Easing implementation & maintenance of academic static analyzers

Raphaël Monat
SyCoMoRES team

`rmonat.fr`

SemAntique
20 September 2024

Introduction

Motivation

Academic research around static analysis

Motivation

Academic research around static analysis

Ideal analyzer

Motivation

Academic research around static analysis

Ideal analyzer

- Sound, precise and scalable

Motivation

Academic research around static analysis

Ideal analyzer

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

Motivation

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

Motivation

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

- Debugging time-consuming

Motivation

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

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

Motivation

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

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

⟹ 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

⚠ Experience report; some things might be folklore.

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

Overview of Mopsa

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`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

Overview of Mopsa

Modular Open Platform for Static Analysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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

One AST to rule them all

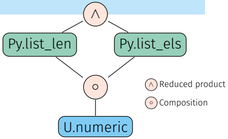
- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation



Contributors (2018–2024, chronological arrival order)

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

Contributors (2018–2024, chronological arrival order)

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

Maintainers in bold.

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

- Absence of RTEs

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]
- Non-exploitability [PM24]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]

Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

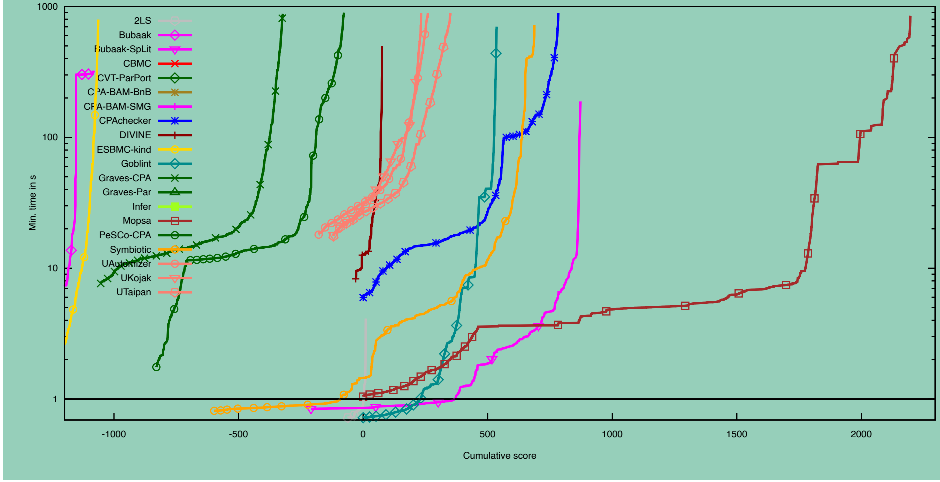
Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]
- Non-exploitability [PM24]
- Sufficient precondition inference [MM24]

Works around Mopsa – II

Software Verification Competition

We won the “SoftwareSystems” track of SV-Comp 2024 [Mon+24]!



Outline

1 Providing transparent analysis results

2 Avoiding regressions

3 Easing debugging

4 A plug-in system of analysis observers

Providing transparent analysis results

Raising the bar in static analyzer transparency

```
$ static-analysis-tool file
```

Raising the bar in static analyzer transparency

```
$ static-analysis-tool file
...
```

Raising the bar in static analyzer transparency

```
$ static-analysis-tool file
...
No errors found
```

Raising the bar in static analyzer transparency

```
$ static-analysis-tool file
...
No errors found
```

What has been checked? What has not?

Mopsa’s approach to being transparent – at a high level

```
if $a^\# \not\sqsubseteq p^\#$ then
  add_alarm $a^\#$ $p^\#$
```

Mopsa’s approach to being transparent – at a high level

```
if $a^\# \not\sqsubseteq p^\#$ then
  add_alarm $a^\#$ $p^\#$
```

$\rightsquigarrow$

```
if $a^\# \not\sqsubseteq p^\#$ then
  add_alarm $a^\#$ $p^\#$
else
  add_safe_check $p^\#$
```

Mopsa’s approach to being transparent – example

Mopsa’s approach to being transparent

- Reporting status of all proofs / checks in every analyzed context

Mopsa’s approach to being transparent – example

Mopsa’s approach to being transparent

- Reporting status of all proofs / checks in every analyzed context
- Quantitative precision measure

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

Mopsa’s approach to being transparent – example

Mopsa’s approach to being transparent

- Reporting status of all proofs / checks in every analyzed context
- Quantitative precision measure

