The Mopsa static analysis platform, and our quest to ease implementation & maintenance

Raphaël Monat – SyCoMoRES team, Lille

`rmonat.fr`

Prosecco
10 March 2025

Introduction

Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

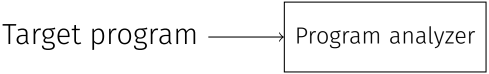
Target program

Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

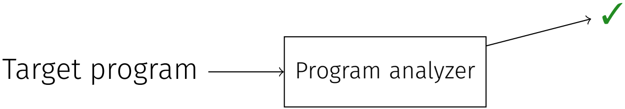


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up



Target program

Program analyzer

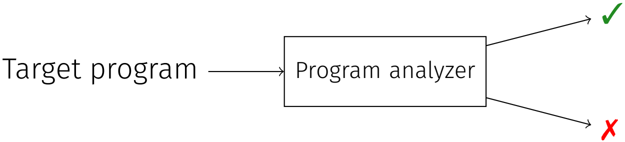
✓

Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

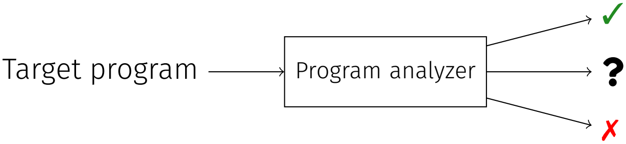


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up



A Program Analysis Recipe

Target property φ

A Program Analysis Recipe

Target property φ

- Absence of runtime errors

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

- Source code

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

- Source code
- Binary executable

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

- Source code
- Binary executable

Requirement: semantics of the program representation

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

- Source code
- Binary executable

Requirement: semantics of the program representation

$\Longrightarrow$ build $\texttt{Analyzer}_{\varphi}(\text{prog} : i)$

A Program Analysis Recipe

Target property φ

- Absence of runtime errors
- Constant-time execution
- Endianness portability [DOM21]

Benchmarks

- Open-source
- Real-world

Input format i

- Source code
- Binary executable

Requirement: semantics of the program representation

$\Longrightarrow$ build $\texttt{Analyzer}_{\varphi}(\text{prog} : i)$ evaluate it on benchmarks

Turing & Rice to the Rescue

Sound

All errors in program reported by analyzer

Turing & Rice to the Rescue

All errors reported by analyzer are replicable in program

Complete

Sound

All errors in program reported by analyzer

Turing & Rice to the Rescue

Guaranteed Termination

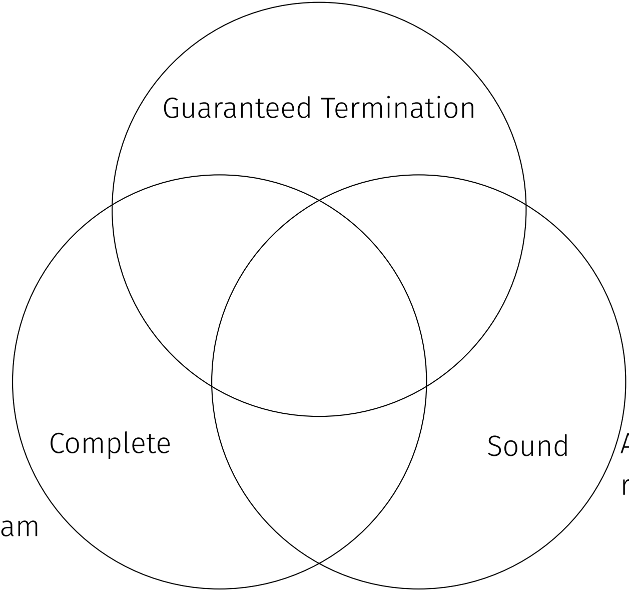
All errors reported by analyzer are replicable in program

Complete

Sound

All errors in program reported by analyzer

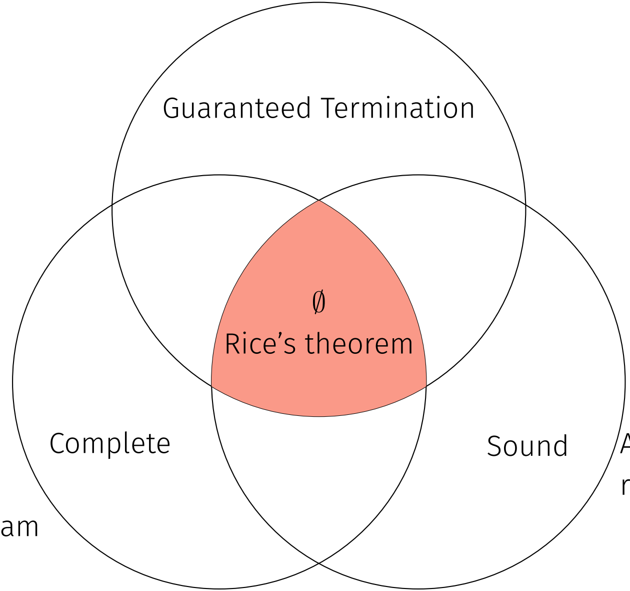
Turing & Rice to the Rescue



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

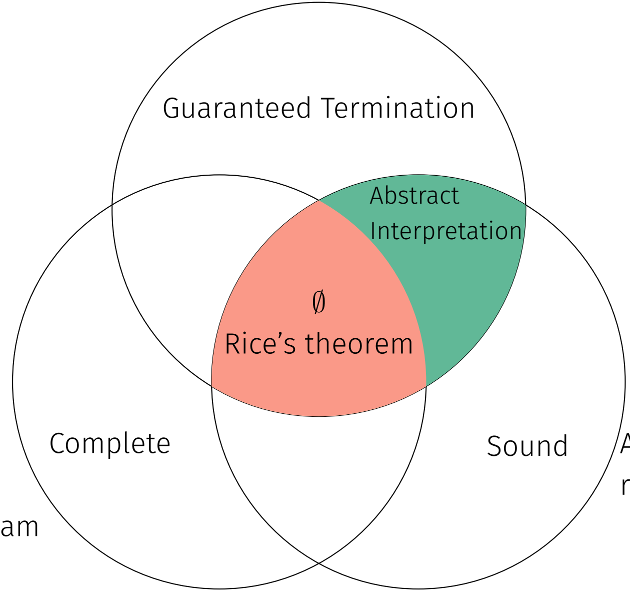
Turing & Rice to the Rescue (or not?)



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Turing & Rice to the Rescue (or not?)



Guaranteed Termination

Abstract Interpretation

∅

Rice’s theorem

Complete

Sound

All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Community Goals

Academic research around static analysis

Community Goals

Academic research around static analysis

Ideal analyzer

Community Goals

Academic research around static analysis

Ideal analyzer

- Sound, precise and scalable

Community Goals

Academic research around static analysis

Ideal analyzer

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

Community Goals

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

Community Goals

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

- Debugging time-consuming

Community Goals

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

Community Goals

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

Outline

1. An AI primer

2. An overview of Mopsa

3. Easing maintenance and implementation

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
  - Alarms: deciding which ones are true

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
  - Alarms: deciding which ones are true
  - Usually cannot prove programs incorrect

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
  - Alarms: deciding which ones are true
  - Usually cannot prove programs incorrect
- Traditionally used for certification

Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
    - Alarms: deciding which ones are true
    - Usually cannot prove programs incorrect
- Traditionally used for certification
    - Airbus A380/A340 control commands with Astrée [Ber+10]

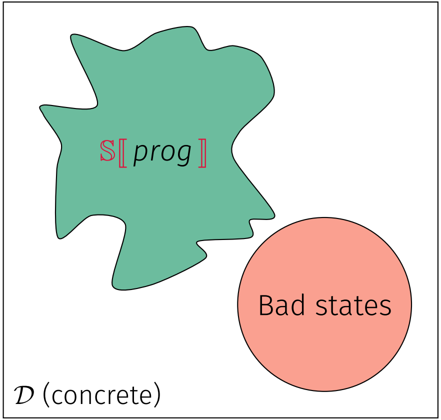
Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
  - Alarms: deciding which ones are true
  - Usually cannot prove programs incorrect
- Traditionally used for certification
  - Airbus A380/A340 control commands with Astrée [Ber+10]
  - Nuclear power plants with Frama-C [BBY17]

