Comp 2024!



Outline

1. Avoiding regressions
   - Measuring precision
   - Comparing analysis reports
2. Easing debugging
   - Developer-friendly interfaces
   - Testcase reduction
3. A plug-in system of analysis observers

Avoiding regressions

Measuring precision

The easy case of benchmarks with precision oracles

Benchmarks with precision oracles

- Know whether a given alarm should be raised
- Can provide absolute precision measure
- Based on manual analysis, not scalable
- NIST's Juliet Benchmarks, SV-Comp labeling of tasks (coarse)

The easy case of benchmarks with precision oracles

Benchmarks with precision oracles

- Know whether a given alarm should be raised
- Can provide absolute precision measure
- Based on manual analysis, not scalable
- NIST's Juliet Benchmarks, SV-Comp labeling of tasks (coarse)

Otherwise: relative precision measures.

A good precision measure

Precision measures

Is the program safe?	Too coarse

A good precision measure

Precision measures

Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?

A good precision measure

Precision measures	
Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?
Number of alarms	Difficult to put in context “2,756 alarms”
	True vs false alarms often requires manual work

A good precision measure

Precision measures

Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?
Number of alarms	Difficult to put in context “2,756 alarms”
	True vs false alarms often requires manual work

Our approach

Reporting status of all proofs / checks in every analyzed context

$$\text{Selectivity} = \frac{\#\text{checks proved safe}}{\#\text{checks}}$$

A good precision measure

Precision measures

Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?
Number of alarms	Difficult to put in context “2,756 alarms”
	True vs false alarms often requires manual work

Our approach

Reporting status of all proofs / checks in every analyzed context

$$\text{Selectivity} = \frac{\#\text{checks proved safe}}{\#\text{checks}}$$

```
1 int main() {
2   int n;
3   int y = -1;
4   for(int x = 0; x < n; x++)
5     y++;
6 }
```

Statement	
x++	
y++	
Selectivity	

A good precision measure

Precision measures

Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?
Number of alarms	Difficult to put in context “2,756 alarms”
	True vs false alarms often requires manual work

Our approach

Reporting status of all proofs / checks in every analyzed context

$$\text{Selectivity} = \frac{\#\text{checks proved safe}}{\#\text{checks}}$$

```
1 int main() {
2   int n;
3   int y = -1;
4   for(int x = 0; x < n; x++)
5     y++;
6 }
```

Statement	Intervals
`x++`	Safe
`y++`	Alarm
Selectivity	50%

A good precision measure

Precision measures

Is the program safe?	Too coarse
Measuring abstract states	Difficult; which states?
Number of alarms	Difficult to put in context “2,756 alarms”
	True vs false alarms often requires manual work

Our approach

Reporting status of all proofs / checks in every analyzed context

$$\text{Selectivity} = \frac{\#\text{checks proved safe}}{\#\text{checks}}$$

```
1 int main() {
2   int n;
3   int y = -1;
4   for(int x = 0; x < n; x++)
5     y++;
6 }
```

Statement	Intervals	Relational
`x++`	Safe	Safe
`y++`	Alarm	Safe
Selectivity	50%	100%

A good precision measure (II)

Benefits of the approach

- Easy to implement

A good precision measure (II)

Benefits of the approach

- Easy to implement
- Enhances transparency of the analysis

A good precision measure (II)

Benefits of the approach

- Easy to implement
- Enhances transparency of the analysis
- “2,756 alarms” $\rightsquigarrow$ 87% checks proved correct – “selectivity”

A good precision measure (II)

Benefits of the approach

- Easy to implement
- Enhances transparency of the analysis
- “2,756 alarms” $\rightsquigarrow$ 87% checks proved correct – “selectivity”
- ~~Program size~~ $\rightsquigarrow$ “expression complexity”

A good precision measure (II)

Benefits of the approach

- Easy to implement
- Enhances transparency of the analysis
- “2,756 alarms” ⇝ 87% checks proved correct – “selectivity”
- ~~Program size~~ ⇝ “expression complexity”