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

```c
int main() {
  int n = _mopsa_rand_s32();
  int y = -1;
  for(int x = 0; x < n; x++)
    y++;
}
```

Mopsa’s approach to being transparent – example

Mopsa’s approach to being transparent

- Reporting status of all proofs / checks in every analyzed context
- Quantitative precision measure

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

```c
int main() {
  int n = _mopsa_rand_s32();
  int y = -1;
  for(int x = 0; x < n; x++)
    y++;
}
```

Stmt
`x++`
`y++`
Selectivity

Mopsa’s approach to being transparent – example

Mopsa’s approach to being transparent

- Reporting status of all proofs / checks in every analyzed context
- Quantitative precision measure

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

```c
int main() {
  int n = _mopsa_rand_s32();
  int y = -1;
  for(int x = 0; x < n; x++)
    y++;
}
```

Stmt	Itv
`x++`	Safe
`y++`	Alarm
Selectivity	50%

Mopsa's approach to being transparent – example

Mopsa's approach to being transparent

- Reporting status of all proofs / checks in every analyzed context
- Quantitative precision measure

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

```c
int main() {
  int n = _mopsa_rand_s32();
  int y = -1;
  for(int x = 0; x < n; x++)
    y++;
}
```

Stmt	Itv	Poly
x++	Safe	Safe
y++	Alarm	Safe
Selectivity	50%	100%

Mopsa’s approach to being transparent – output

Benefits of the approach

Mopsa's approach to being transparent – output

Benefits of the approach

- Easy to implement

Mopsa’s approach to being transparent – output

Benefits of the approach

- Easy to implement
- “2,756 alarms” ⇝ 87% checks proved correct – “selectivity”

Mopsa’s approach to being transparent – output

Benefits of the approach

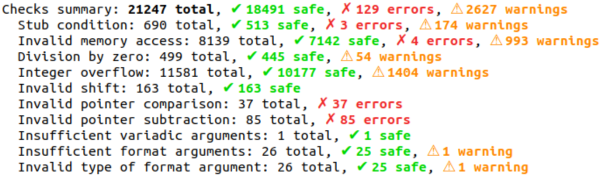
- Easy to implement
- “2,756 alarms” ⇝ 87% checks proved correct – “selectivity”
- ~~Program size~~ ⇝ “expression complexity”

Mopsa’s approach to being transparent – output

Benefits of the approach

- Easy to implement
- “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
```

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

```
extern int f(int *x)
```

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently ✗

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently ✗
3. Assume and report: f has no effect

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently ✗
3. Assume and report: f has no effect
4. Assume and report: f has any effect on its parameters

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently ✗
3. Assume and report: f has no effect
4. Assume and report: f has any effect on its parameters
5. Assume and report: f has any effect on its parameters and on globals

Mopsa’s approach to being transparent – soundness assumptions

Soundness assumptions, through an example

`extern int f(int *x)`, handling gradations

1. Crash ✗
2. Ignore silently ✗
3. Assume and report: f has no effect
4. Assume and report: f has any effect on its parameters
5. Assume and report: f has any effect on its parameters and on globals

Related topic: soundiness paper [Liv+15]

Avoiding regressions

Avoiding regressions

$\Longrightarrow$ check for precision changes

Avoiding regressions

$\Longrightarrow$ check for precision changes

Benchmarks with precision oracles

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

Avoiding regressions

$\Longrightarrow$ **check for precision changes**

Benchmarks with precision oracles

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

Otherwise: relative precision measures, rely on our selectivity computation.

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
```

CI, tests & benchmarks

Detecting breaking changes using continuous integration

- `mopsa-diff` to compare with previous results

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

- third-party real code

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

- third-party real code
- open-source – for the sake of reproducible science

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

- third-party real code
- open-source – for the sake of reproducible science
- unmodified*

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

- third-party real code
- open-source – for the sake of reproducible science
- unmodified*
  - Underscores practicality of our approach

CI, tests & benchmarks

Detecting breaking changes using continuous integration

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

Benchmark selection

Our benchmarks are

- third-party real code
- open-source – for the sake of reproducible science
- unmodified*
  - Underscores practicality of our approach
  - * stubs can be added in marginal cases

Some benchmarks

See SV-Comp 2024 results.

Some benchmarks

See SV-Comp 2024 results.