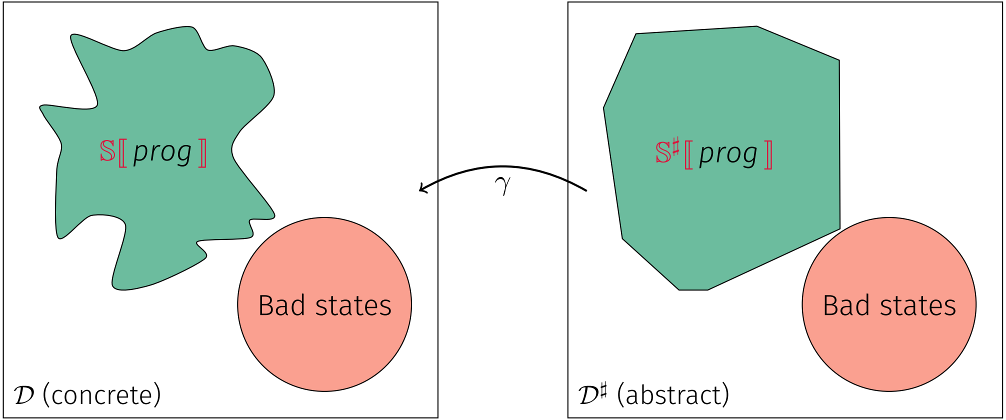
Abstract Interpretation

- Approximate analysis, ensuring soundness and termination
- Invented by Radhia and Patrick Cousot in the late 70s.
- Analysis tries to prove a program correct.
  - Alarms: deciding which ones are true
  - Usually cannot prove programs incorrect
- Traditionally used for certification
  - Airbus A380/A340 control commands with Astrée [Ber+10]
  - Nuclear power plants with Frama-C [BBY17]
- Suggested entry-point: Miné [Min17]

Approximating a Complex World, with Guarantees



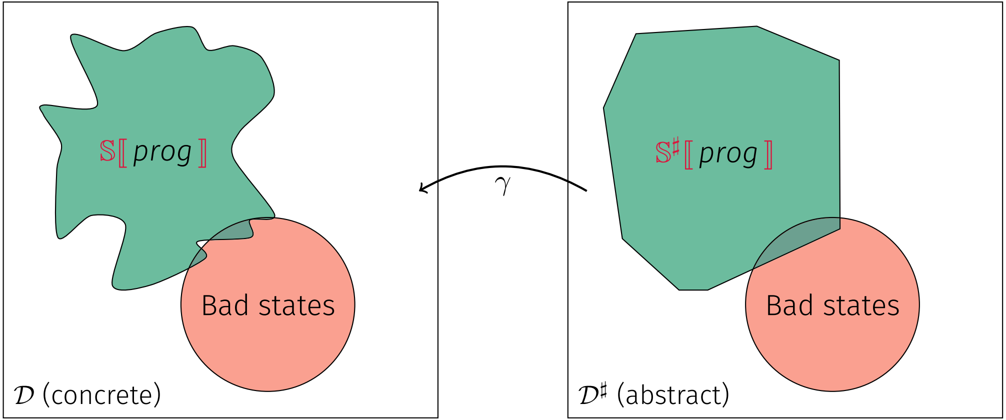
Approximating a Complex World, with Guarantees


$\mathbb{S}[\![\textit{prog}]\!]$
Bad states
$\mathcal{D}$ (concrete)
γ
$\mathbb{S}^\sharp[\![\textit{prog}]\!]$
Bad states
$\mathcal{D}^\sharp$ (abstract)

Interpret in non-standard domain

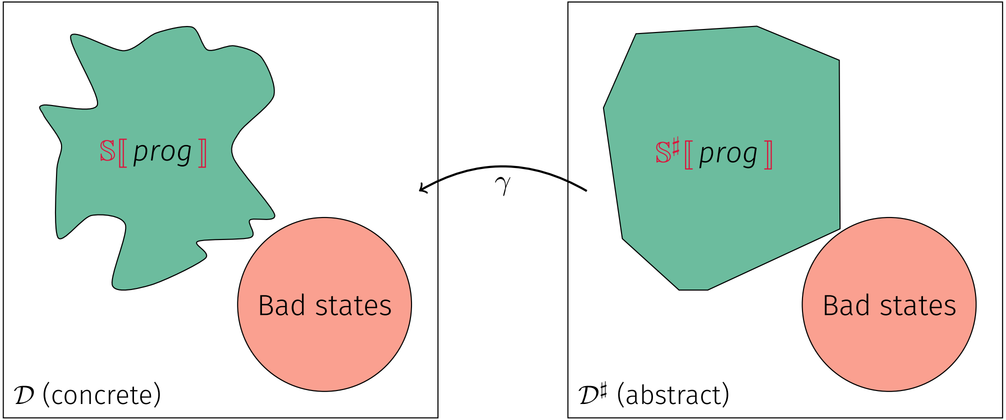
Program proved safe

Approximating a Complex World, with Guarantees

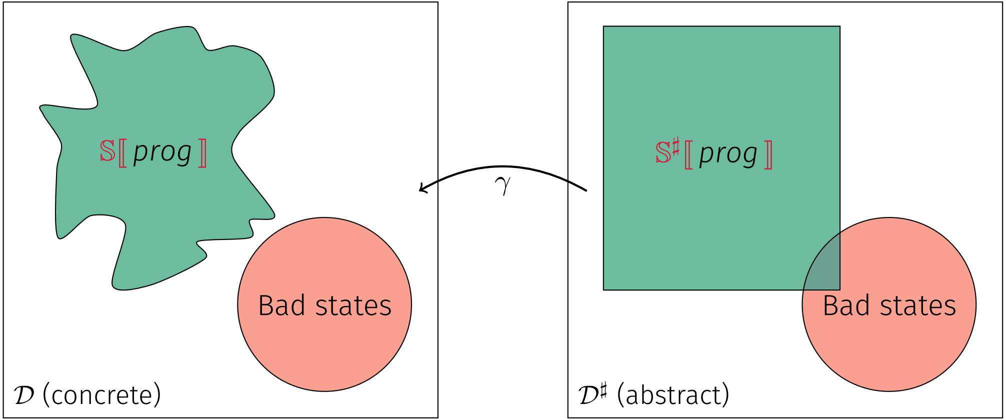

$\mathbb{S}[\![prog]\!]$
Bad states
$\mathcal{D}$ (concrete)
γ
$\mathbb{S}^\sharp[\![prog]\!]$
Bad states
$\mathcal{D}^\sharp$ (abstract)

True alarm

Approximating a Complex World, with Guarantees

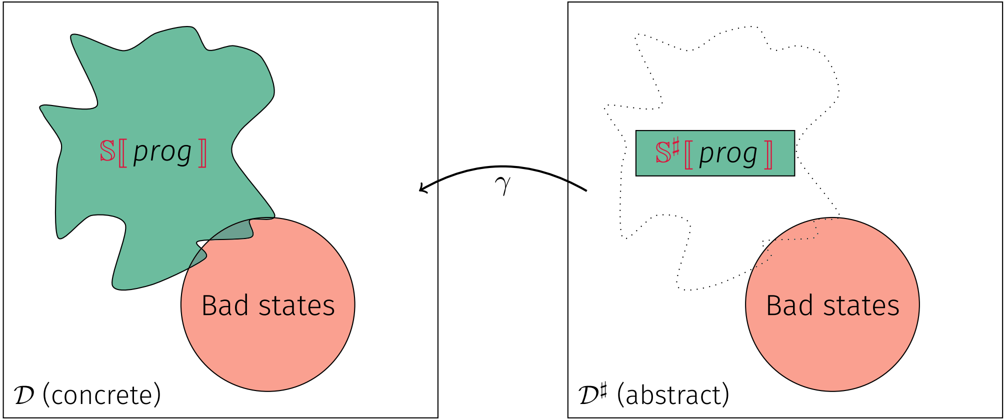

$\mathbb{S}[\![\, prog \,]\!]$
Bad states
$\mathcal{D}$ (concrete)
γ
$\mathbb{S}^\sharp[\![\, prog \,]\!]$
Bad states
$\mathcal{D}^\sharp$ (abstract)