Analysis of coreutils `fmt`

```
Checks summary: 21247 total, ✔ 18491 safe, ✗ 129 errors, ⚠ 2627 warnings
  Stub condition: 690 total, ✔ 513 safe, ✗ 3 errors, ⚠ 174 warnings
  Invalid memory access: 8139 total, ✔ 7142 safe, ✗ 4 errors, ⚠ 993 warnings
  Division by zero: 499 total, ✔ 445 safe, ⚠ 54 warnings
  Integer overflow: 11581 total, ✔ 10177 safe, ⚠ 1404 warnings
  Invalid shift: 163 total, ✔ 163 safe
  Invalid pointer comparison: 37 total, ✗ 37 errors
  Invalid pointer subtraction: 85 total, ✗ 85 errors
  Insufficient variadic arguments: 1 total, ✔ 1 safe
  Insufficient format arguments: 26 total, ✔ 25 safe, ⚠ 1 warning
  Invalid type of format argument: 26 total, ✔ 25 safe, ⚠ 1 warning
```

Avoiding regressions

Comparing analysis reports

Comparing analysis reports

`mopsa-diff` script, used to compare:

- analysis report(s): either single output or set of outputs
- usecases: different configurations, different versions of Mopsa

Comparing analysis reports

`mopsa-diff` script, used to compare:

- analysis report(s): either single output or set of outputs
- usecases: different configurations, different versions of Mopsa

```
--- baseline/touch-many-symbolic-args-a4.json
+++ pplite/touch-many-symbolic-args-a4.json

- time: 589.0760
+ time: 675.1761

+ parse-datetime.y:1399.44-46: alarm: Invalid memory access
- parse-datetime.y:965.56-71: alarm: Invalid memory access
- parse-datetime.y:980.25-52: alarm: Invalid memory access
- parse-datetime.y:1003.23-50: alarm: Invalid memory access
- parse-datetime.y:921.56-71: alarm: Invalid memory access
- parse-datetime.c:1733.2-8: alarm: Invalid memory access
- parse-datetime.y:781.26-41: alarm: Invalid memory access
- parse-datetime.y:772.23-38: alarm: Invalid memory access
- parse-datetime.y:755.23-38: alarm: Invalid memory access
- parse-datetime.y:973.25-52: alarm: Invalid memory access
- parse-datetime.y:610.8-41: alarm: Invalid memory access
- parse-datetime.y:743.25-40: alarm: Invalid memory access
```

Comparing analysis reports

`mopsa-diff` script, used to compare:

- analysis report(s): either single output or set of outputs
- usecases: different configurations, different versions of Mopsa

```
--- baseline/touch-many-symbolic-args-a4.json
+++ pplite/touch-many-symbolic-args-a4.json

- time: 589.0760
+ time: 675.1761

+ parse-datetime.y:1399.44-46: alarm: Invalid memory access
- parse-datetime.y:965.56-71: alarm: Invalid memory access
- parse-datetime.y:980.25-52: alarm: Invalid memory access
- parse-datetime.y:1003.23-50: alarm: Invalid memory access
- parse-datetime.y:921.56-71: alarm: Invalid memory access
- parse-datetime.c:1733.2-8: alarm: Invalid memory access
- parse-datetime.y:781.26-41: alarm: Invalid memory access
- parse-datetime.y:772.23-38: alarm: Invalid memory access
- parse-datetime.y:755.23-38: alarm: Invalid memory access
- parse-datetime.y:973.25-52: alarm: Invalid memory access
- parse-datetime.y:610.8-41: alarm: Invalid memory access
- parse-datetime.y:743.25-40: alarm: Invalid memory access
```

```
139 reports compared
avg. time change       +52.065s
avg. speedup               -36%
new alarms                   2
removed alarms              32
new assumptions              0
removed assumptions          0
new successes                0
new failures                 0
```

A word on continuous integration (CI)

Detecting breaking changes in the CI

- `mopsa-diff` to compare with previous results

A word on continuous integration (CI)

Detecting breaking changes in the CI

- `mopsa-diff` to compare with previous results
- Reusing all benchmarks from our experimental evaluations

A word on continuous integration (CI)

Detecting breaking changes in the CI

- `mopsa-diff` to compare with previous results
- Reusing all benchmarks from our experimental evaluations

