Analyzing Rsync with Mopsa

Raphaël Monat, joint exploration with Antoine Miné

rmonat.fr

AnalyzeThat @ ETAPS
12 April 2026

Inria

Outline

An Overview of Mopsa for C analyses

Framework overview

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs

Including multi-language support

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs — Including multi-language support
- Ease development of relational static analyses
  - High expressivity $0 \leq i < \text{strlen}(s)$

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs — Including multi-language support
- Ease development of relational static analyses
  High expressivity $0 \leq i < \text{strlen}(s)$
- Open-source (LGPL)

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs — Including multi-language support
- Ease development of relational static analyses
  High expressivity $0 \leq i < \text{strlen}(s)$
- Open-source (LGPL)
- Can be used as an experimentation platform

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

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs — Including multi-language support
- Ease development of relational static analyses
  High expressivity $0 \leq i < \text{strlen}(s)$
- Open-source (LGPL)
- Can be used as an experimentation platform

Currently, fully context-sensitive analyses

Contributors (2018–2026, 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
- T. Goalard

Contributors (2018–2026, 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
- T. Goalard

Maintainers in bold.

Analysis = composition of abstract domains

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

Mopsa design

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms

Mopsa design

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages

Analysis = composition of abstract domains

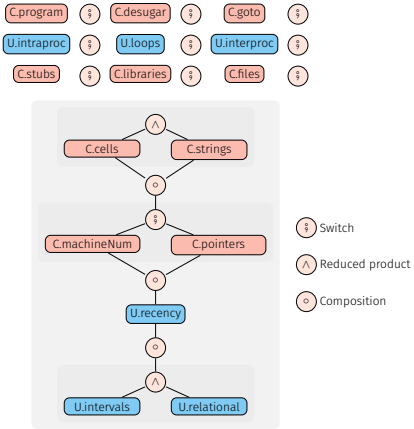
unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages
- defined as json files in `share/mopsa/configs`

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages
- defined as json files in `share/mopsa/configs`



Mopsa's transparent reporting

Mopsa's approach to being transparent

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

Mopsa's transparent reporting

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 transparent reporting

Mopsa's approach to being transparent

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

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

```
1  int main() {
2    int n = _mopsa_rand_s32();
3    int y = -1;
4    for(int x = 0; x < n; x++)
5      y++;
6  }
```

Mopsa's transparent reporting

Mopsa's approach to being transparent

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

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

```
1  int main() {
2    int n = _mopsa_rand_s32();
3    int y = -1;
4    for(int x = 0; x < n; x++)
5      y++;
6  }
```

Stmt
x++
y++
Selectivity

Mopsa's transparent reporting

Mopsa's approach to being transparent

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

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

```
1  int main() {
2    int n = _mopsa_rand_s32();
3    int y = -1;
4    for(int x = 0; x < n; x++)
5      y++;
6  }
```

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

Mopsa's transparent reporting

Mopsa's approach to being transparent

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

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

```
1  int main() {
2    int n = _mopsa_rand_s32();
3    int y = -1;
4    for(int x = 0; x < n; x++)
5      y++;
6  }
```

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

An Overview of Mopsa for C analyses

C analysis

C analysis overview

- Checks for run-time errors
  (integer overflows, invalid dereferences, . . .)

C analysis overview

- Checks for run-time errors
  (integer overflows, invalid dereferences, . . .)
- Supports ints, floats, pointers, structs, . . .

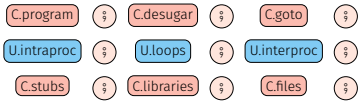
C analysis overview

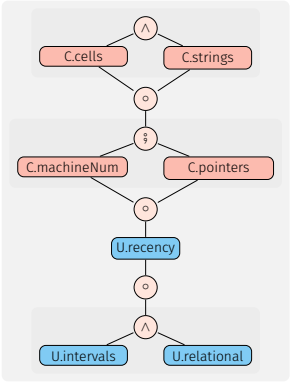
- Checks for run-time errors
  (integer overflows, invalid dereferences, . . .)
- Supports ints, floats, pointers, structs, . . .
- Inlining-based analysis
  ⚠ scalability

C analysis overview

- Checks for run-time errors
  (integer overflows, invalid dereferences, . . .)
- Supports ints, floats, pointers, structs, . . .
- Inlining-based analysis
  ⚠ scalability
- No concurrency support