Benchmark	# Tests	Total LOC	Time	Precision
CWE121	2,508	234,930	3,064s	22.13%
CWE122	1,556	166,664	1,948s	25.84%
CWE124	758	93,372	961s	36.94%
CWE126	600	75,984	769s	46.83%
CWE127	758	89,022	963s	37.07%
CWE190	3,420	440,749	4,356s	78.13%
CWE191	2,622	340,884	3,236s	78.87%
CWE369	497	83,238	674s	70.42%
CWE415	190	17,990	228s	100.00%
CWE416	118	14,782	142s	67.80%
CWE469	18	1,520	22s	100.00%
CWE476	216	20,427	254s	100.00%

Table 1: Juliet benchmarks (non-relational configuration, no partitioning).

Some benchmarks

See SV-Comp 2024 results.

Benchmark	# Tests	Total LOC	Time	Precision
CWE121	2,508	234,930	3,064s	22.13%
CWE122	1,556	166,664	1,948s	25.84%
CWE124	758	93,372	961s	36.94%
CWE126	600	75,984	769s	46.83%
CWE127	758	89,022	963s	37.07%
CWE190	3,420	440,749	4,356s	78.13%
CWE191	2,622	340,884	3,236s	78.87%
CWE369	497	83,238	674s	70.42%
CWE415	190	17,990	228s	100.00%
CWE416	118	14,782	142s	67.80%
CWE469	18	1,520	22s	100.00%
CWE476	216	20,427	254s	100.00%

Table 1: Juliet benchmarks (non-relational configuration, no partitioning).

Benchmark	Time	Selectivity	# checks
basename	33.79s	98.65%	11,731
comm	42.67s	97.32%	12,654
dircolors	34.82s	99.74%	20,062
dirname	21.68s	99.61%	11,307
echo	19.26s	99.43%	11,010
false	14.50s	99.72%	10,774
getlimits	34.62s	98.54%	11,711
hostid	18.05s	99.65%	11,303
id	32.69s	99.04%	12,338
link	23.03s	99.52%	11,572
logname	20.36s	99.66%	11,307
mkfifo	34.87s	99.20%	11,807

Table 2: `coreutils` benchmarks (fully symbolic arguments, relational analysis).

Easing debugging

Where static analyzers usually start from

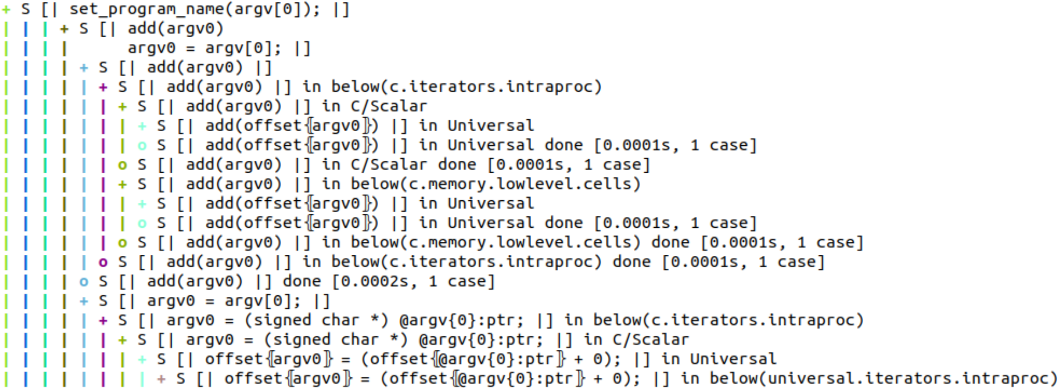
- Analysis output Too coarse

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 <u>back</u> 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

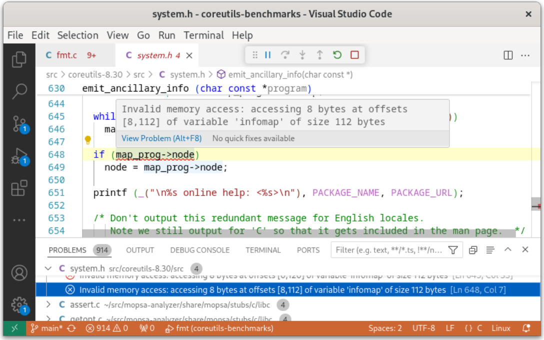
GDB-like interface to the abstract interpretation of the program