Approximating a Complex World, with Guarantees


$\mathbb{S}[\![\, prog \,]\!]$
Bad states
$\mathcal{D}$ (concrete)
γ
$\mathbb{S}^\sharp[\![\, prog \,]\!]$
Bad states
$\mathcal{D}^\sharp$ (abstract)

False alarm (Abstraction too coarse)

Approximating a Complex World, with Guarantees


$\mathbb{S}[\![\, prog \,]\!]$
Bad states
$\mathcal{D}$ (concrete)
γ
$\mathbb{S}^\sharp[\![\, prog \,]\!]$
Bad states
$\mathcal{D}^\sharp$ (abstract)

Unsound analysis
(shouldn’t happen)

An AI primer

Key Ingredients

Merging Everything to Scale

Merging States

```
int x = rand();
```

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World
  - Set of program states $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World
  - Set of program states $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$
  - $\Sigma = \{\, x \mapsto n \mid 0 \leq n < 2^{31} \,\}$

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World
  - Set of program states $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$
  - $\Sigma = \{\, x \mapsto n \mid 0 \leq n < 2^{31} \,\}$
- Abstract World

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World
  - Set of program states $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$
  - $\Sigma = \{ x \mapsto n \mid 0 \leq n < 2^{31} \}$
- Abstract World
  - Represent multiple concrete states at once $\mathcal{V} \to \texttt{Intervals}$

Merging Everything to Scale

Merging States

```
int x = rand();
```

- Concrete World
  - Set of program states $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$
  - $\Sigma = \{ x \mapsto n \mid 0 \leq n < 2^{31} \}$
- Abstract World
  - Represent multiple concrete states at once $\mathcal{V} \to \texttt{Intervals}$
  - $\sigma^\sharp = x \mapsto [0, 2147483647]$

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Precision tradeoffs

⚠ computing an over-approximation, potential imprecision

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Precision tradeoffs

⚠ computing an over-approximation, potential imprecision

- Concrete $\Sigma = \{ x \mapsto n \mid -1 \leq n \leq 11 \land n \neq 10 \}$

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Precision tradeoffs

⚠ computing an over-approximation, potential imprecision

- Concrete $\Sigma = \{ x \mapsto n \mid -1 \leq n \leq 11 \land n \neq 10 \}$
- Abstract $\sigma^\sharp = x \mapsto [-1, 11]$

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Precision tradeoffs

⚠ computing an over-approximation, potential imprecision

- Concrete $\Sigma = \{ x \mapsto n \mid -1 \leq n \leq 11 \land n \neq 10 \}$
- Abstract $\sigma^\sharp = x \mapsto [-1, 11]$

$\Longrightarrow$ may require better abstractions!

Merging Everything to Scale (II)

Merging Paths

```
int x = rand(); if(x > 10) { x = 11; } else { x--; }; print(x);
```

- Contrary to symbolic execution, merge paths
- Rely on least upper bound operator ($\sqcup$) and lattice structure

Precision tradeoffs

⚠ computing an over-approximation, potential imprecision

- Concrete $\Sigma = \{ x \mapsto n \mid -1 \leq n \leq 11 \wedge n \neq 10 \}$
- Abstract $\sigma^\sharp = x \mapsto [-1, 11]$

$\implies$ may require better abstractions!

Merging can also be applied to arrays, . . .

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

Widening – Generalization Operator

```c
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!
- ❗ large nb(iterations)

Widening – Generalization Operator

```c
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of **i** in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

❗ large nb(iterations)

Introduce generalization operator ∇

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- ▶ Stabilization reached!

❗ large nb(iterations)

Introduce generalization operator ∇

- ▶ Over-approximating least upper bound $\sqcup$

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

❗ large nb(iterations)

Introduce generalization operator ∇

- Over-approximating least upper bound $\sqcup$
- Ensures finite termination of loop iterations

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of `i` in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

❗ large nb(iterations)

Introduce generalization operator ∇

- Over-approximating least upper bound $\sqcup$
- Ensures finite termination of loop iterations
- nb(iterations) does not depend on loop bound

Widening – Generalization Operator

```
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of **i** in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

❗ large nb(iterations)

Introduce generalization operator ∇

- Over-approximating least upper bound $\sqcup$
- Ensures finite termination of loop iterations
- nb(iterations) does not depend on loop bound

Iteration	Values of **i** in loop
0	[0, 0]
1	[0, 1]
2	$[0, 0] \nabla [0, 1] = [0, +\infty]$
3	$[0, +\infty]$

Widening – Generalization Operator

```c
int i = 0;
while(i < 100) {
  i++;
}
```

Iteration	Values of **i** in loop
0	[0, 0]
1	[0, 1]
...	...
99	[0, 99]
100	[0, 99]

- Stabilization reached!

! large nb(iterations)

Introduce generalization operator ∇

- Over-approximating least upper bound $\sqcup$
- Ensures finite termination of loop iterations
- nb(iterations) does not depend on loop bound

Iteration	Values of **i** in loop
0	[0, 0]
1	[0, 1]
2	$[0, 0]\ \nabla [0, 1] = [0, +\infty]$
3	$[0, +\infty]$

Precision can be recovered through decreasing iterations

$\implies \mathtt{i} = [0, 99]$

An overview of Mopsa

Mopsa

Modular Open Platform for Static Analysis [Jou+19]

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

Mopsa

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

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

Goals

- Explore new designs

  Including multi-language support

Mopsa

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

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

Goals

- Explore new designs
  - Including multi-language support
- Ease development of relational static analyses
  - High expressivity

Mopsa

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

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

Goals

- Explore new designs

  Including multi-language support
- Ease development of relational static analyses

  High expressivity
- Open-source (LGPL)

Mopsa

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

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

Goals

- Explore new designs

  Including multi-language support
- Ease development of relational static analyses

  High expressivity
- Open-source (LGPL)
- Can be used as an experimentation platform

Contributors (2018–2025, 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 (2018–2025, 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.

An overview of Mopsa

Key design decisions

Mopsa design

Analysis = composition of abstract domains

Mopsa design

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

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
- allows reuse of domains across languages

Mopsa design

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

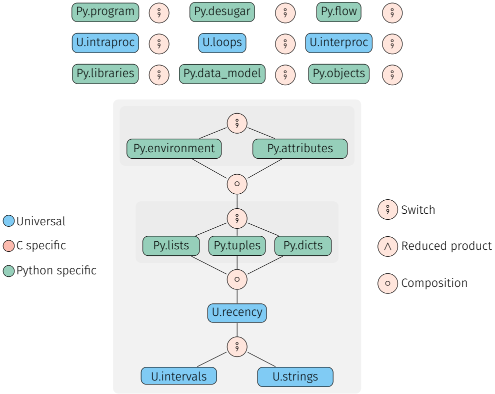
- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages
- defined as json files in `share/mopsa/configs`

Mopsa design

Analysis = composition of abstract domains

unified domain signature ⟹ 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`



Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Mopsa

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Mopsa

- No loss of precision from the frontend

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Mopsa

- No loss of precision from the frontend

By default, 3-address code may result in precision loss [NP18]

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Mopsa

- No loss of precision from the frontend

  By default, 3-address code may result in precision loss [NP18]

- Various programming paradigms supported!

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

Example: Infer's IR has five (!) constructors

Mopsa

- ▶ No loss of precision from the frontend

  By default, 3-address code may result in precision loss [NP18]
- ▶ Various programming paradigms supported!
- ▶ All constructs have to be handled – but rewritings are possible

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

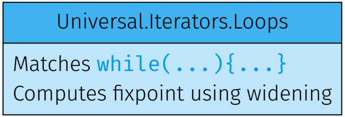
Example: Infer's IR has five (!) constructors

Mopsa

- No loss of precision from the frontend

  By default, 3-address code may result in precision loss [NP18]
- Various programming paradigms supported!
- All constructs have to be handled – but rewritings are possible
- A single AST type which can be extended for new languages

Dynamic, semantic iterators with delegation



Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation

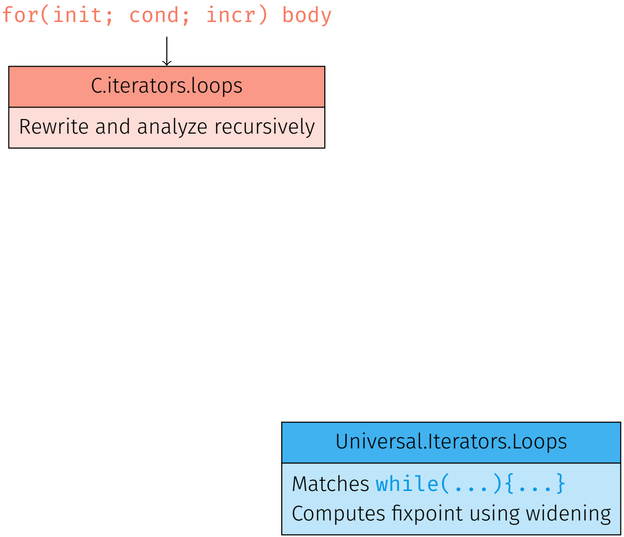
```
for(init; cond; incr) body
```

Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation



`for(init; cond; incr) body`

C.iterators.loops

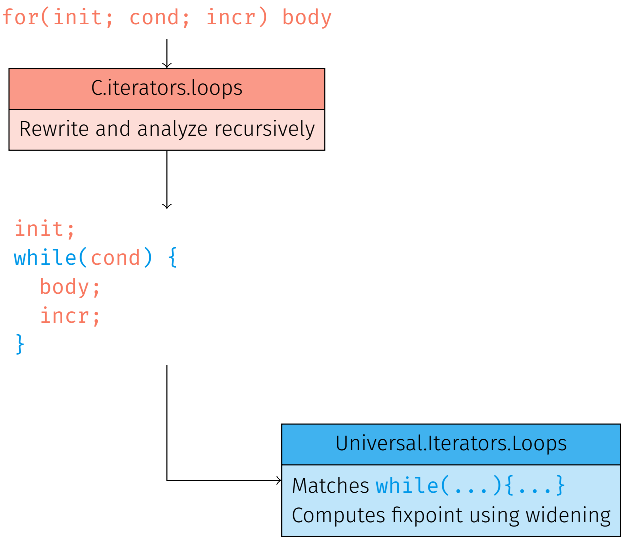
Rewrite and analyze recursively

Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation



```
for(init; cond; incr) body
```

C.iterators.loops

Rewrite and analyze recursively

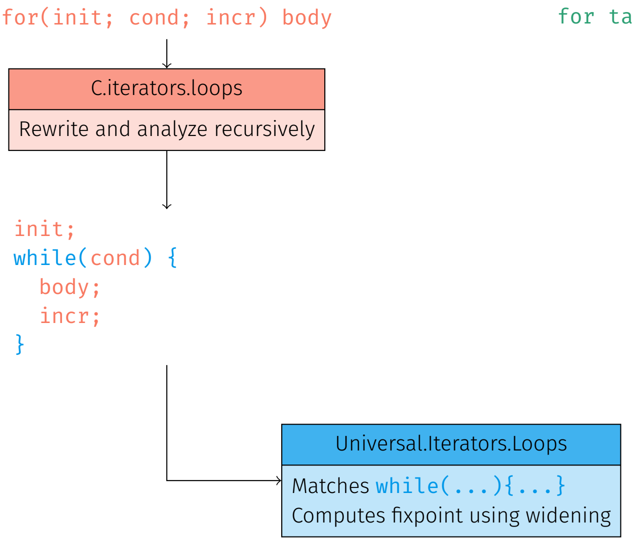
```
init;
while(cond) {
  body;
  incr;
}
```

Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation



```
for(init; cond; incr) body
```

C.iterators.loops

Rewrite and analyze recursively

```
init;
while(cond) {
  body;
  incr;
}
```

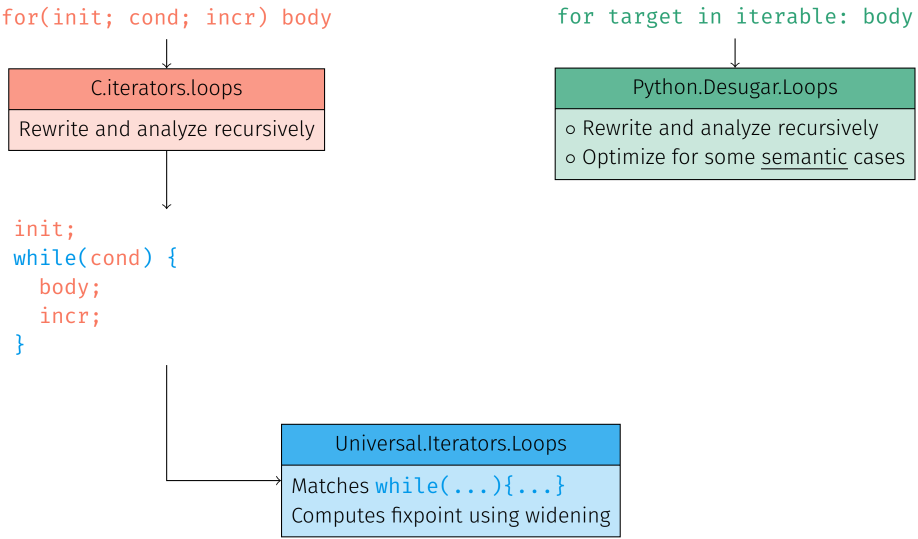
Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

```
for target in iterable: body
```

Dynamic, semantic iterators with delegation



```
for(init; cond; incr) body
```

C.iterators.loops

Rewrite and analyze recursively

```
init;
while(cond) {
  body;
  incr;
}
```

```
for target in iterable: body
```

Python.Desugar.Loops

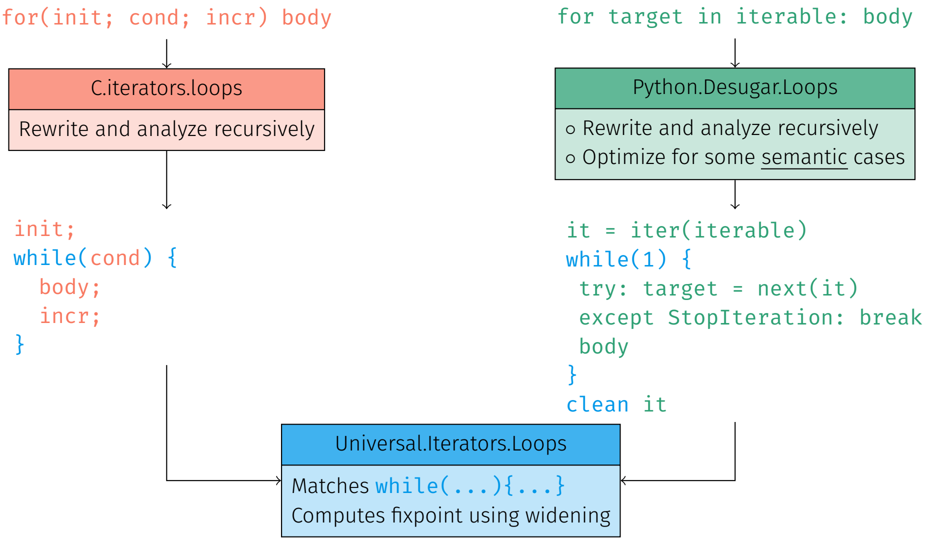
- ○ Rewrite and analyze recursively
- ○ Optimize for some semantic cases

Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation



```
for(init; cond; incr) body
```

C.iterators.loops

Rewrite and analyze recursively

```
init;
while(cond) {
  body;
  incr;
}
```

```
for target in iterable: body
```

Python.Desugar.Loops

- Rewrite and analyze recursively
- Optimize for some semantic cases