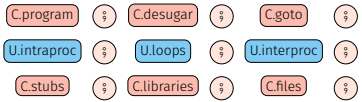
C analysis domains

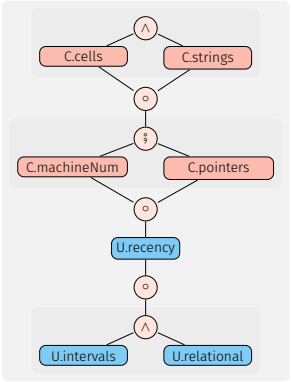


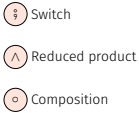


- Cells [Min06] low-level memory

C analysis domains

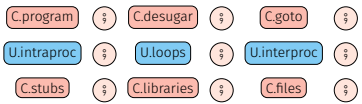


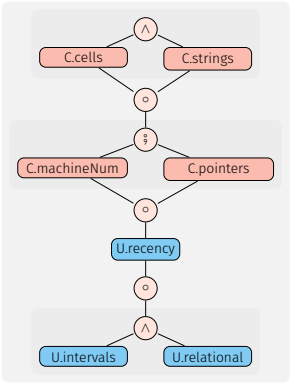




- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations

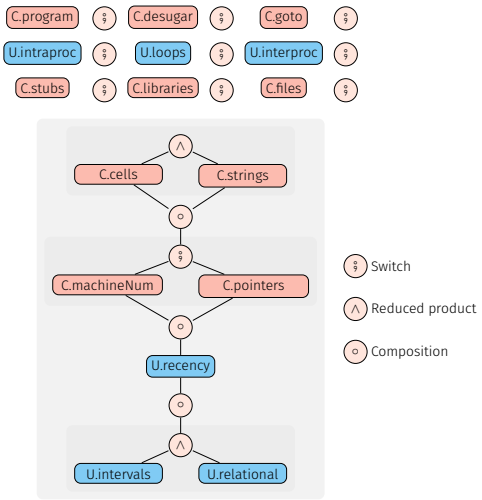
C analysis domains





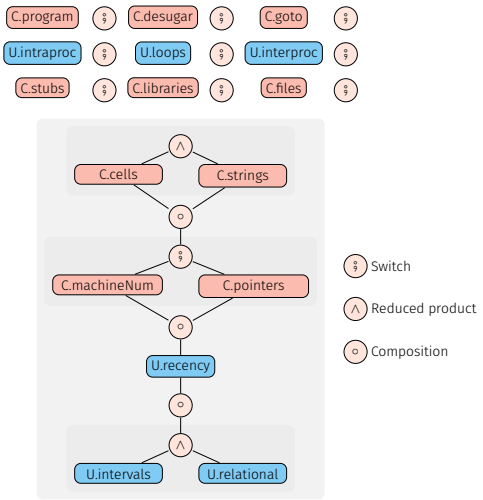
- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]

C analysis domains



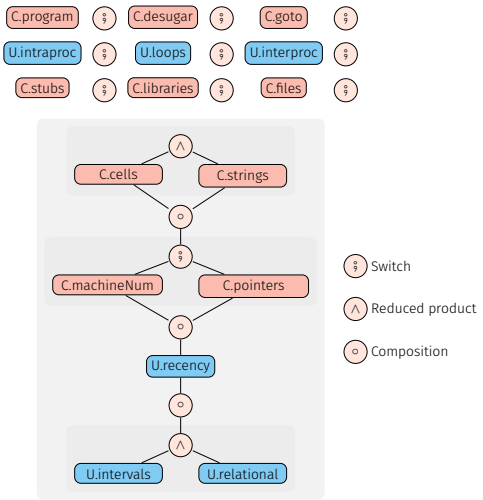
- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block

C analysis domains



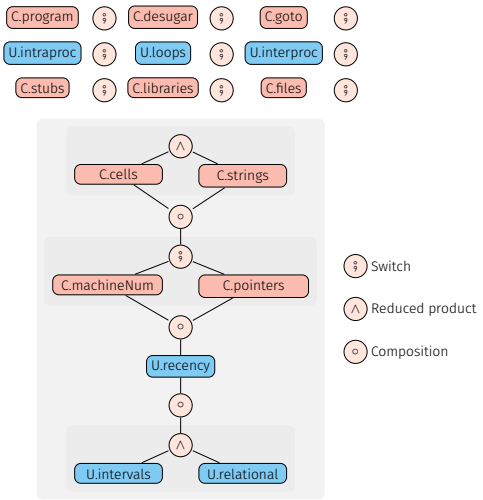
- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block
  - slen(`var`) position of first must 0.

