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

Raphaël Monat – SyCoMoRES team, Lille

`rmonat.fr`

CUPLV
4 April 2025

Introduction

whoami

Research Scientist at Inria since Sep. 2022.

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

Other Research Interests within SyCoMoRES team

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

Other Research Interests within SyCoMoRES team

- Scheduling for real-time embedded systems

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

Other Research Interests within SyCoMoRES team

- Scheduling for real-time embedded systems
- Binary code analysis [Bal+19] (for worst-case execution time, security)

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

Other Research Interests within SyCoMoRES team

- Scheduling for real-time embedded systems
- Binary code analysis [Bal+19] (for worst-case execution time, security)
- Rocq and coinduction

whoami

Research Scientist at Inria since Sep. 2022.

Research Interests

- Static analysis: C, Python, multi-language paradigms
- Formal methods for public administrations

Automated Verification of Catala Programs

Other Research Interests within SyCoMoRES team

- Scheduling for real-time embedded systems
- Binary code analysis [Bal+19] (for worst-case execution time, security)
- Rocq and coinduction
- Type systems for privacy

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

Motivation

Sheer quantity of programs and changes during their life:

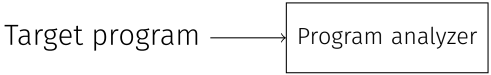
Automated analyses will help scaling up

Target program

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

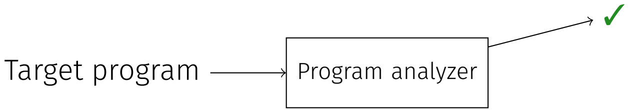


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

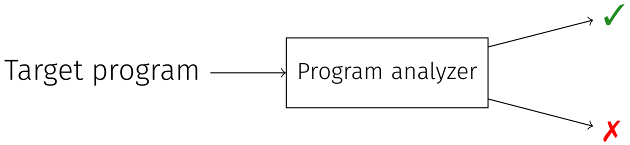


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

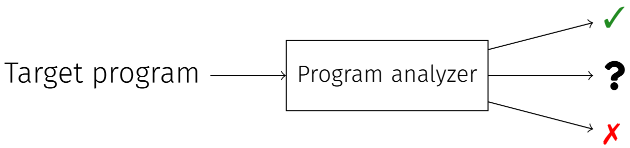
Automated analyses will help scaling up



Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up



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

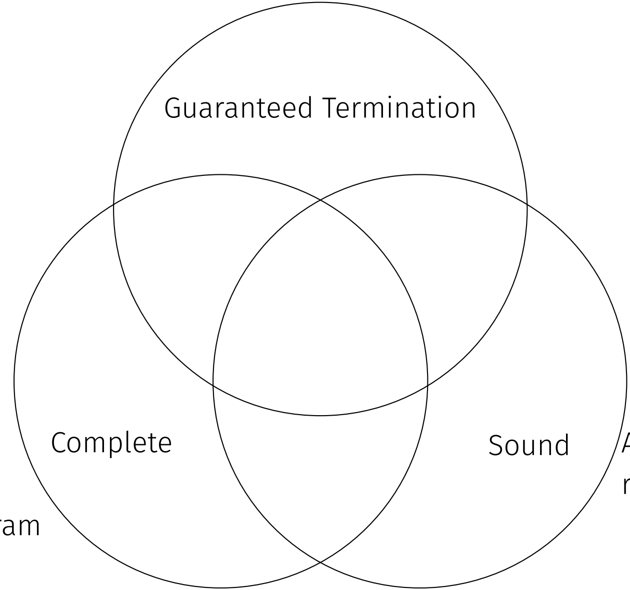
All errors reported by analyzer are replicable in program