```
it = iter(iterable)
while(1) {
 try: target = next(it)
 except StopIteration: break
 body
}
clean it
```

Universal.Iterators.Loops

Matches `while(...){...}`

Computes fixpoint using widening

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

$i \in [0, 20]$, $\underline{len}(a) \in [10, 20]$, unable to prove safe access ✗

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

$i \in [0, 20]$, $\underline{len}(a) \in [10, 20]$, unable to prove safe access ✗

Relational domains to the rescue

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

i ∈ [0, 20], $\underline{\text{len}}(a) \in [10, 20]$, unable to prove safe access ✗

Relational domains to the rescue

- Able to express relationships between variables, e.g: $0 \leq i < \underline{\text{len}}(a) \leq 20$

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

i $\in [0, 20]$, $\underline{\text{len}}(a) \in [10, 20]$, unable to prove safe access ✗

Relational domains to the rescue

- Able to express relationships between variables, e.g: $0 \leq i < \underline{\text{len}}(a) \leq 20$
- Polyhedra domain [CH78; BHZ08; BZ20] $\sum_i \alpha_i V_i \leq \beta_i$

Expressivity through relational domains

Motivational example

```
// Hyp: a array of size len(a) ∈ [10, 20]
s = 0;
for(int i = 0; i < len(a); i++) {
  s += a[i];
}
```

$i \in [0, 20]$, $\underline{len}(a) \in [10, 20]$, unable to prove safe access ✗

Relational domains to the rescue

- Able to express relationships between variables, e.g: $0 \leq i < \underline{len}(a) \leq 20$
- Polyhedra domain [CH78; BHZ08; BZ20] $\sum_i \alpha_i V_i \leq \beta_i$
- Bindings from the convenient Apron library [JM09]

Relational domains in Mopsa

Difficulties arising from relational domains

Relational domains in Mopsa

Difficulties arising from relational domains

- Computational cost at least $\mathcal{O}(|\mathcal{V}|^3)$

Relational domains in Mopsa

Difficulties arising from relational domains

- Computational cost at least $\mathcal{O}(|\mathcal{V}|^3)$
- Evaluating expressions into (abstract) values is not enough!

Relational domains in Mopsa

Difficulties arising from relational domains

- Computational cost at least $\mathcal{O}(|\mathcal{V}|^3)$
- Evaluating expressions into (abstract) values is not enough!
- Need to force cohabitation of variables

Relational domains in Mopsa

Difficulties arising from relational domains

- Computational cost at least $\mathcal{O}(|\mathcal{V}|^3)$
- Evaluating expressions into (abstract) values is not enough!
- Need to force cohabitation of variables

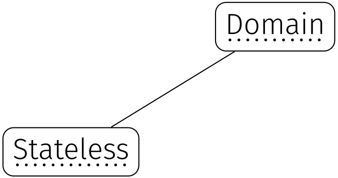
Mopsa relies on rewriting, symbolic expressions and ghost variables

to leverage relational domains.

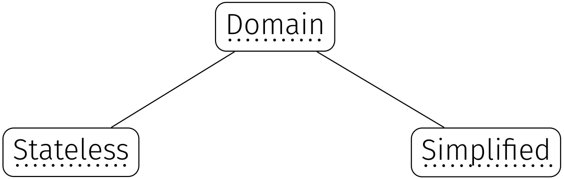
A zoology of domains and combinators in Mopsa

Domain

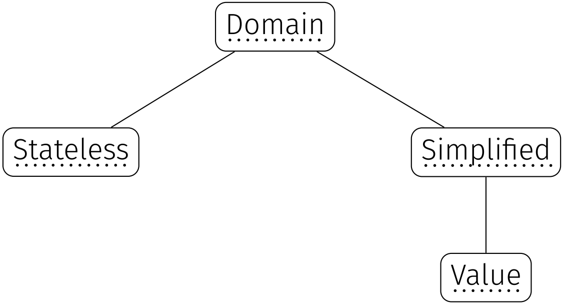
A zoology of domains and combinators in Mopsa



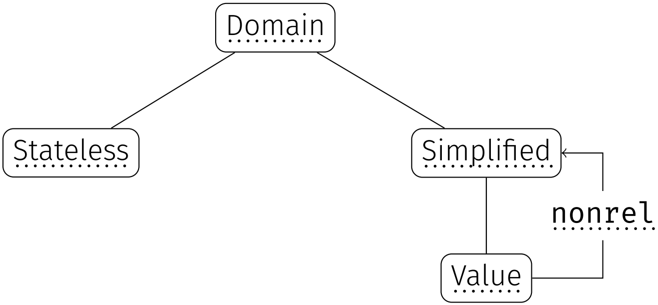
A zoology of domains and combinators in Mopsa



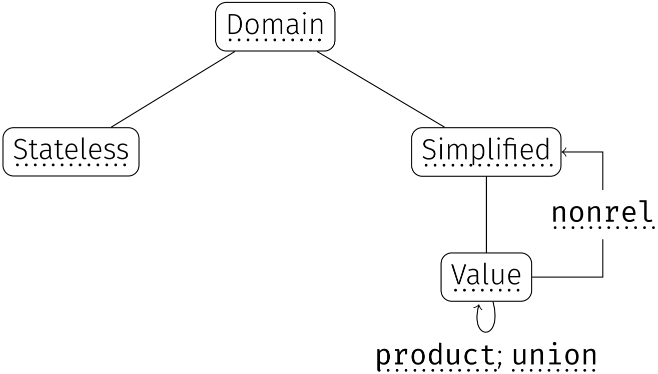
A zoology of domains and combinators in Mopsa



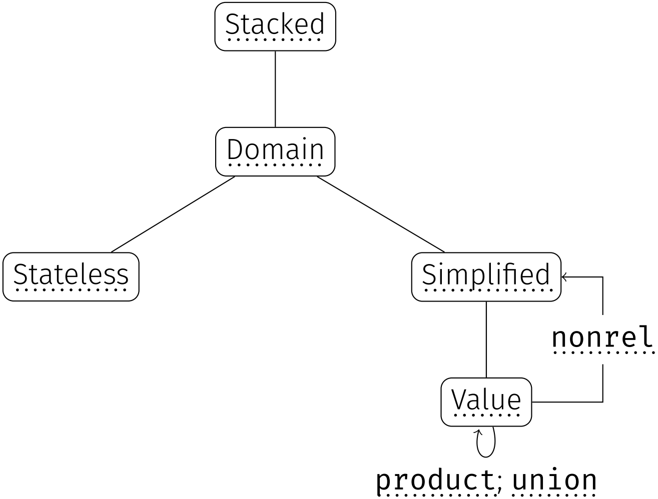
A zoology of domains and combinators in Mopsa



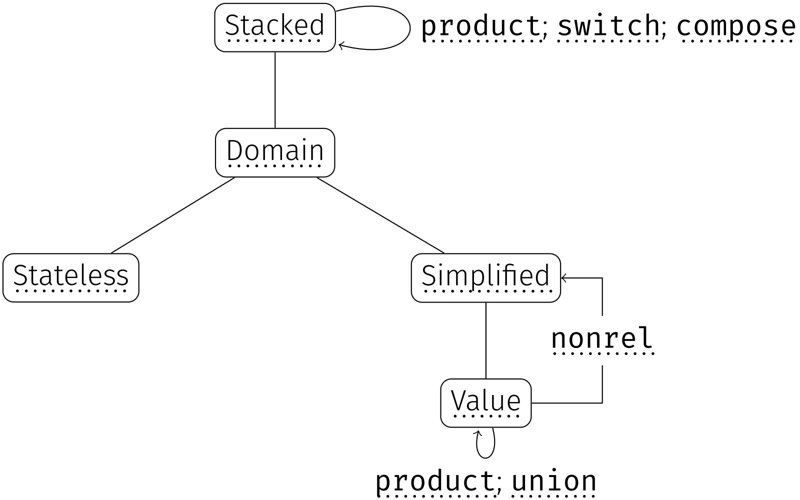
A zoology of domains and combinators in Mopsa



A zoology of domains and combinators in Mopsa



A zoology of domains and combinators in Mopsa



Overview of Mopsa

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

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

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` or `opam install mopsa`

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` or `opam install mopsa`

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

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

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

One AST to rule them all

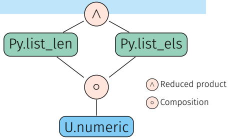
- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation



An overview of Mopsa

Works around Mopsa

Coreutils – Ouadjaout and Miné [OM20]

- Large support of `libc` through stubs

Coreutils – Ouadjaout and Miné [OM20]

- Large support of `libc` through stubs
- Check for all C runtime errors

Coreutils – Ouadjaout and Miné [OM20]

- Large support of `libc` through stubs
- Check for all C runtime errors
- Ability to analyze real-world programs

Coreutils – Ouadjaout and Miné [OM20]

- Large support of `libc` through stubs
- Check for all C runtime errors
- Ability to analyze real-world programs

Benchmark	Time	Selectivity	# checks
basename	33.79s	98.65%	11,731
dirname	21.68s	99.61%	11,307
echo	19.26s	99.43%	11,010
false	14.50s	99.72%	10,774
pwd	22.04s	99.62%	11,502