C analysis domains



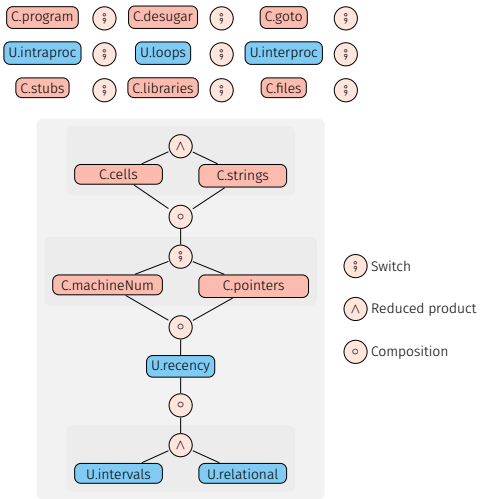
- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block
  - slen(`var`) position of first must 0.
- Symbolic argument modeling

C analysis domains



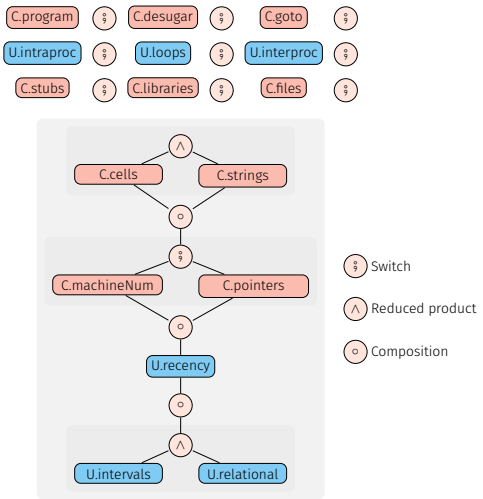
- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block
  - slen(`var`) position of first must 0.
- Symbolic argument modeling
  - $1 \leq i < \texttt{argc} \implies \text{valid\_string}(\texttt{argv}[i])$

C analysis domains



- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block
  - slen(`var`) position of first must 0.
- Symbolic argument modeling
  - $1 \leq i < \texttt{argc} \implies \text{valid\_string}(\texttt{argv}[i])$
  - $\texttt{argv}[\texttt{argc}] = \texttt{NULL}$

C analysis domains



- Cells [Min06] low-level memory
- Recency [BR06] for dynamic allocations
- String-length domain [JMO18]
  - bytes(`var`) size in bytes of memory block
  - slen(`var`) position of first must 0.
- Symbolic argument modeling
  - $1 \leq i < \texttt{argc} \implies \text{valid\_string}(\texttt{argv}[i])$
  - $\texttt{argv}[\texttt{argc}] = \texttt{NULL}$
- Contract language for libc [OM20], inspired from Frama-C ACSL

Some benchmarks

Mopsa wins the *SoftwareSystems* track of SV-Comp 2024 and 2026!

Some benchmarks

Mopsa wins the *SoftwareSystems* track of SV-Comp 2024 and 2026!

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

Mopsa wins the *SoftwareSystems* track of SV-Comp 2024 and 2026!

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).

An Overview of Mopsa for C analyses

Easing implementation and maintenance

Easing implementation and maintenance

"std. tools on the concrete execution of the *abstract interpreter*"

Easing implementation and maintenance

"std. tools on the concrete execution of the *abstract interpreter*"

$\rightsquigarrow$ "new tools on abstract execution of *target program*"

Read more in our STTT paper [MOM24]