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

IDE support

- Language Server Protocol for linters (report alarms)


system.h - coreutils-benchmarks - Visual Studio Code
File
Edit
Selection
View
Go
Run
Terminal
Help
fmt.c 9+
system.h 4
src > coreutils-8.30 > src > system.h > emit_ancillary_info(char const *)
PROBLEMS 914
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/n...
main*
914 0
0
fmt (coreutils-benchmarks)
Spaces: 2
UTF-8
LF
{} C
Linux

```
emit_ancillary_info (char const *program)

  whil
    ma

  if (map_prog->node)
    node = map_prog->node;

  printf (_("\n%s online help: <%s>\n"), PACKAGE_NAME, PACKAGE_URL);

  /* Don't output this redundant message for English locales.
     Note we still output for 'C' so that it gets included in the man page.  */
```

```
Invalid memory access: accessing 8 bytes at offsets
[8,112] of variable 'infomap' of size 112 bytes
```

View Problem (Alt+F8)   No quick fixes available

system.h src/coreutils-8.30/src 4

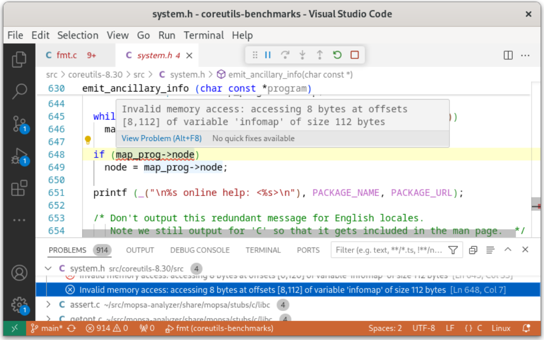
Invalid memory access: accessing 8 bytes at offsets [8,112] of variable 'infomap' of size 112 bytes [Ln 648, Col 7]

assert.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4

getopt.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4

IDE support

- Language Server Protocol for linters (report alarms)
- Debug Adapter Protocol providing interactive engine interface


system.h - coreutils-benchmarks - Visual Studio Code
File Edit Selection View Go Run Terminal Help
fmt.c 9+
system.h 4
src > coreutils-8.30 > src > system.h > emit_ancillary_info(char const *)
PROBLEMS 914
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/n...
main* ⊗ 914 ⚠ 0 0 fmt (coreutils-benchmarks) Spaces: 2 UTF-8 LF {} C Linux

```
emit_ancillary_info (char const *program)

  whil
    ma

  if (map_prog->node)
    node = map_prog->node;

  printf (_("\n%s online help: <%s>\n"), PACKAGE_NAME, PACKAGE_URL);

  /* Don't output this redundant message for English locales.
     Note we still output for 'C' so that it gets included in the man page.  */
```

Invalid memory access: accessing 8 bytes at offsets [8,112] of variable 'infomap' of size 112 bytes

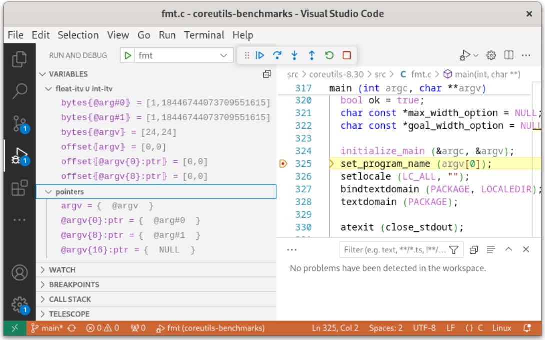
View Problem (Alt+F8) No quick fixes available

system.h src/coreutils-8.30/src 4

⊗ Invalid memory access: accessing 8 bytes at offsets [8,112] of variable 'infomap' of size 112 bytes [Ln 648, Col 7]

assert.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4

getopt.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4


fmt.c - coreutils-benchmarks - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
fmt
src > coreutils-8.30 > src > fmt.c > main(int, char **)
WATCH
BREAKPOINTS
CALL STACK
TELESCOPE
Filter (e.g. text, **/*.ts, !**/...
main* ⊗ 0 ⚠ 0 0 fmt (coreutils-benchmarks) Ln 325, Col 2 Spaces: 2 UTF-8 LF {} C Linux