rmdir	39.00s	99.22%	11,699
sleep	23.79s	99.46%	11,546
tee	35.69s	98.76%	12,057
timeout	32.28s	98.51%	12,420
true	9.55s	99.72%	10,774
uname	20.61s	99.52%	11,943
users	20.82s	99.06%	11,668
whoami	13.03s	99.66%	11,329

Multilanguage Analysis – Monat, Ouadjaout, and Miné [MOM21]

Assessment 20% of the 200 most popular Python libraries rely on C code

Multilanguage Analysis – Monat, Ouadjaout, and Miné [MOM21]

Assessment 20% of the 200 most popular Python libraries rely on C code

Dangers: different values ($\mathbb{Z}$ vs. `Int32`); shared memory state

Multilanguage Analysis – Monat, Ouadjaout, and Miné [MOM21]

Assessment 20% of the 200 most popular Python libraries rely on C code

Dangers: different values ($\mathbb{Z}$ vs. `Int32`); shared memory state

Our approach: Combined analysis of C, Python and interface code

Library	C + Py. Loc	Tests	🕒/test	$\frac{\#\text{ proved checks}}{\#\text{ checks}}$ %	# checks
`noise`	1397	15/15	1.2s	99.7%	6690
`cdistance`	2345	28/28	4.1s	98.0%	13716
`llist`	4515	167/194	1.5s	98.8%	36255
`ahocorasick`	4877	46/92	1.2s	96.7%	6722
`levenshtein`	5798	17/17	5.3s	84.6%	4825
`bitarray`	5841	159/216	1.6s	94.9%	25566

Non-exploitability – Parolini and Miné [PM24]

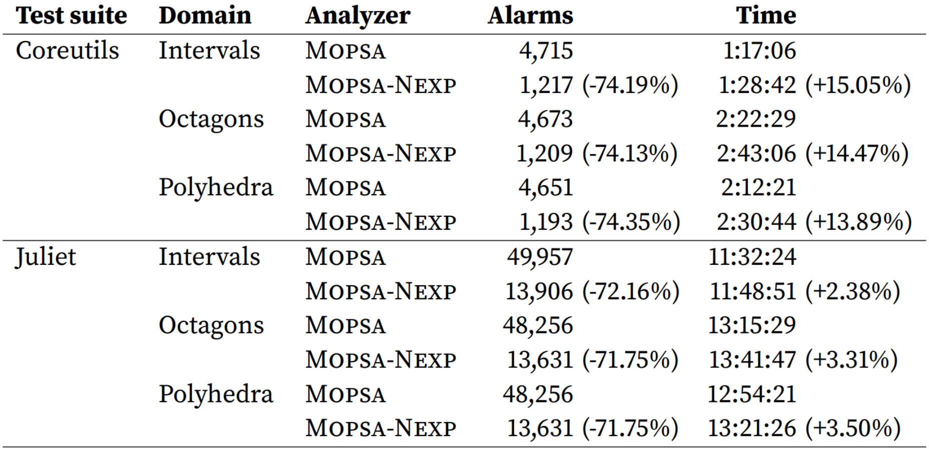
- Focus on bugs that a user can trigger through program interaction

Non-exploitability – Parolini and Miné [PM24]

- Focus on bugs that a user can trigger through program interaction
- Relies on combination of taint+value analysis

Non-exploitability – Parolini and Miné [PM24]

- Focus on bugs that a user can trigger through program interaction
- Relies on combination of taint+value analysis



Test suite	Domain	Analyzer	Alarms	Time
Coreutils	Intervals	Mopsa	4,715	1:17:06
		Mopsa-Nexp	1,217 (-74.19%)	1:28:42 (+15.05%)
	Octagons	Mopsa	4,673	2:22:29
		Mopsa-Nexp	1,209 (-74.13%)	2:43:06 (+14.47%)
	Polyhedra	Mopsa	4,651	2:12:21
		Mopsa-Nexp	1,193 (-74.35%)	2:30:44 (+13.89%)
Juliet	Intervals	Mopsa	49,957	11:32:24
		Mopsa-Nexp	13,906 (-72.16%)	11:48:51 (+2.38%)
	Octagons	Mopsa	48,256	13:15:29
		Mopsa-Nexp	13,631 (-71.75%)	13:41:47 (+3.31%)
	Polyhedra	Mopsa	48,256	12:54:21
		Mopsa-Nexp	13,631 (-71.75%)	13:21:26 (+3.50%)

Software Verification Competition [Mon+24]

- Tools have to

Software Verification Competition [Mon+24]

- Tools have to
  - Decide whether a program is correct (large penalties if wrong)

Software Verification Competition [Mon+24]

- Tools have to
  - Decide whether a program is correct (large penalties if wrong)
  - Within limited machine resources (15 minutes CPU time, 8GB RAM)

Software Verification Competition [Mon+24]

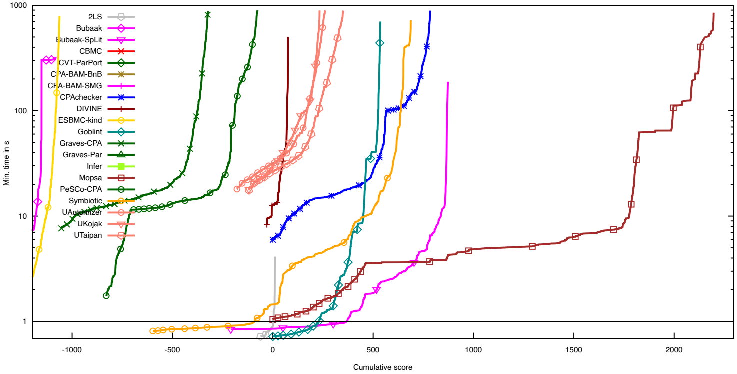
- Tools have to
  - Decide whether a program is correct (large penalties if wrong)
  - Within limited machine resources (15 minutes CPU time, 8GB RAM)
- Corpus of $\simeq$ 23,000 C benchmarks, now acts as a reference

Software Verification Competition [Mon+24]

- Tools have to
  - Decide whether a program is correct (large penalties if wrong)
  - Within limited machine resources (15 minutes CPU time, 8GB RAM)
- Corpus of $\simeq$ 23,000 C benchmarks, now acts as a reference
- For our second participation, Mopsa won the “Software Systems” track!

Software Verification Competition [Mon+24]

- Tools have to
  - Decide whether a program is correct (large penalties if wrong)
  - Within limited machine resources (15 minutes CPU time, 8GB RAM)
- Corpus of $\simeq$ 23,000 C benchmarks, now acts as a reference
- For our second participation, Mopsa won the “Software Systems” track!



Summary of analyses

Languages

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

Summary of analyses

Languages

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

Multilanguage Python+C [MOM21]

Summary of analyses

Languages

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

Multilanguage Python+C [MOM21]

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

Summary of analyses

Languages

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

Multilanguage Python+C [MOM21]

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

Properties

Summary of analyses

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

Summary of analyses

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]

Summary of analyses

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]

Summary of analyses

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]

Summary of analyses

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 [MM24a; MM24b]

Easing maintenance and implementation

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

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