Complete

Sound

All errors in program reported by analyzer

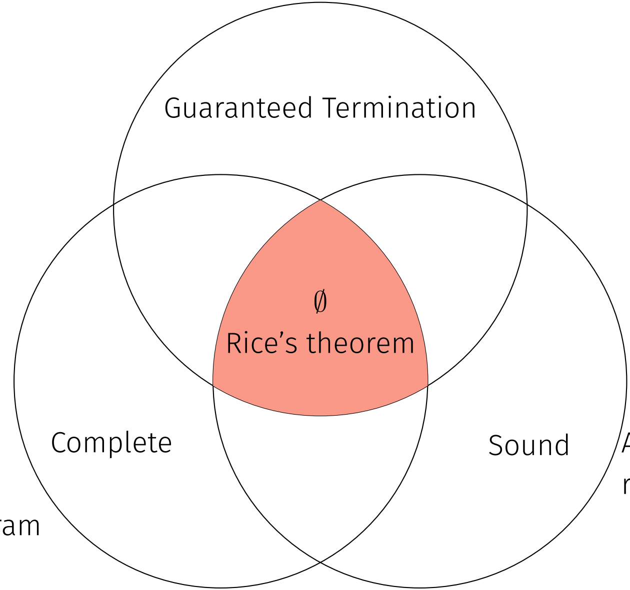
Turing & Rice to the Rescue



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

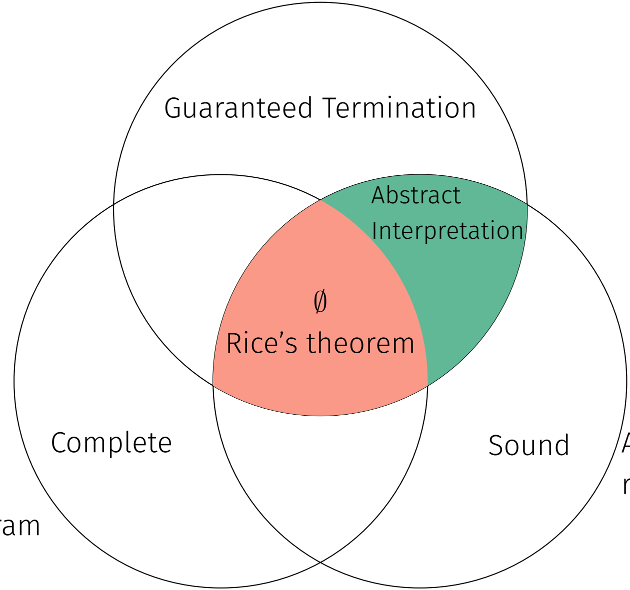
Turing & Rice to the Rescue (or not?)



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Turing & Rice to the Rescue (or not?)



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Academic research around static analysis

Academic research around static analysis

Ideal analyzer

Academic research around static analysis

Ideal analyzer

- Sound, precise and scalable

Academic research around static analysis

Ideal analyzer

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

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

Academic research around static analysis

Ideal analyzer

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

Implementation hurdles

- Debugging time-consuming

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

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

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

Outline

An overview of Mopsa

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 Open Platform for Static Analysis [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

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
- 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
- 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

Analysis = composition of abstract domains

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms

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

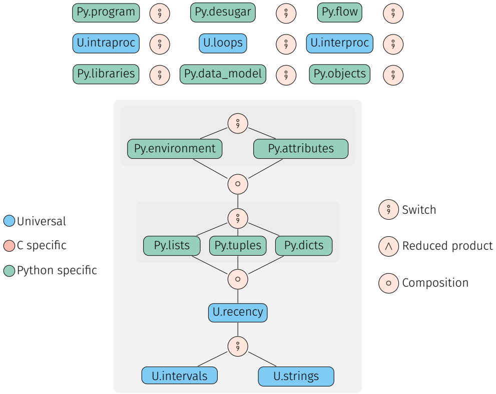
unified domain signature $\Longrightarrow$ iterators are abstract domains