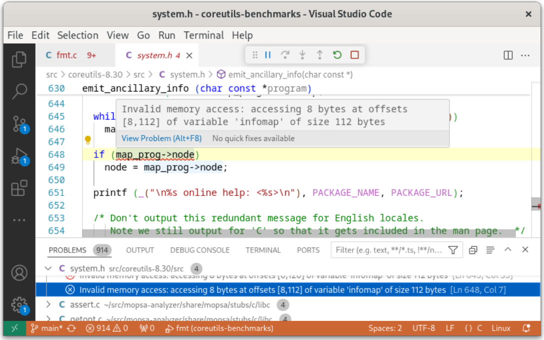
```
VARIABLES
  float-itv ∪ int-itv
    bytes{@arg#0} = [1,18446744073709551615]
    bytes{@arg#1} = [1,18446744073709551615]
    bytes{@argv} = [24,24]
    offset{argv} = [0,0]
    offset{@argv{0}:ptr} = [0,0]
    offset{@argv{8}:ptr} = [0,0]
  pointers
    argv = {  @argv  }
    @argv{0}:ptr = {  @arg#0  }
    @argv{8}:ptr = {  @arg#1  }
    @argv{16}:ptr = {  NULL  }
```

```
main (int argc, char **argv)
  bool ok = true;
  char const *max_width_option = NULL;
  char const *goal_width_option = NULL

  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  setlocale (LC_ALL, "");
  bindtextdomain (PACKAGE, LOCALEDIR);
  textdomain (PACKAGE);

  atexit (close_stdout);
```

No problems have been detected in the workspace.

IDE support

- Language Server Protocol for linters (report alarms)
- Debug Adapter Protocol providing interactive engine interface
- Both protocols introduced by VSCode, supported by multiple IDEs


system.h - coreutils-benchmarks - Visual Studio Code
File Edit Selection View Go Run Terminal Help
fmt.c 9+
system.h 4
src > coreutils-8.30 > src > system.h > emit_ancillary_info(char const *)
PROBLEMS 914
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/n...
main* ⊗ 914 ⚠ 0 0 fmt (coreutils-benchmarks)
Spaces: 2 UTF-8 LF {} C Linux

```
emit_ancillary_info (char const *program)

  whil
    ma

  if (map_prog->node)
    node = map_prog->node;

  printf (_("\n%s online help: <%s>\n"), PACKAGE_NAME, PACKAGE_URL);

  /* Don't output this redundant message for English locales.
     Note we still output for 'C' so that it gets included in the man page.  */
```

```
Invalid memory access: accessing 8 bytes at offsets
[8,112] of variable 'infomap' of size 112 bytes
```

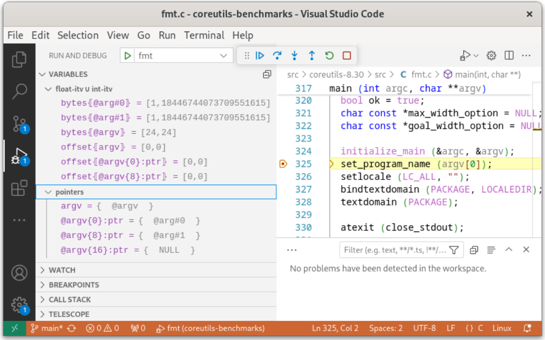
View Problem (Alt+F8)   No quick fixes available

system.h src/coreutils-8.30/src 4

Invalid memory access: accessing 8 bytes at offsets [8,112] of variable 'infomap' of size 112 bytes [Ln 648, Col 7]

assert.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4

getopt.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4


fmt.c - coreutils-benchmarks - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
fmt
VARIABLES
src > coreutils-8.30 > src > fmt.c > main(int, char **)
WATCH
BREAKPOINTS
CALL STACK
TELESCOPE
Filter (e.g. text, **/*.ts, !**/...
main* ⊗ 0 ⚠ 0 0 fmt (coreutils-benchmarks)
Ln 325, Col 2 Spaces: 2 UTF-8 LF {} C Linux

```
float-itv ∪ int-itv
  bytes{@arg#0} = [1,18446744073709551615]
  bytes{@arg#1} = [1,18446744073709551615]
  bytes{@argv} = [24,24]
  offset{argv} = [0,0]
  offset{@argv{0}:ptr} = [0,0]
  offset{@argv{8}:ptr} = [0,0]
pointers
  argv = {  @argv  }
  @argv{0}:ptr = {  @arg#0  }
  @argv{8}:ptr = {  @arg#1  }
  @argv{16}:ptr = {  NULL  }
```

```
main (int argc, char **argv)
  bool ok = true;
  char const *max_width_option = NULL;
  char const *goal_width_option = NULL

  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  setlocale (LC_ALL, "");
  bindtextdomain (PACKAGE, LOCALEDIR);
  textdomain (PACKAGE);

  atexit (close_stdout);
```

No problems have been detected in the workspace.