Static analyses interfaces: traditional approaches

- Analysis output

Too coarse

Static analyses interfaces: traditional approaches

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

Static analyses interfaces: traditional approaches

- 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)
```

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

- Breakpoints

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

- Breakpoints
  - Program location

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online demo)

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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online 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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online 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

Static analyses interface: an abstract debugger

GDB-like interface to the abstract interpretation of the program (online 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

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

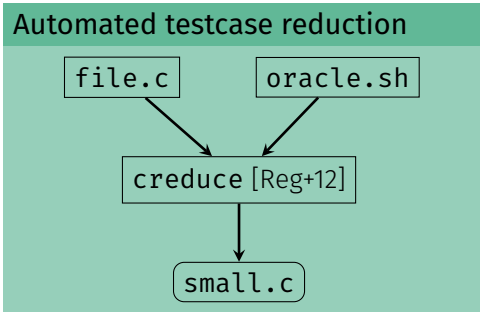
Abstract profiling hook

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

- Loop-level profiling
- Function-level profiling

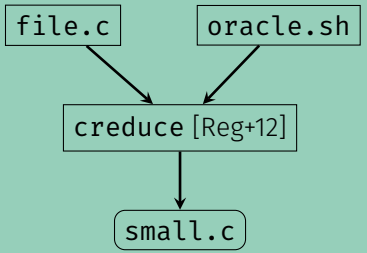
Partial results reporting supported!

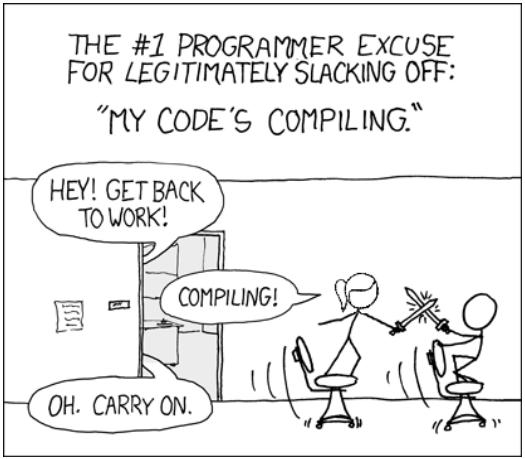
Testcase reduction



Testcase reduction

Automated testcase reduction





Testcase reduction – II

Mopsa supports multi-file C projects

- `mopsa-build`

Testcase reduction – II

Mopsa supports multi-file C projects

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

Testcase reduction – II

Mopsa supports multi-file C projects

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

Testcase reduction – II

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`

Testcase reduction – II

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
```

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

Testcase reduction – II

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
  - Source-level linking

Testcase reduction – II

Mopsa supports multi-file C projects

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

  $\rightsquigarrow$ `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
  - Source-level linking
- Usability: upon encountering an RTE, Mopsa helps triggering the reduction

Analyzing Rsync

Envisioned workflow

Envisioned workflow

1. Parsing fixes

Envisioned workflow

1. Parsing fixes
2. Maximize coverage

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision
   - Choice of abstract domains

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision
   - Choice of abstract domains
   - Choice of other analysis options (unrolling, allocation sensitivity, ...)

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision
   - Choice of abstract domains
   - Choice of other analysis options (unrolling, allocation sensitivity, ...)
5. Manually investigate some alarms

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision
   - Choice of abstract domains
   - Choice of other analysis options (unrolling, allocation sensitivity, ...)
5. Manually investigate some alarms
   - Manual work...

Envisioned workflow

1. Parsing fixes
2. Maximize coverage
   - Symbolic argument modeling by default
   - Coverage hook for high-level inspection
3. Check for soundness assumptions and issues
   - Transparent report of Mopsa
   - Heuristic hook detecting dubious cases
4. Find sweet spot between performance and precision
   - Choice of abstract domains
   - Choice of other analysis options (unrolling, allocation sensitivity, ...)
5. Manually investigate some alarms
   - Manual work...
   - Can rely on the interactive engine to understand the alarm

Analyzing Rsync

What happened in practice

gitlab.com/mopsa/benchmarks/rsync-analysis/

Source code parsing

Mostly fine with the AnalyzeThat fixes.

gitlab.com/mopsa/benchmarks/rsync-analysis/

Source code parsing

Mostly fine with the AnalyzeThat fixes.

Then: a manual quest

gitlab.com/mopsa/benchmarks/rsync-analysis/

Source code parsing

Mostly fine with the AnalyzeThat fixes.

Then: a manual quest

for scalability

gitlab.com/mopsa/benchmarks/rsync-analysis/

Source code parsing

Mostly fine with the AnalyzeThat fixes.