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

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`



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

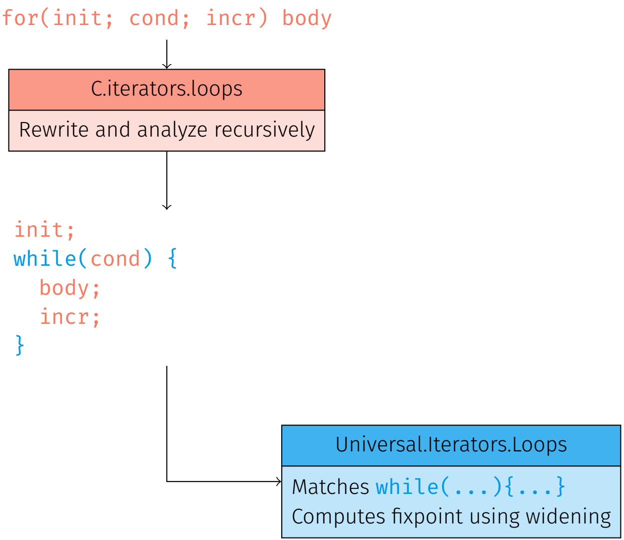
Rewrite and analyze recursively

Universal.Iterators.Loops

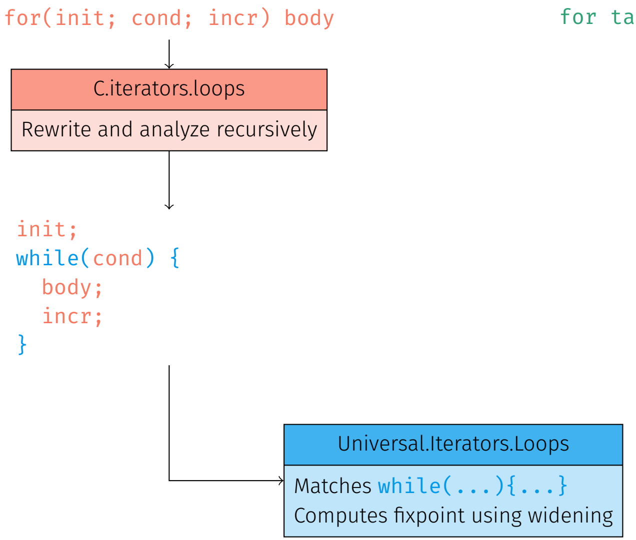
Matches `while(...){...}`

Computes fixpoint using widening

Dynamic, semantic iterators with delegation

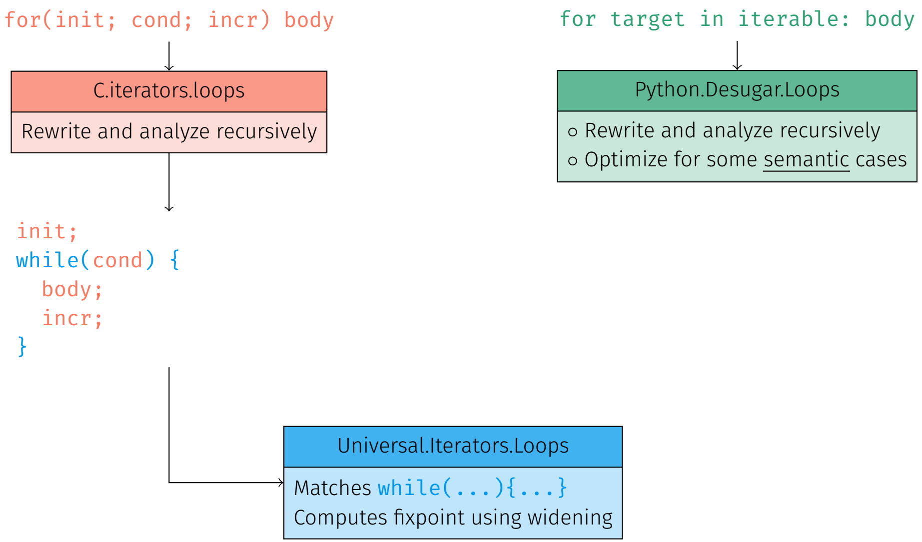


Dynamic, semantic iterators with delegation

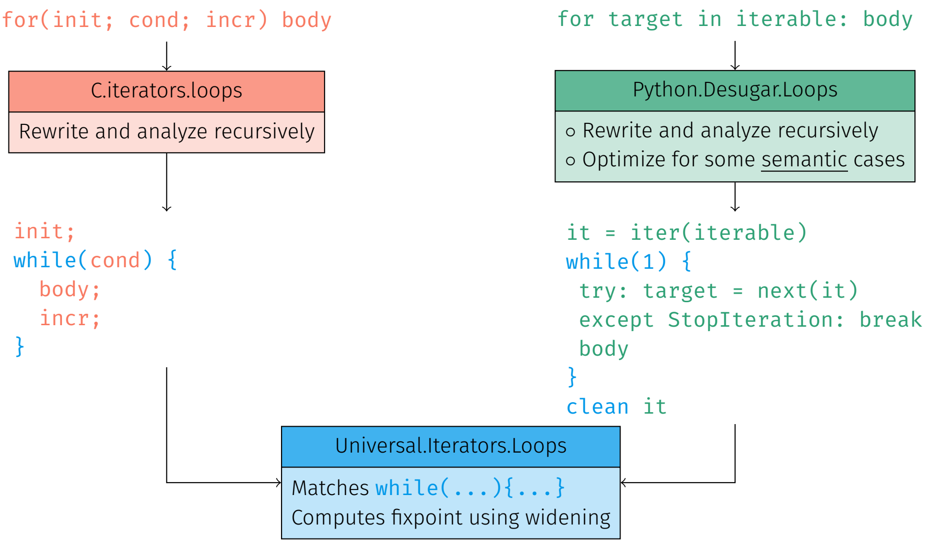


`for target in iterable: body`

Dynamic, semantic iterators with delegation



Dynamic, semantic iterators with delegation



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 ✗

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];
}
```

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

Relational domains to the rescue

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

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], len(a) ∈ [10, 20], unable to prove safe access ✗

Relational domains to the rescue

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

$i \in [0, 20]$, $\underline{\text{len}}(a) \in [10, 20]$, 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] $\sum_i \alpha_i V_i \leq \beta_i$
- 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.