```
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

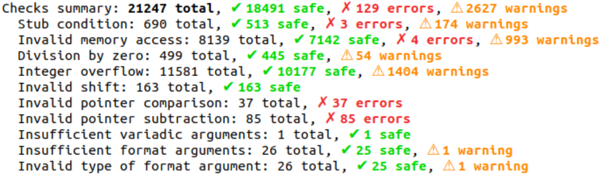
- Easy to implement
- “2,756 alarms” ⇝ 87% checks proved correct – “selectivity”
- ~~Program size~~ ⇝ “expression complexity”

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

```c
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]

Easing maintenance and implementation

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

Easing maintenance and implementation

Easing debugging

Where static analyzers usually start from

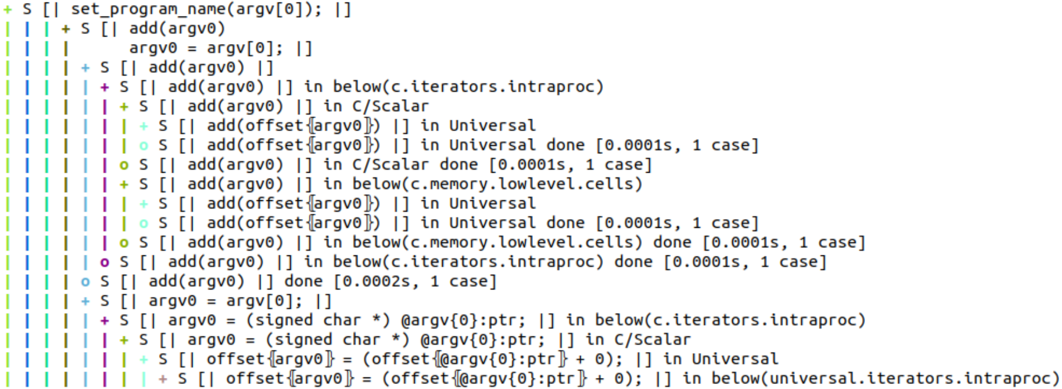
- Analysis output Too coarse

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 back to statement generating first alarm
- Navigation

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

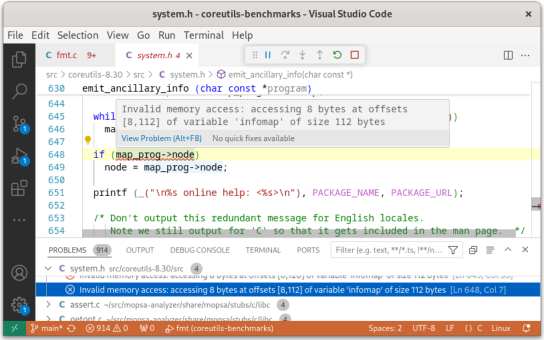
GDB-like interface to the abstract interpretation of the program

Demo!

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

IDE support

- Language Server Protocol for linters (report alarms)



system.h - coreutils-benchmarks - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C fmt.c 9+
C system.h 4
src > coreutils-8.30 > src > C system.h > emit_ancillary_info(char const *)
View Problem (Alt+F8)
No quick fixes available
PROBLEMS 914
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/n…
C system.h src/coreutils-8.30/src 4
C assert.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4
C getopt.c ~/src/mopsa-analyzer/share/mopsa/stubs/c/libc 4
main* ⊗ 914 ⚠ 0 0 fmt (coreutils-benchmarks) Spaces: 2 UTF-8 LF { } C Linux

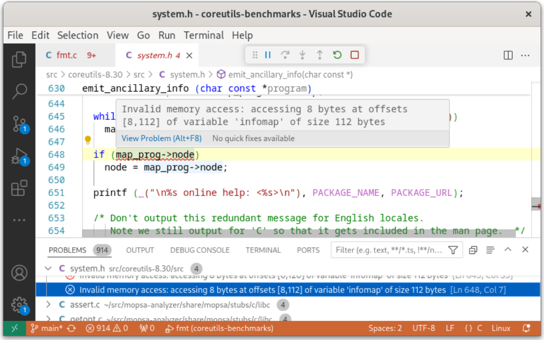
```
emit_ancillary_info (char const *program)

  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
```

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

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
C fmt.c 9+
C system.h 4
src > coreutils-8.30 > src > C 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

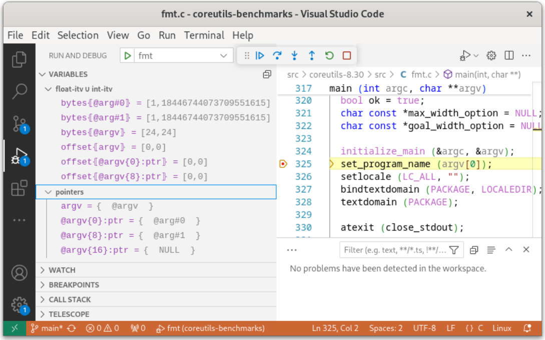
View Problem (Alt+F8) No quick fixes available

C 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]

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

C 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
WATCH
BREAKPOINTS
CALL STACK
TELESCOPE
src > coreutils-8.30 > src > C fmt.c > main(int, char **)
Filter (e.g. text, **/*.ts, !**/...
main* ⊗ 0 ⚠ 0 0 fmt (coreutils-benchmarks)
Ln 325, Col 2 Spaces: 2 UTF-8 LF {} C Linux

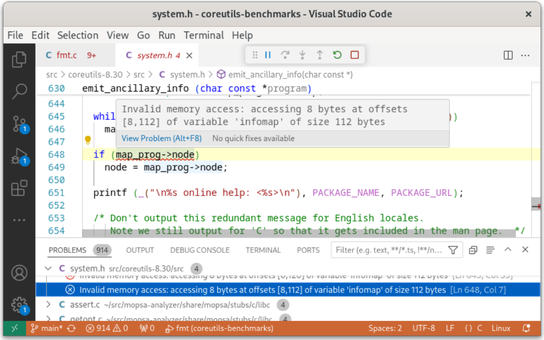
```
float-itv ∪ int-itv
  bytes⟦@arg#0⟧ = [1,18446744073709551615]
  bytes⟦@arg#1⟧ = [1,18446744073709551615]
  bytes⟦@argv⟧ = [24,24]
  offset⟦argv⟧ = [0,0]
  offset⟦@argv{0}:ptr⟧ = [0,0]
  offset⟦@argv{8}:ptr⟧ = [0,0]
pointers
  argv = {  @argv  }
  @argv{0}:ptr = {  @arg#0  }
  @argv{8}:ptr = {  @arg#1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

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 *)

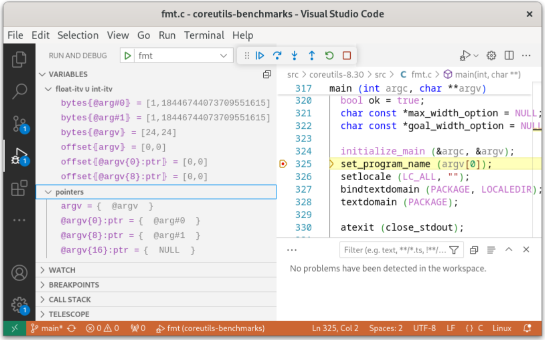
```
emit_ancillary_info (char const *program)

  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 [0,120] of variable 'infomap' of size 112 bytes [Ln 645, Col 33]
  - 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

main*
⊗ 914 ⚠ 0
0
fmt (coreutils-benchmarks)
Spaces: 2
UTF-8
LF
{} C
Linux


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 **)

```
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

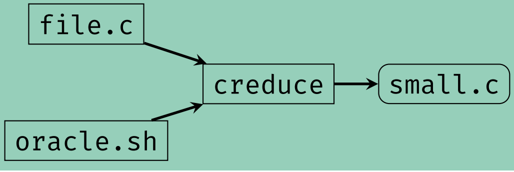
- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases
- Debugging extremely difficult: size of the program, analysis time