Language	Benchmark	Max. LoC	≃ Time	Selectivity
C[1]	Coreutils	550	20s	99.8%
	Juliet	340,000	2.5h	98.9%

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

A word on continuous integration (CI)

Detecting breaking changes in the CI

- `mopsa-diff` to compare with previous results
- Reusing all benchmarks from our experimental evaluations

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

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

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

A word on continuous integration (CI)

Detecting breaking changes in the CI

- `mopsa-diff` to compare with previous results
- Reusing all benchmarks from our experimental evaluations

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

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

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

[3] Monat, Ouadjaout, and Miné. "A Multilanguage Static Analysis of Python Programs with Native C Extensions". 2021

Easing debugging

Developer-friendly interfaces

Where static analyzers usually start from

- Analysis output

Too coarse

Where static analyzers usually start from

- Analysis output — Too coarse
- Printing abstract state using builtins — Not interactive

Where static analyzers usually start from

- Analysis output — Too coarse
- Printing abstract state using builtins — Not interactive
- Interpretation trace — Can be dozens of gigabytes of text

```
+ S [| set_program_name(argv[0]); |]
| | + S [| add(argv0)
| | |       argv0 = argv[0]; |]
| | | + S [| add(argv0) |]
| | | | + S [| add(argv0) |] in below(c.iterators.intraproc)
| | | | | + S [| add(argv0) |] in C/Scalar
| | | | | | + S [| add(offset⟦argv0⟧) |] in Universal
| | | | | | o S [| add(offset⟦argv0⟧) |] in Universal done [0.0001s, 1 case]
| | | | | o S [| add(argv0) |] in C/Scalar done [0.0001s, 1 case]
| | | | | + S [| add(argv0) |] in below(c.memory.lowlevel.cells)
| | | | | | + S [| add(offset⟦argv0⟧) |] in Universal
| | | | | | o S [| add(offset⟦argv0⟧) |] in Universal done [0.0001s, 1 case]
| | | | | o S [| add(argv0) |] in below(c.memory.lowlevel.cells) done [0.0001s, 1 case]
| | | | o S [| add(argv0) |] in below(c.iterators.intraproc) done [0.0001s, 1 case]
| | | o S [| add(argv0) |] done [0.0002s, 1 case]
| | | + S [| argv0 = argv[0]; |]
| | | | + S [| argv0 = (signed char *) @argv{0}:ptr; |] in below(c.iterators.intraproc)
| | | | | + S [| argv0 = (signed char *) @argv{0}:ptr; |] in C/Scalar
| | | | | | + S [| offset⟦argv0⟧ = (offset⟦@argv{0}:ptr⟧ + 0); |] in Universal
| | | | | | | + S [| offset⟦argv0⟧ = (offset⟦@argv{0}:ptr⟧ + 0); |] in below(universal.iterators.intraproc)
```

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm
- Navigation

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm
- Navigation
- Observation of the abstract state

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm
- Navigation
- Observation of the abstract state
  - Full state

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm
- Navigation
- Observation of the abstract state
  - Full state
  - Projection on specific variables

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location
  - Specific transfer function, analysis of subexpression
  - Alarm: jumping back to statement generating first alarm
- Navigation
- Observation of the abstract state
  - Full state
  - Projection on specific variables
- Some scripting capabilities

DAP support

- Sibling of Language Server Protocol (LSP)

DAP support

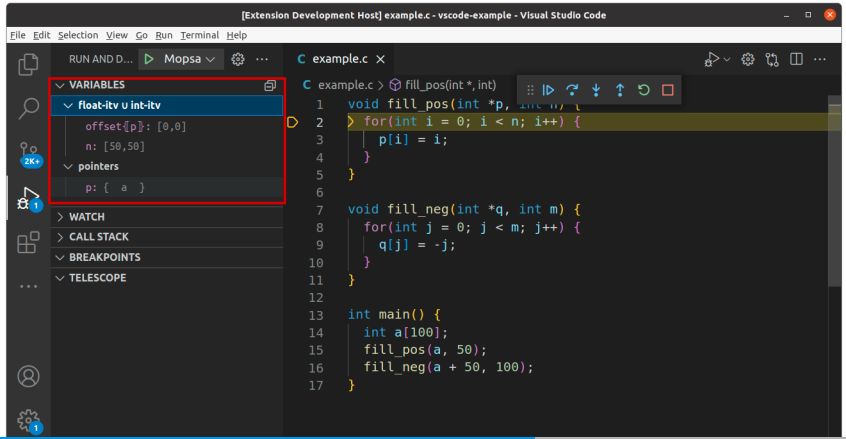
- Sibling of Language Server Protocol (LSP)
- Debug Adapter Protocol introduced by VSCode