Then: a manual quest

for scalability

without sacrificing too much coverage

gitlab.com/mopsa/benchmarks/rsync-analysis/

Source code parsing

Mostly fine with the AnalyzeThat fixes.

Then: a manual quest

for scalability

without sacrificing too much coverage

Approach

Observe analysis progress and investigate using

- Interactive engine
- Profilers

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues
- we crafted efficient stubs equivalent in the abstract

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues
- we crafted efficient stubs equivalent in the abstract

State-space reduction

Large state space due to options and behaviors

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues
- we crafted efficient stubs equivalent in the abstract

State-space reduction

Large state space due to options and behaviors

- Manual state partitioning on boolean values `am_server`, `am_sender`.

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues
- we crafted efficient stubs equivalent in the abstract

State-space reduction

Large state space due to options and behaviors

- Manual state partitioning on boolean values `am_server`, `am_sender`.
  - Could be done in a single analysis but reduces parallelization factor

Tradeoffs and simplifications

Argument parsing

- `popt` entailed analysis performance issues
- we crafted efficient stubs equivalent in the abstract

State-space reduction

Large state space due to options and behaviors

- Manual state partitioning on boolean values `am_server`, `am_sender`.
  - Could be done in a single analysis but reduces parallelization factor
- Set `protocol_version`=30. Same for remote version

Tradeoffs and simplifications (II)

Sidestepping other issues

Tradeoffs and simplifications (II)

Sidestepping other issues

- Recursive functions

Tradeoffs and simplifications (II)

Sidestepping other issues

- Recursive functions
- Logging and debug info; cleanup

Tradeoffs and simplifications (II)

Sidestepping other issues

- Recursive functions
- Logging and debug info; cleanup
- Signals

Tradeoffs and simplifications (II)

Sidestepping other issues

- Recursive functions
- Logging and debug info; cleanup
- Signals
- Fork

Tradeoffs and simplifications (II)

Sidestepping other issues

- Recursive functions
- Logging and debug info; cleanup
- Signals
- Fork
- Directories...

Some bugs and improvements

Some bugs and improvements

On the Mopsa side...

Some bugs and improvements

On the Mopsa side...

`select` stub

```
1 * requires: __nfds >= 1;
2 * requires: null_or_valid_ptr(__exceptfds);
3 * requires: null_or_valid_ptr(__timeout);
4 * assigns: _errno;
5 * ensures: return >= -1 and return <- __nfds;
6 * unsound: "select does not check that file-descriptors are valid";
```

Found using heuristic unsoundness detection hook

Some bugs and improvements

On the Mopsa side...

`select` stub

```
1 * requires: __nfds >= 1;
2 * requires: null_or_valid_ptr(__exceptfds);
3 * requires: null_or_valid_ptr(__timeout);
4 * assigns: _errno;
5 * ensures: return >= -1 and return <- __nfds;
6 * unsound: "select does not check that file-descriptors are valid";
```

Found using heuristic unsoundness detection hook

Some bugs and improvements

On the Mopsa side...

`select` stub

```
1 * requires: __nfds >= 1;
2 * requires: null_or_valid_ptr(__exceptfds);
3 * requires: null_or_valid_ptr(__timeout);
4 * assigns: _errno;
5 * ensures: return >= -1 and return <- __nfds;
6 * unsound: "select does not check that file-descriptors are valid";
```

Found using heuristic unsoundness detection hook

Scalability of the interactive engine

Some bugs and improvements (II)

Callstack $\rightsquigarrow$ Control Context

Track also loop iteration numbers to roughly estimate progresss

Trace partitioning issues

`creduce/cvise` came in handy!

Profiling-directed simplifications

Function	Total time	Self time	Count
%program	19582.7525s	0.8358s	1
main	19581.7933s	0.6412s	1
start_client	19537.7848s	0.7921s	1
client_run	19386.0082s	0.9445s	2
send_files	14713.8498s	4.1589s	2
read_buf	14118.5177s	40.2160s	115
read_a_msg	13454.5005s	76.4954s	279
perform_io	11979.7325s	4204.6916s	8836
start_socket_client	10030.5979s	0.4299s	1
read_ndx_and_attrs	8883.6031s	3.1239s	6
raw_read_int	7557.4465s	37.7534s	2511
raw_read_buf	7007.3097s	76.1009s	4367
receive_sums	5230.0320s	2.2389s	4
read_ndx	5195.5426s	2.0533s	8
read_int	5139.1159s	1.1859s	35
write_buf	3402.1354s	67.0873s	1436
read_sum_head	2683.9100s	0.7428s	4
match_sums	2639.4841s	2.2675s	4
read_final_goodbye	2497.0366s	0.2859s	2
_exit_cleanup	2476.6955s	2476.6955s	152816