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]

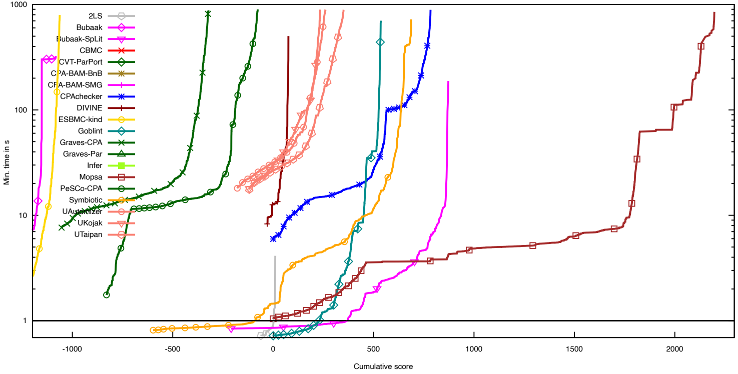
- Tools have to
  - Decide whether a program is correct (large penalties if wrong)
  - Within limited machine resources (15 minutes CPU time, 8GB RAM)
- Corpus of $\simeq$ 23,000 C benchmarks, now acts as a reference

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]

Implementation details

Ocaml tidbits

Mopsa is implemented in OCaml.

- Curried functions: `f x y z` $\rightsquigarrow f(x, y, z)$

Ocaml tidbits

Mopsa is implemented in OCaml.

- Curried functions: `f x y z` $\rightsquigarrow f(x, y, z)$
- `fun x -> e` $\Leftrightarrow \lambda x.e$

Ocaml tidbits

Mopsa is implemented in OCaml.

- Curried functions: `f x y z` $\rightsquigarrow f(x,y,z)$
- `fun x -> e` $\Leftrightarrow \lambda x.e$
- Variable binding `let x = e1 in e2`

Mopsa is implemented in OCaml.

- Curried functions: `f x y z` $\rightsquigarrow f(x, y, z)$
- `fun x -> e` $\Leftrightarrow \lambda x.e$
- Variable binding `let x = e1 in e2`
- Algebraic datatypes and pattern matching

```
type 'a option = None | Some of 'a

match e with
| None -> e1
| Some y -> e2 y
```

Ocaml tidbits

Mopsa is implemented in OCaml.

- Curried functions: `f x y z` $\rightsquigarrow f(x, y, z)$
- `fun x -> e` $\Leftrightarrow \lambda x.e$
- Variable binding `let x = e1 in e2`
- Algebraic datatypes and pattern matching