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

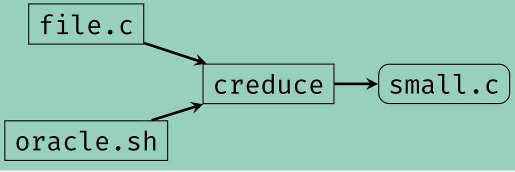
- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases
- Debugging extremely difficult: size of the program, analysis time

Testcase reduction

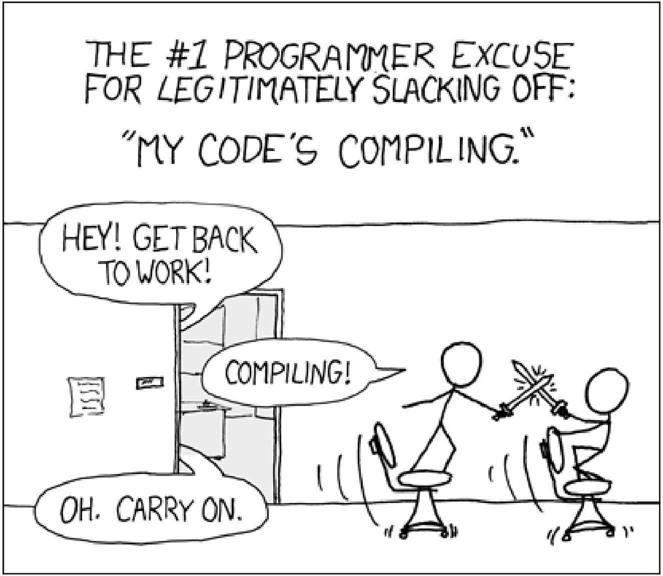
Motivation

- 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` [Reg+12]



Testcase reduction – II



Testcase reduction – III

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 – III

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

Reference	Origin	Original LoC	Reduced LoC	Reduction
Issue 76	SV-Comp	28,737	18	99.94%
Issue 81	SV-Comp	15,627	8	99.95%
Issue 134	SV-Comp	17,411	10	99.94%
Issue 135	SV-Comp	7,016	12	99.83%
M.R. 130	`coreutils`	77,981	20	99.97%
M.R. 145	`coreutils`	77,427	19	99.98%

Testcase reduction – IV

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

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 top expressions are created

Simplifies the search for sources of large imprecision (esp. with rewritings)

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

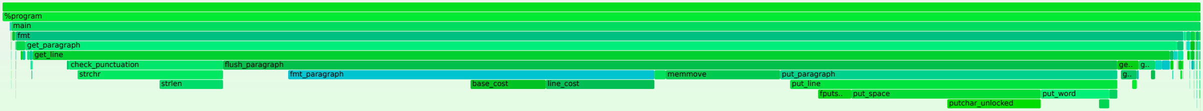
Standard profiling

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



Mopsa analysis of coreutils fmt

Profiling – II

Apron vs PPLite on Coreutils touch

- PPLite is 14% slower but more precise (11 alarms removed). Why?

Profiling – II

Apron vs PPLite on Coreutils touch

- PPLite is 14% slower but more precise (11 alarms removed). Why?
- Suggestion from Enea Zaffanella: widening operator.