Investigating spurious loop iterations

```
$ stat -c "%y %n" -- sender.226.{1,2}
2026-04-07 12:02:50.152871307 +0200 sender.226.1
2026-04-07 13:07:52.679079724 +0200 sender.226.2
```

```
<       phase:./sender.c:210.1-14 -> [0,2147483647],
---
>       phase:./sender.c:210.1-14 -> [-2147483648,2147483647],
```

Investigating spurious loop iterations

```
$ stat -c "%y %n" -- sender.226.{1,2}
2026-04-07 12:02:50.152871307 +0200 sender.226.1
2026-04-07 13:07:52.679079724 +0200 sender.226.2
```

```
<     phase:./sender.c:210.1-14 -> [0,2147483647],
---
>     phase:./sender.c:210.1-14 -> [-2147483648,2147483647],
```

```c
void send_files(int f_in, int f_out) /* Sliced version */
{
  int phase = 0, max_phase = protocol_version >= 29 ? 2 : 1;
  while (1) {
    /* ... */
    if (ndx == NDX_DONE) {
      if (++phase > max_phase)
        break;
    }
  }
```

Analyzing Rsync

Current results

Current results

Hardware details

- Core i7-12700 desktop

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs

⚠ rough experimental data

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

- Intervals

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

- Intervals
- Tracking string length

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

- Intervals
- Tracking string length
- Reduced allocation sensitivity

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

- Intervals
- Tracking string length
- Reduced allocation sensitivity
- Widening with thresholds

Current results

Hardware details

- Core i7-12700 desktop
- Single-threaded analysis
- RAM usage: few GBs
- ⚠ rough experimental data

Analysis

- Intervals
- Tracking string length
- Reduced allocation sensitivity
- Widening with thresholds

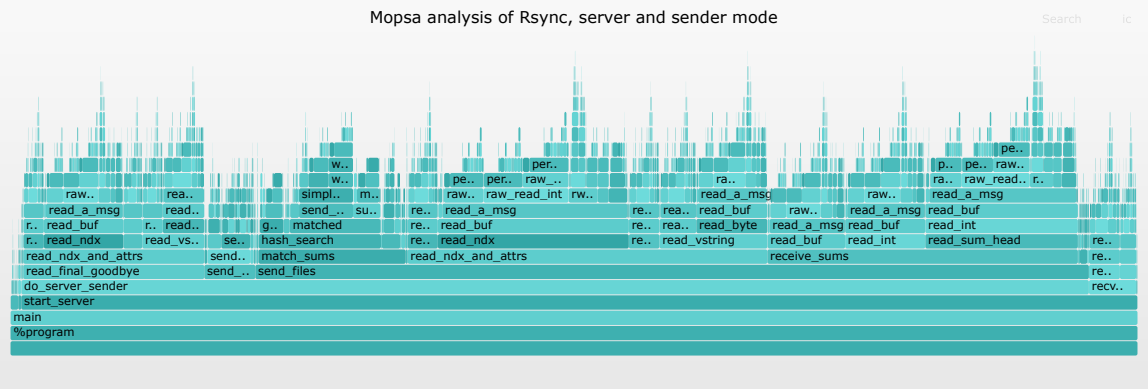
sender? \ server?	1	0
1	2h25m	5h26m
0	> 36h	24h11m