```
type 'a option = None | Some of 'a

match e with
| None -> e1
| Some y -> e2 y
```

Polymorphism = Type Variables

A new frontend cookbook

1. Parse: `register_frontend`, and in particular `parse_program`

A new frontend cookbook

1. Parse: `register_frontend`, and in particular `parse_program`
2. Create new AST nodes and their helper functions (printer, comparator, visitor)
   Future: `ppx_deriving` support

A new frontend cookbook

1. Parse: `register_frontend`, and in particular `parse_program`
2. Create new AST nodes and their helper functions (printer, comparator, visitor)
   Future: `ppx_deriving` support
3. Encode the semantics of the language in stateless domains

A new frontend cookbook

1. Parse: `register_frontend`, and in particular `parse_program`
2. Create new AST nodes and their helper functions (printer, comparator, visitor)
   Future: `ppx_deriving` support
3. Encode the semantics of the language in stateless domains
4. Abstract domains:

A new frontend cookbook

1. Parse: `register_frontend`, and in particular **`parse_program`**
2. Create new AST nodes and their helper functions (printer, comparator, visitor)
   Future: **`ppx_deriving`** support
3. Encode the semantics of the language in stateless domains
4. Abstract domains:
   - delegate to universal domains, or

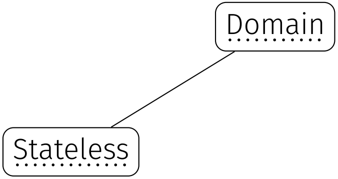
A new frontend cookbook

1. Parse: `register_frontend`, and in particular `parse_program`
2. Create new AST nodes and their helper functions (printer, comparator, visitor)
   Future: `ppx_deriving` support
3. Encode the semantics of the language in stateless domains
4. Abstract domains:
   - delegate to universal domains, or
   - add new custom domains. Can rely on predefined `Lattices`.

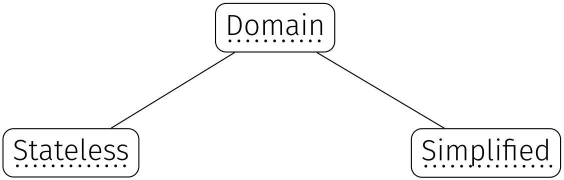
A zoology of domains and combinators in Mopsa

Domain

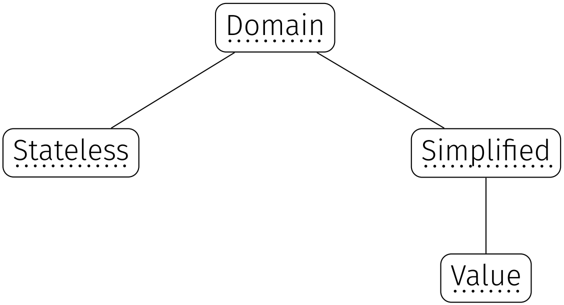
A zoology of domains and combinators in Mopsa



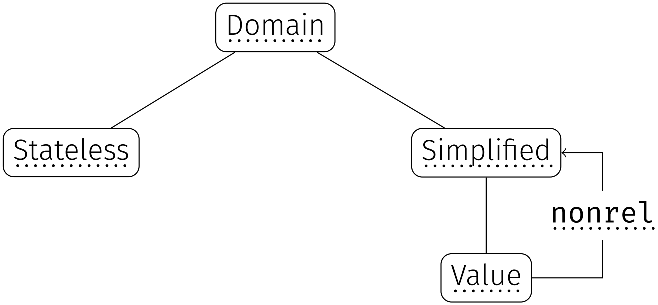
A zoology of domains and combinators in Mopsa



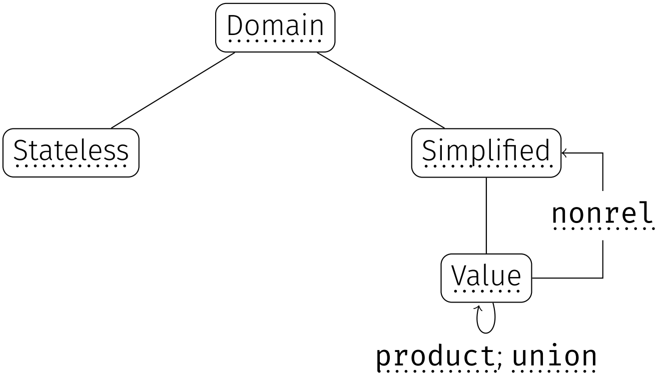
A zoology of domains and combinators in Mopsa



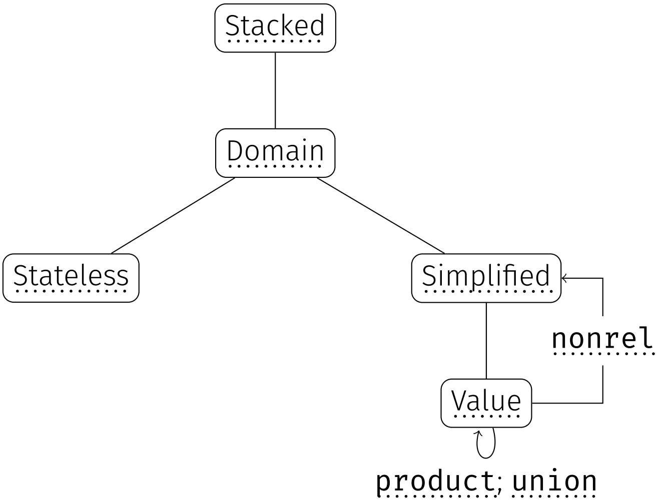
A zoology of domains and combinators in Mopsa



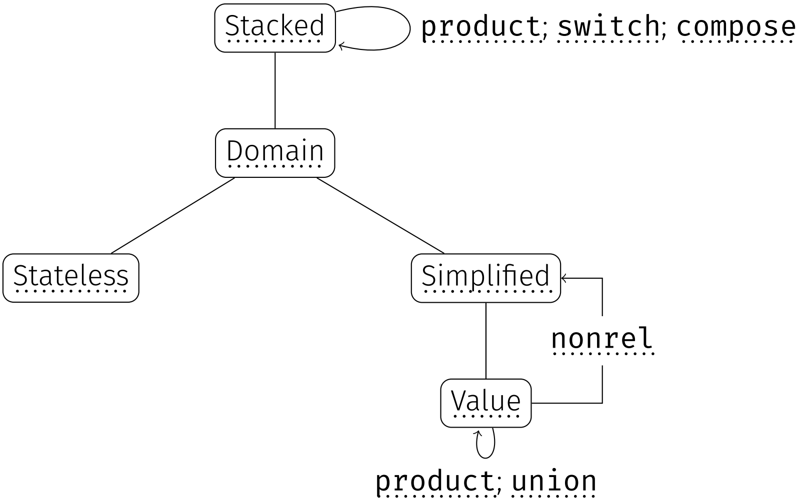
A zoology of domains and combinators in Mopsa



A zoology of domains and combinators in Mopsa



A zoology of domains and combinators in Mopsa



Abstract state & domain signature

Which type can we give to the abstract state?

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

- `type ('a, 'r) cases` as DNFs over `'a flow * 'r`

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

- `type ('a, 'r) cases` as DNFs over `'a flow * 'r`
- `Cases.singleton : 'r -> 'a flow -> ('a, 'r) cases`

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

- `type ('a, 'r) cases` as DNFs over `'a flow * 'r`
- `Cases.singleton : 'r -> 'a flow -> ('a, 'r) cases`
- Binding operator `cases >>$ fun r flow -> ...`

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

- `type ('a, 'r) cases` as DNFs over `'a flow * 'r`
- `Cases.singleton : 'r -> 'a flow -> ('a, 'r) cases`
- Binding operator `cases >>$ fun r flow -> ...`

  `>>$ : ('a,'r) cases -> ('r -> 'a flow -> ('a,'s) cases) -> ('a,'s) cases`

Abstract state & domain signature

Which type can we give to the abstract state?

- Polymorphism to the rescue: `'a` represents the abstract state
- Extended into `'a flow` to maintain additional info

Handling cases

- `type ('a, 'r) cases` as DNFs over `'a flow * 'r`
- `Cases.singleton : 'r -> 'a flow -> ('a, 'r) cases`
- Binding operator `cases >>$ fun r flow -> ...`

  `>>$ : ('a,'r) cases -> ('r -> 'a flow -> ('a,'s) cases) -> ('a,'s) cases`
- Side note: this is a monad

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`
- Whole abstract state of `type 'a`

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`
- Whole abstract state of `type 'a`

$\Big\}\Longrightarrow$ `type ('a, t) man`

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`
- Whole abstract state of `type 'a`

$\Longrightarrow$ `type ('a, t) man`

From global analysis to local domain

- Get the domain's data