Profiling – II

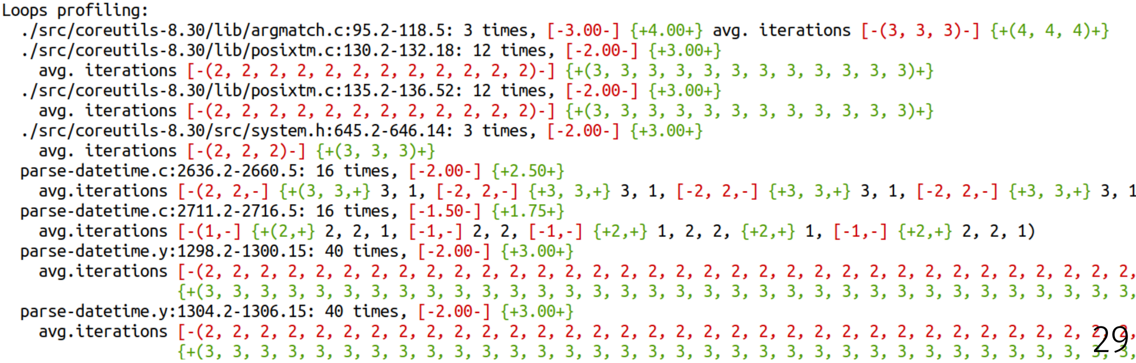
Apron vs PPLite on Coreutils touch

- PPLite is 14% slower but more precise (11 alarms removed). Why?
- Suggestion from Enea Zaffanella: widening operator.
- Easy to confirm intuition!