DAP support

- Sibling of Language Server Protocol (LSP)
- Debug Adapter Protocol introduced by VSCode
- Features shared with interactive engine

DAP support

- Sibling of Language Server Protocol (LSP)
- Debug Adapter Protocol introduced by VSCode
- Features shared with interactive engine



Easing debugging

Testcase reduction

Testcase reduction

Motivation

Testcase reduction

Motivation

- Static analyzers are complex piece of code and may contain bugs

Testcase reduction

Motivation

- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases

Testcase reduction

Motivation

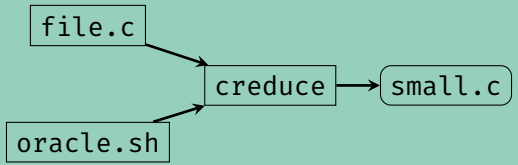
- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases
- Debugging extremely difficult: size of the program, analysis time

Testcase reduction

Motivation

- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases
- Debugging extremely difficult: size of the program, analysis time

Automated testcase reduction using `creduce`[1]



[1] Regehr, Chen, Cuoq, Eide, Ellison, and Yang. “Test-case reduction for C compiler bugs”. 2012

Testcase reduction (II)

Internal errors debugging

- Highly helpful to significantly reduce debugging time of runtime errors (Apron mishandlings, raised exceptions, ...)
- Has been applied to coreutils programs, SV-Comp programs of 10,000+ LoC

Testcase reduction (II)

Internal errors debugging

- Highly helpful to significantly reduce debugging time of runtime errors (Apron mishandlings, raised exceptions, ...)
- Has been applied to coreutils programs, SV-Comp programs of 10,000+ LoC

Differential-configuration debugging

```
$ mopsa-c -config=confA.json file.c
Alarm: assertion failure
$ mopsa-c -config=confB.json file.c
No alarm
```

Has been used to simplify cases in externally reported soundness issues

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

- `mopsa-build`

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

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

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

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

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

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

  ⇝ `mopsa-build make` drop-in replacement for `make`

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

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

  ⇝ `mopsa-build make` drop-in replacement for `make`
- `mopsa-c` leverages the compilation database

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

Handling multi-file projects

`creduce` limited to reducing a specific file

Mitigation: generate a pre-processed, standalone file

Painful operation on large projects such as `coreutils`

Mopsa supports multi-file C projects

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

  ⇝ `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

A plug-in system of analysis observers

Hooks: a plug-in system of analysis observers

Hooks	
Observe analyzer state	before/after any expression/statement analysis

Hooks: a plug-in system of analysis observers

Hooks

Observe analyzer state — before/after any expression/statement analysis

Current hooks

- Logs: trace of interpretation performed by the analysis

Hooks: a plug-in system of analysis observers

Hooks

Observe analyzer state — before/after any expression/statement analysis

Current hooks

- Logs: trace of interpretation performed by the analysis
- Thresholds for widening

Hooks: a plug-in system of analysis observers

Hooks

Observe analyzer state — before/after any expression/statement analysis

Current hooks

- Logs: trace of interpretation performed by the analysis
- Thresholds for widening
- Coverage

Hooks: a plug-in system of analysis observers

Hooks

Observe analyzer state — before/after any expression/statement analysis

Current hooks

- Logs: trace of interpretation performed by the analysis
- Thresholds for widening
- Coverage
- Heuristic unsoundness/imprecision detection

Hooks: a plug-in system of analysis observers

Hooks

Observe analyzer state — before/after any expression/statement analysis

Current hooks

- Logs: trace of interpretation performed by the analysis
- Thresholds for widening
- Coverage
- Heuristic unsoundness/imprecision detection
- Profiling