```
get : 'a -> t
```

- Set the domain's data

```
set : t -> 'a -> 'a
```

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`
- Whole abstract state of `type 'a`

$\Longrightarrow$ `type ('a, t) man`

From global analysis to local domain

- Get the domain's data
  `get : 'a -> t`
- Set the domain's data
  `set : t -> 'a -> 'a`

From local domain to global analysis

- Analyze a given expression
- Analyze a given statement
  `man.exec stmt` $\sigma \Leftrightarrow \mathbb{S}^\sharp[\![\text{stmt}]\!]\sigma$

Signatures later

Abstract state & domain signature – II

The manager: interroperating the whole analysis and local domains

- Local domain has a private `type t`
- Whole abstract state of `type 'a`

$\Longrightarrow$ `type ('a, t) man`

From global analysis to local domain

- Get the domain's data

```
get : 'a -> t
```

- Set the domain's data

```
set : t -> 'a -> 'a
```

From local domain to global analysis

- Analyze a given expression
- Analyze a given statement

`man.exec stmt` $\sigma \Leftrightarrow \mathbb{S}^\sharp[\![\text{stmt}]\!]\sigma$

Signatures later

Also: lattice operators

Abstract state & domain signature – III

Utilities

```
type ('a, 'r) cases (* ≃ DNF of 'a flow * 'r *)

type 'a eval = ('a, expr) cases
type 'a post = ('a, unit) cases

(* Manager, allowing interaction between a
   domain ('t) and whole analysis ('a) *)
type ('a, 't) man = {
  get : 'a -> 't;
  set : 't -> 'a -> 'a;
  exec : stmt -> 'a flow -> 'a post;
  eval : expr -> 'a flow -> 'a eval;
  (* [...] *)
}
```

Domain type overview

```
module type DOMAIN = sig

type t
(* private, opaque data of the domain *)
val name : string

val join : t -> t -> t (* and other lattice operators *)

(* Transfer functions *)
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option

(* [...] *)
end
```

Focus on the domain-local transfer functions

```
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option
```

- `('a, t) man` manager

Focus on the domain-local transfer functions

```
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option
```

- `('a, t) man` manager
- `'a flow` abstract state

Focus on the domain-local transfer functions

```
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option
```

- `('a, t) man` manager
- `'a flow` abstract state
- option: domains return `None` for unsupported statements/expressions.

Focus on the domain-local transfer functions

```
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option
```

- `('a, t) man` manager
- `'a flow` abstract state
- option: domains return `None` for unsupported statements/expressions.
  - `'a post = ('a, unit) cases`. DNF of abstract states.

Focus on the domain-local transfer functions

```
val exec : stmt -> ('a, t) man -> 'a flow -> 'a post option
val eval : expr -> ('a, t) man -> 'a flow -> 'a eval option
```

- `('a, t) man` manager
- `'a flow` abstract state
- option: domains return `None` for unsupported statements/expressions.
  - `'a post = ('a, unit) cases`. DNF of abstract states.
  - `'a eval = ('a, expr) cases`. DNF of abstract states and symbolic expressions. Useful for rewriting, esp. for relational analyses

Example: loop iterator

Iterators are stateless domains:

- `type t = unit`, trivial lattice operators

Example: loop iterator

Iterators are stateless domains:

- `type t = unit`, trivial lattice operators
- We have an automatic lifter for all that!

Example: loop iterator

Iterators are stateless domains:

- `type t = unit`, trivial lattice operators
- We have an automatic lifter for all that!

The loop iterator focuses on postfixpoint computation, and delegates the rest.

Example: loop iterator

Iterators are stateless domains:

- `type t = unit`, trivial lattice operators
- We have an automatic lifter for all that!

The loop iterator focuses on postfixpoint computation, and delegates the rest.

Universal.Iterators.Loops