Server & sender case (2h25m)

```
Checks summary: 916479 total, ✔ 881015 safe, ✗ 639 errors, ⚠ 34825 warnings (selectivity: 96.14%)
  Stub condition: 21789 total, ✔ 15447 safe, ✗ 83 errors, ⚠ 6259 warnings
  Invalid memory access: 180065 total, ✔ 166598 safe, ✗ 489 errors, ⚠ 12978 warnings
  Division by zero: 15965 total, ✔ 15452 safe, ⚠ 513 warnings
  Integer overflow: 422425 total, ✔ 407905 safe, ✗ 2 errors, ⚠ 14518 warnings
  Invalid shift: 33077 total, ✔ 33059 safe, ⚠ 18 warnings
  Invalid pointer comparison: 693 total, ✔ 321 safe, ⚠ 372 warnings
  Invalid pointer subtraction: 838 total, ✔ 699 safe, ⚠ 139 warnings
  Double free: 65 total, ✗ 65 errors
  Insufficient format arguments: 4688 total, ✔ 4674 safe, ⚠ 14 warnings
  Invalid type of format argument: 4639 total, ✔ 4625 safe, ⚠ 14 warnings
  Incorrect number of arguments: 232235 total, ✔ 232235 safe

384 assumptions:
```

Server & sender case (2h25m)



Server & sender case

Impact of analysis options?

⚠ (very) early results

Abstract domains	Allocation Policy	Analysis Time	Selectivity
Cell Itv	Range	2h10m	96.0108%
Cell StrLen Itv	Range	2h25m	96.1304%
Cell StrLen Itv	Range Callstack	5h27m	96.4447%
Cell StrLen Itv Oct	Range	5h23m	96.2272%
Cell StrLen Itv Oct	Range Callstack	16h20m	96.5900%
Cell StrLen Itv Poly	Range	†19h50m	96.2294%
Cell StrLen Itv Poly	Range Callstack	†30h16m	96.5897%

Client & sender case (5h26m)

```
Checks summary: 2186352 total, ✔ 2097262 safe, ✗ 1122 errors, ⚠ 87968 warnings (selectivity: 95.93%)
  Stub condition: 54835 total, ✔ 36596 safe, ✗ 700 errors, ⚠ 17539 warnings
  Invalid memory access: 417177 total, ✔ 382810 safe, ✗ 328 errors, ⚠ 34039 warnings
  Division by zero: 38720 total, ✔ 37578 safe, ⚠ 1142 warnings
  Integer overflow: 993399 total, ✔ 960289 safe, ✗ 4 errors, ⚠ 33106 warnings
  Invalid shift: 78408 total, ✔ 77404 safe, ⚠ 1004 warnings
  Invalid pointer comparison: 1343 total, ✔ 696 safe, ⚠ 647 warnings
  Invalid pointer subtraction: 1809 total, ✔ 1488 safe, ⚠ 321 warnings
  Double free: 90 total, ✗ 90 errors
  Insufficient variadic arguments: 2 total, ⚠ 2 warnings
  Insufficient format arguments: 15283 total, ✔ 15199 safe, ⚠ 84 warnings
  Invalid type of format argument: 15255 total, ✔ 15171 safe, ⚠ 84 warnings
  Incorrect number of arguments: 570031 total, ✔ 570031 safe

600 assumptions:
```

Conclusion

Conclusion

Mixed bag...

- Not enough time spent?

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...
- Scalability issues

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...
- Scalability issues
- $\simeq$ 8 improvements to be added in Mopsa

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...
- Scalability issues
- $\simeq$ 8 improvements to be added in Mopsa

Future considerations

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...
- Scalability issues
- $\simeq$ 8 improvements to be added in Mopsa

Future considerations

- Modular function analyses, require memory model overhaul

Conclusion

Mixed bag...

- Not enough time spent?
  - Investigative step of the ideal workflow not reached...
- Scalability issues
- $\simeq$ 8 improvements to be added in Mopsa

Future considerations

- Modular function analyses, require memory model overhaul
- How to identify and sidestep scalability issues? Identify progress blockers?

References – I

[BR06] G. Balakrishnan and T. W. Reps. **“Recency-Abstraction for Heap-Allocated Storage”.** In: LNCS. Springer, 2006, pp. 221–239.

[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.

[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.

References – II

[Min06] Antoine Miné. **“Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics”.** In: ed. by Mary Jane Irwin and Koen De Bosschere. ACM, 2006, pp. 54–63. DOI: 10.1145/1134650.1134659.

[MOM24] Raphaël Monat, Abdelraouf Ouadjaout, and Antoine Miné. **“Easing maintenance of academic static analyzers”.** In: Int. J. Softw. Tools Technol. Transf. 6 (2024), pp. 673–686. DOI: 10.1007/S10009-024-00770-1.

[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.

References – III

[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.