Coverage hooks

Coverage

- Global metric for the analysis' results
- Good to detect issues in the instrumentation of the fully context-sensitive analysis

No symbolic argument

```
./src/coreutils-8.30/src/fmt.c:
    'main' 76% of 72 statements analyzed
    'set_prefix' 100% of 12 statements analyzed
    'same_para' 100% of 1 statement analyzed
    'get_line' 100% of 30 statements analyzed
    'fmt' 100% of 7 statements analyzed
    'base_cost' 100% of 16 statements analyzed
    'line_cost' 100% of 10 statements analyzed
    'get_prefix' 100% of 18 statements analyzed
```

Symbolic arguments

```
./src/coreutils-8.30/src/fmt.c:
    'main' 100% of 72 statements analyzed
```

Heuristic unsoundness/imprecision detection

Detection of unsound transfer functions

Bottom shouldn’t appear after some statements (such as assignments)

Detection of imprecise analysis

Warns when assignments to top are performed

Simplifies the search for sources of large imprecision

Profiling

Standard profiling

Measures which parts of Mopsa are the most time-consuming

Profiling

Standard profiling

Measures which parts of Mopsa are the most time-consuming

Abstract profiling hook

Measures which parts of the analyzed program are the most time-consuming

- Loop-level profiling
- Function-level profiling

Profiling

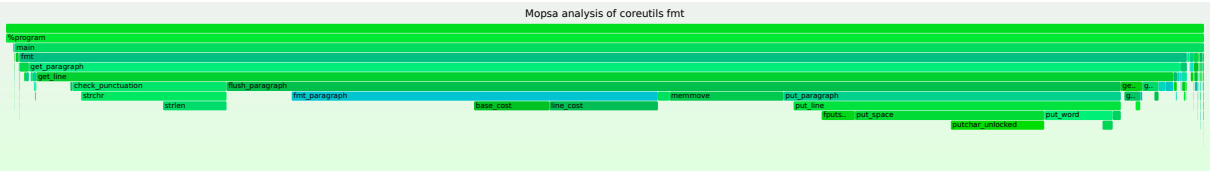
Standard profiling

Measures which parts of Mopsa are the most time-consuming

Abstract profiling hook

Measures which parts of the analyzed program are the most time-consuming

- Loop-level profiling
- Function-level profiling



Conclusion

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017

Related work

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017
- Frama-C & Goblint: flamegraphs, testcase reduction

Related work

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP: Luo, Dolby, and Bodden. “MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”. 2019

Related work

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP: Luo, Dolby, and Bodden. “MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”. 2019
- Klinger, Christakis, and Wüstholz. “Differentially testing soundness and precision of program analyzers”. 2019

Related work

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP: Luo, Dolby, and Bodden. “MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”. 2019
- Klinger, Christakis, and Wüstholz. “Differentially testing soundness and precision of program analyzers”. 2019
- Taneja, Liu, and Regehr. “Testing static analyses for precision and soundness”. 2020

Related work

Lots of folklore

- Andreasen, Møller, and Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. 2017
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP: Luo, Dolby, and Bodden. “MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”. 2019
- Klinger, Christakis, and Wüstholz. “Differentially testing soundness and precision of program analyzers”. 2019
- Taneja, Liu, and Regehr. “Testing static analyses for precision and soundness”. 2020
- Molle, Vandenbogaerde, and Roover. “Cross-Level Debugging for Static Analysers”. 2023

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa "std. tools on the concrete execution of the *abstract interpreter*"

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Ongoing challenges

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Ongoing challenges

- Handling the exponential number of configurations

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Ongoing challenges

- Handling the exponential number of configurations
- Code maintenance time is still high (for me)

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Ongoing challenges

- Handling the exponential number of configurations
- Code maintenance time is still high (for me)
- Onboarding material

Conclusion

Our current approach

- Non-regression testing of soundness & precision. CI on real-world software.
- Combination of existing techniques and new tools to debug & profile Mopsa
  "std. tools on the concrete execution of the *abstract interpreter*"
  $\rightsquigarrow$ "new tools on abstract execution of *target program*"

Ongoing challenges

- Handling the exponential number of configurations
- Code maintenance time is still high (for me)
- Onboarding material
- Online availability, install-free tool testing