```
let rec lfp man cond body flow_init flow =
  man.exec (mk_block [mk_assume cond; body]) flow >>$ fun () flow' ->
  if man.lattice.subset (man.lattice.join flow_init flow') flow
  then Cases.singleton () flow'
  else lfp man cond body flow_init (man.lattice.widen flow flow')

let exec stmt man flow = match stmt.skind with
  | S_while (cond, body) ->
    Some (lfp man cond body flow flow >>$ fun () lfp_flow ->
          man.exec (mk_assume (mk_not cond)) lfp_flow)
  | _ -> None
```

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
...
```

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

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

What has been checked? What has not?

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

$$\texttt{if}\ a^\# \not\sqsubseteq p^\#\ \texttt{then}$$
$$\quad \texttt{add\_alarm}\ a^\#\ p^\#$$

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

$$\texttt{if}\ a^\# \not\sqsubseteq p^\#\ \texttt{then}\ \ \texttt{add\_alarm}\ a^\#\ p^\# \quad\rightsquigarrow\quad \texttt{if}\ a^\# \not\sqsubseteq p^\#\ \texttt{then}\ \ \texttt{add\_alarm}\ a^\#\ p^\#\ \ \texttt{else}\ \ \texttt{add\_safe\_check}\ p^\#$$

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

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

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

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

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

Mopsa's approach to being transparent – example

Mopsa's approach to being transparent

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

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

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

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

Mopsa’s approach to being transparent – output

Benefits of the approach

Mopsa's approach to being transparent – output

Benefits of the approach

- Easy to implement

Mopsa’s approach to being transparent – output

Benefits of the approach

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

Mopsa's approach to being transparent – output

Benefits of the approach

- Easy to implement
- "2,756 alarms" $\rightsquigarrow$ 87% checks proved correct – "selectivity"
- ~~Program size~~ $\rightsquigarrow$ "expression complexity"

Mopsa's approach to being transparent – output

Benefits of the approach

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

Analysis of coreutils `fmt`

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

1. Crash

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

1. Crash ✗

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

1. Crash ✗
2. Ignore silently

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

1. Crash ✗
2. Ignore silently ✗

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

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

Mopsa's approach to being transparent – soundness assumptions

Soundness assumptions, through an example

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

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

Related topic: soundiness paper [Liv+15]

Easing debugging

Developer-friendly interfaces

Where static analyzers usually start from

- Analysis output Too coarse

Where static analyzers usually start from

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

Where static analyzers usually start from

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

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

- Breakpoints
  - Program location

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

GDB-like interface to the abstract interpretation of the program

Demo!

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

An interactive engine acting as abstract debugger

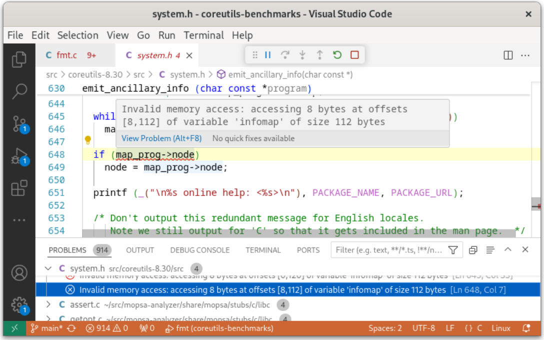
GDB-like interface to the abstract interpretation of the program

Demo!

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

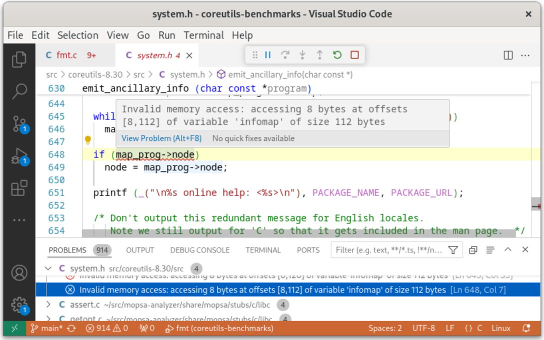
IDE support

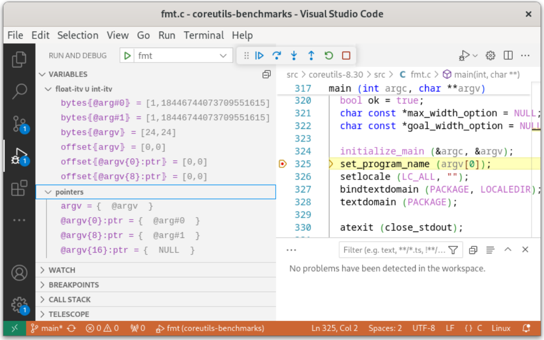
- Language Server Protocol for linters (report alarms)