Testcase reduction

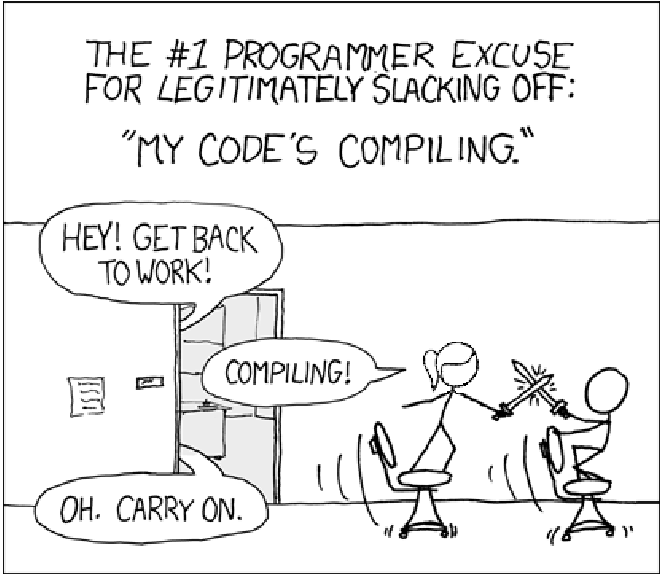
Motivation

- 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

Easing maintenance and implementation

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

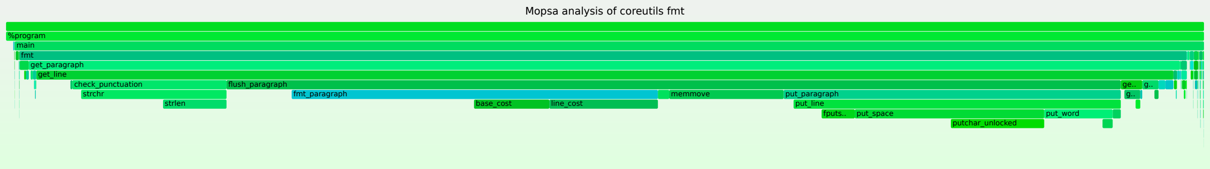
Standard profiling

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

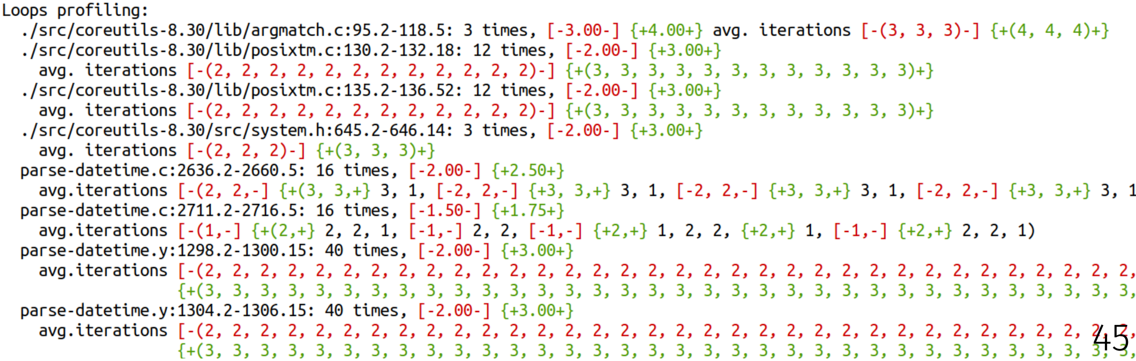
Apron vs PPLite on Coreutils touch

- 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,
```

Easing maintenance and implementation

Related work

Related work

Lots of folklore

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP [LDB19]

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP [LDB19]
- Testing the soundness and precision of static analyzers [KCW19; TLR20; MVR23; Kai+24; Fle+24]

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP [LDB19]
- Testing the soundness and precision of static analyzers [KCW19; TLR20; MVR23; Kai+24; Fle+24]
- Debugging:

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP [LDB19]
- Testing the soundness and precision of static analyzers [KCW19; TLR20; MVR23; Kai+24; Fle+24]
- Debugging:
  - Mixing concrete+abstract [MVR23]

Related work

Lots of folklore

- First work, applying and combining S.E. techniques for TAJS [AMN17]
- Frama-C & Goblint: flamegraphs, testcase reduction
- Leveraging LSP [LDB19]
- Testing the soundness and precision of static analyzers [KCW19; TLR20; MVR23; Kai+24; Fle+24]
- Debugging:
  - Mixing concrete+abstract [MVR23]
  - Sound abstract debugger in Goblint [Hol+24a; Hol+24b]

Conclusion

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 around maintenance

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 around maintenance

- Handling the exponential number of configurations

Conclusion

Our current approach

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

Ongoing challenges around maintenance

- 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 around maintenance

- 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*"
  ⇝ "new tools on abstract execution of *target program*"

Ongoing challenges around maintenance

- 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`.

[BBY17] S. Blazy, D. Bühler, and B. Yakobowski. **“Structuring Abstract Interpreters Through State and Value Abstractions”.** In: LNCS. Springer, 2017, pp. 112–130.

References – II

[Ber+10] J. Bertrane et al. “Static analysis and verification of aerospace software by abstract interpretation”. In: AIAA-2010-3385. 2010.

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. “The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems”. In: Sci. Comput. Program. 1-2 (2008), pp. 3–21.

[BZ20] Anna Becchi and Enea Zaffanella. “PPLite: Zero-overhead encoding of NNC polyhedra”. In: Inf. Comput. (2020), p. 104620. DOI: 10.1016/J.IC.2020.104620.

[CH78] Patrick Cousot and Nicolas Halbwachs. “Automatic Discovery of Linear Restraints Among Variables of a Program”. In: 1978.

References – III

[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.

[Fle+24] Markus Fleischmann et al. **“Constraint-Based Test Oracles for Program Analyzers”.** In: ed. by Vladimir Filkov, Baishakhi Ray, and Minghui Zhou. ACM, 2024, pp. 344–355. DOI: `10.1145/3691620.3695035`.

References – IV

[Hol+24a] Karoliine Holter et al. **"Abstract Debuggers: Exploring Program Behaviors using Static Analysis Results".** In: Onward! '24. Pasadena, CA, USA: Association for Computing Machinery, 2024, pp. 130–146. DOI: `10.1145/3689492.3690053`.

[Hol+24b] Karoliine Holter et al. **"Abstract Debugging with GobPie".** In: ed. by Elisa Gonzalez Boix and Christophe Scholliers. ACM, 2024, pp. 32–33. DOI: `10.1145/3678720.3685320`.

[JM09] Bertrand Jeannet and Antoine Miné. **"Apron: A Library of Numerical Abstract Domains for Static Analysis".** In: Lecture Notes in Computer Science. Springer, 2009, pp. 661–667. DOI: `10.1007/978-3-642-02658-4_52`.

References – V

[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.

[Kai+24] David Kaindlstorfer et al. **“Interrogation Testing of Program Analyzers for Soundness and Precision Issues”.** In: ed. by Vladimir Filkov, Baishakhi Ray, and Minghui Zhou. ACM, 2024, pp. 319–330. DOI: `10.1145/3691620.3695034`.

References – VI

[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.

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

References – VII

[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`.

[Min17] Antoine Miné. **“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation”.** In: Found. Trends Program. Lang. 3-4 (2017), pp. 120–372.

[MM24a] 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`.

References – VIII

[MM24b] Marco Milanese and Antoine Miné. “Under-Approximating Memory Abstractions”. In: ed. by Roberto Giacobazzi and Alessandra Gorla. Lecture Notes in Computer Science. Springer, 2024, pp. 300–326. DOI: 10.1007/978-3-031-74776-2_12.

[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.

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

References – IX

[Mon+24] Raphaël Monat et al. “Mopsa-C: Improved Verification for C Programs, Simple Validation of Correctness Witnesses (Competition Contribution)”. In: Lecture Notes in Computer Science. Springer, 2024, pp. 387–392.

[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 – X

[NP18] Kedar S. Namjoshi and Zvonimir Pavlinovic. **“The Impact of Program Transformations on Static Program Analysis”.** In: ed. by Andreas Podelski. Lecture Notes in Computer Science. Springer, 2018, pp. 306–325. DOI: `10.1007/978-3-319-99725-4_19`.

[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: Lecture Notes in Computer Science. Springer, 2024, pp. 314–337.

References – XI

[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.

[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.