Profiling – II

Apron vs PPLite on Coreutils touch

- PPLite is 14% slower but more precise (11 alarms removed). Why?
- Suggestion from Enea Zaffanella: widening operator.
- Easy to confirm intuition!



```
Loops profiling:
  ./src/coreutils-8.30/lib/argmatch.c:95.2-118.5: 3 times, [-3.00-] {+4.00+} avg. iterations [-(3, 3, 3)-] {+(4, 4, 4)+}
  ./src/coreutils-8.30/lib/posixtm.c:130.2-132.18: 12 times, [-2.00-] {+3.00+}
    avg. iterations [-(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)-] {+(3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3)+}
  ./src/coreutils-8.30/lib/posixtm.c:135.2-136.52: 12 times, [-2.00-] {+3.00+}
    avg. iterations [-(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)-] {+(3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3)+}
  ./src/coreutils-8.30/src/system.h:645.2-646.14: 3 times, [-2.00-] {+3.00+}
    avg. iterations [-(2, 2, 2)-] {+(3, 3, 3)+}
  parse-datetime.c:2636.2-2660.5: 16 times, [-2.00-] {+2.50+}
    avg.iterations [-(2, 2,-] {+(3, 3,+} 3, 1, [-2, 2,-] {+3, 3,+} 3, 1, [-2, 2,-] {+3, 3,+} 3, 1, [-2, 2,-] {+3, 3,+} 3, 1
  parse-datetime.c:2711.2-2716.5: 16 times, [-1.50-] {+1.75+}
    avg.iterations [-(1,-] {+(2,+} 2, 2, 1, [-1,-] 2, 2, [-1,-] {+2,+} 1, 2, 2, {+2,+} 1, [-1,-] {+2,+} 2, 2, 1)
  parse-datetime.y:1298.2-1300.15: 40 times, [-2.00-] {+3.00+}
    avg.iterations [-(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                   {+(3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  parse-datetime.y:1304.2-1306.15: 40 times, [-2.00-] {+3.00+}
    avg.iterations [-(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                   {+(3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

Conclusion

Related work

Lots of folklore

Related work

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*”
  ⇝ “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*"
  ⇝ "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*”
  ⇝ “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

References – I

[AMN17] Esben Sparre Andreasen, Anders Møller, and Benjamin Barslev Nielsen. “Systematic approaches for increasing soundness and precision of static analyzers”. In: ed. by Karim Ali and Cristina Cifuentes. ACM, 2017, pp. 31–36. DOI: `10.1145/3088515.3088521`.

[Bau+22] Guillaume Bau et al. “Abstract interpretation of Michelson smart-contracts”. In: ed. by Laure Gonnord and Laura Titolo. ACM, 2022, pp. 36–43. DOI: `10.1145/3520313.3534660`.

References – II

[DM19] David Delmas and Antoine Miné. “Analysis of Software Patches Using Numerical Abstract Interpretation”. In: ed. by Bor-Yuh Evan Chang. Lecture Notes in Computer Science. Springer, 2019, pp. 225–246. DOI: `10.1007/978-3-030-32304-2_12`.

[DOM21] David Delmas, Abdelraouf Ouadjaout, and Antoine Miné. “Static Analysis of Endian Portability by Abstract Interpretation”. In: Lecture Notes in Computer Science. Springer, 2021, pp. 102–123.

[JMO18] Matthieu Journault, Antoine Miné, and Abdelraouf Ouadjaout. “Modular Static Analysis of String Manipulations in C Programs”. In: ed. by Andreas Podelski. Lecture Notes in Computer Science. Springer, 2018, pp. 243–262. DOI: `10.1007/978-3-319-99725-4_16`.

References – III

[Jou+19] M. Journault et al. “Combinations of reusable abstract domains for a multilingual static analyzer”. In: New York, USA, July 2019, pp. 1–17.

[KCW19] Christian Klinger, Maria Christakis, and Valentin Wüstholz. “Differentially testing soundness and precision of program analyzers”. In: ed. by Dongmei Zhang and Anders Møller. ACM, 2019, pp. 239–250. DOI: 10.1145/3293882.3330553.

[LDB19] Linghui Luo, Julian Dolby, and Eric Bodden. “MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper)”. In: ed. by Alastair F. Donaldson. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 21:1–21:25. DOI: 10.4230/LIPICS.ECOOP.2019.21.

References – IV

[Liv+15] Benjamin Livshits et al. “In defense of soundiness: a manifesto”. In: Commun. ACM 2 (2015), pp. 44–46. DOI: 10.1145/2644805.

[MFM24] Raphaël Monat, Aymeric Fromherz, and Denis Merigoux. “Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law”. In: ed. by Stephanie Weirich. Lecture Notes in Computer Science. Springer, 2024, pp. 421–450. DOI: 10.1007/978-3-031-57267-8_16.

References – V

[MM24] Marco Milanese and Antoine Miné. “Generation of Violation Witnesses by Under-Approximating Abstract Interpretation”. In: ed. by Rayna Dimitrova, Ori Lahav, and Sebastian Wolff. Lecture Notes in Computer Science. Springer, 2024, pp. 50–73. DOI: 10.1007/978-3-031-50524-9_3.

[MOM20a] R. Monat, A. Ouadjaout, and A. Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. In: LIPIcs. 2020.

[MOM20b] R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. In: ACM, 2020, pp. 8–13. DOI: 10.1145/3394451.3397205.

References – VI

[MOM21] R. Monat, A. Ouadjaout, and A. Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. In: 2021.

[Mon+24] Raphaël Monat et al. “Mopsa-C: Improved Verification for C Programs, Simple Validation of Correctness Witnesses (Competition Contribution)”. In: ed. by Bernd Finkbeiner and Laura Kovács. Lecture Notes in Computer Science. Springer, 2024, pp. 387–392. DOI: `10.1007/978-3-031-57256-2_26`.

[MVR23] Mats Van Molle, Bram Vandenbogaerde, and Coen De Roover. “Cross-Level Debugging for Static Analysers”. In: ed. by João Saraiva, Thomas Degueule, and Elizabeth Scott. ACM, 2023, pp. 138–148. DOI: `10.1145/3623476.3623512`.

References – VII

[OM20] A. Ouadjaout and A. Miné. "A Library Modeling Language for the Static Analysis of C Programs". In: ed. by David Pichardie and Mihaela Sighireanu. Lecture Notes in Computer Science. Springer, 2020, pp. 223–247. DOI: 10.1007/978-3-030-65474-0_11.

[PM24] Francesco Parolini and Antoine Miné. "Sound Abstract Nonexploitability Analysis". In: ed. by Rayna Dimitrova, Ori Lahav, and Sebastian Wolff. Lecture Notes in Computer Science. Springer, 2024, pp. 314–337. DOI: 10.1007/978-3-031-50521-8_15.

[Reg+12] John Regehr et al. "Test-case reduction for C compiler bugs". In: ed. by Jan Vitek, Haibo Lin, and Frank Tip. ACM, 2012, pp. 335–346. DOI: 10.1145/2254064.2254104.

References – VIII

[TLR20] Jubi Taneja, Zhengyang Liu, and John Regehr. "Testing static analyses for precision and soundness". In: ACM, 2020, pp. 81–93. DOI: 10.1145/3368826.3377927.

[VMM23] Milla Valnet, Raphaël Monat, and Antoine Miné. "Analyse statique de valeurs par interprétation abstraite de programmes fonctionnels manipulant des types algébriques récursifs". In: ed. by Timothy Bourke and Delphine Demange. Praz-sur-Arly, France, Jan. 2023, pp. 211–242.