IDE support

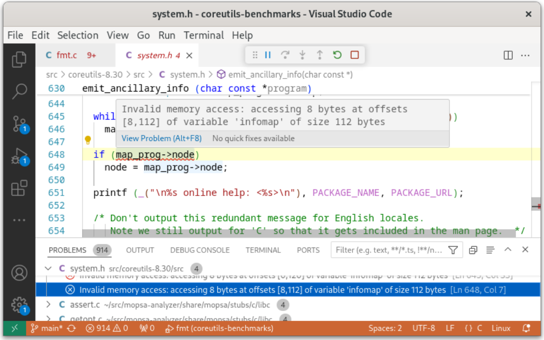
- Language Server Protocol for linters (report alarms)
- Debug Adapter Protocol providing interactive engine interface

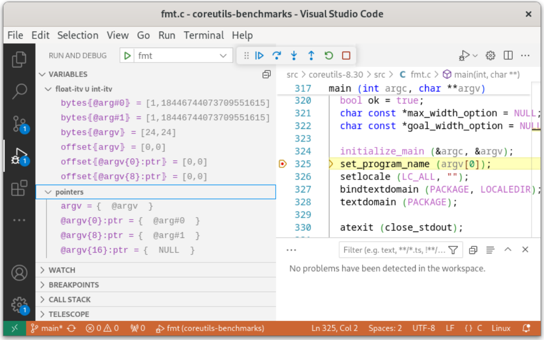




IDE support

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





Easing debugging

Testcase reduction

Testcase reduction

Motivation

Testcase reduction

Motivation

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

Testcase reduction

Motivation

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

Testcase reduction

Motivation

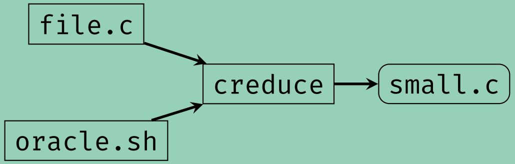
- Static analyzers are complex piece of code and may contain bugs
- In practice, some bugs will only be detected in large codebases
- Debugging extremely difficult: size of the program, analysis time

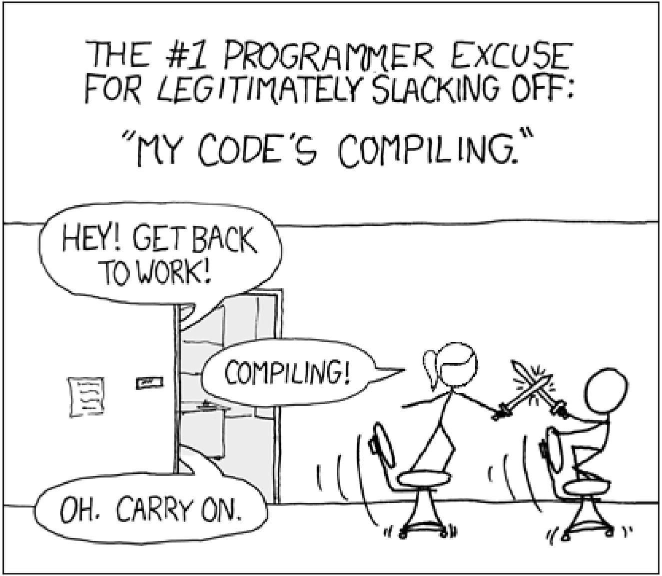
Testcase reduction

Motivation

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

Automated testcase reduction using `creduce` [Reg+12]




THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

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

$\rightsquigarrow$ `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

Conclusion

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:

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

Modular Open Platform for Static Analysis [Jou+19]

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

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

Conclusion

Modular Open Platform for Static Analysis [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

Conclusion

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

Conclusion

Modular Open Platform for Static Analysis [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

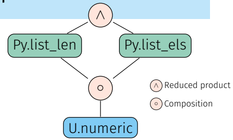
- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation



Conclusion

Modular Open Platform for Static Analysis [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

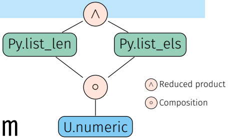
- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation



Mopsa Questions? Do Get in Touch!

`raphael.monat@inria.fr` `mopsa.zulipchat.com`

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.

[Bal+19] Clément Ballabriga et al. **"Static Analysis of Binary Code with Memory Indirections Using Polyhedra".** In: Lecture Notes in Computer Science. Springer, 2019, pp. 114–135.

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

References – II

[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.

[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`.

[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.