Multilanguage Static Analysis of Python/C Programs with Mopsa

Raphaël Monat – SyCoMoRES team

rmonat.fr

AWS

5 November 2025

Inria

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 in SyCoMoRES

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 in SyCoMoRES

- 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 in SyCoMoRES

- 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

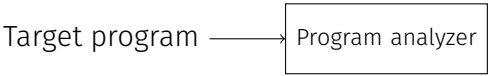
Automated verification of Catala programs

Other Research Interests in SyCoMoRES

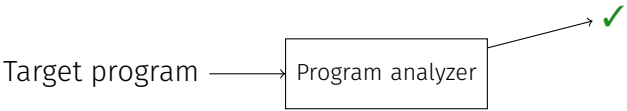
- Scheduling for real-time embedded systems
- Binary code analysis [Bal+19] (for worst-case execution time, security)
- Type systems for privacy

Target program

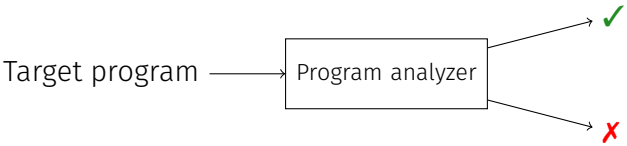
Automated Program Analysis



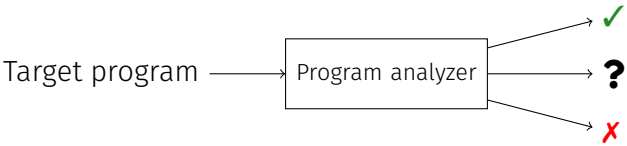
Automated Program Analysis



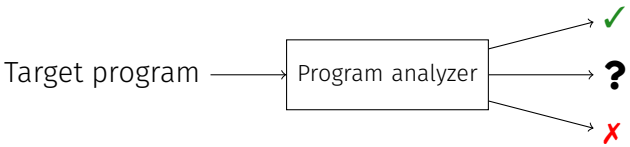
Automated Program Analysis



Automated Program Analysis



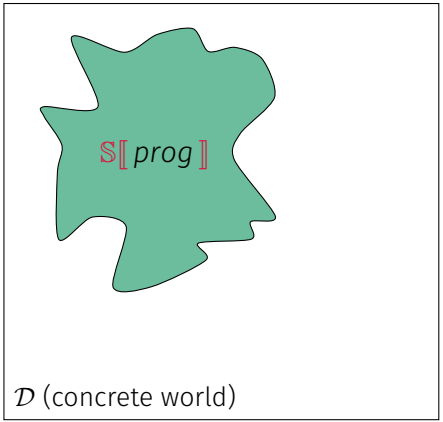
Automated Program Analysis

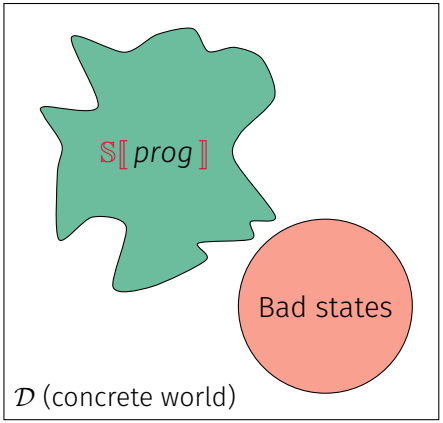


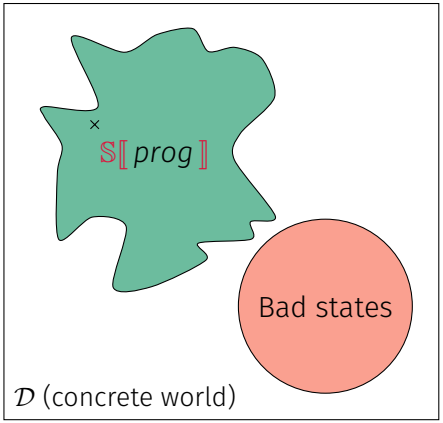
Motivation

Sheer quantity of programs and changes during their life:

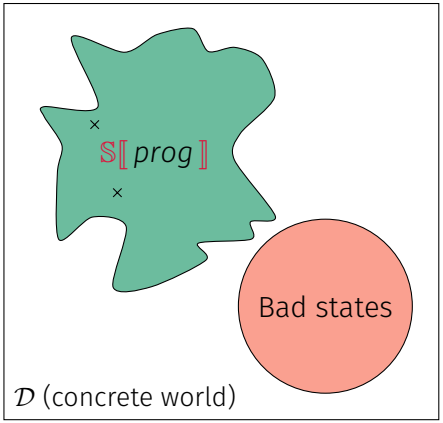
Manual processes (e.g. testing, manual verification) will not scale!


$\mathbb{S}[\![prog]\!]$
$\mathcal{D}$ (concrete world)

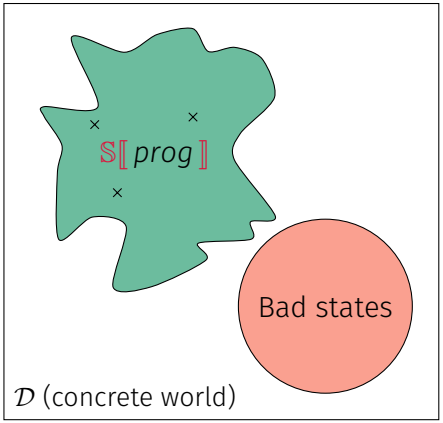

$\mathbb{S}[\![prog]\!]$
Bad states
$\mathcal{D}$ (concrete world)



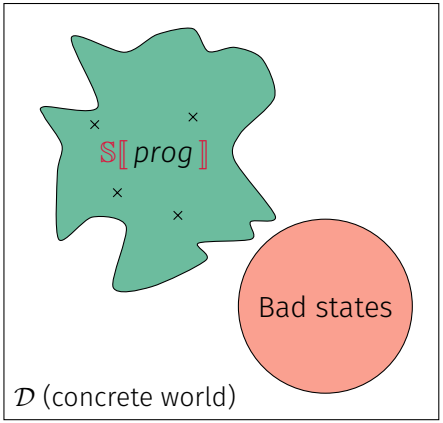
Tests



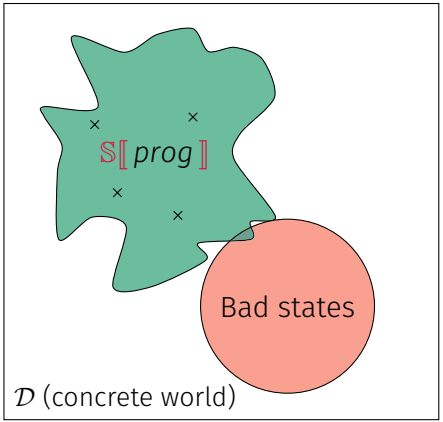
Tests



Tests

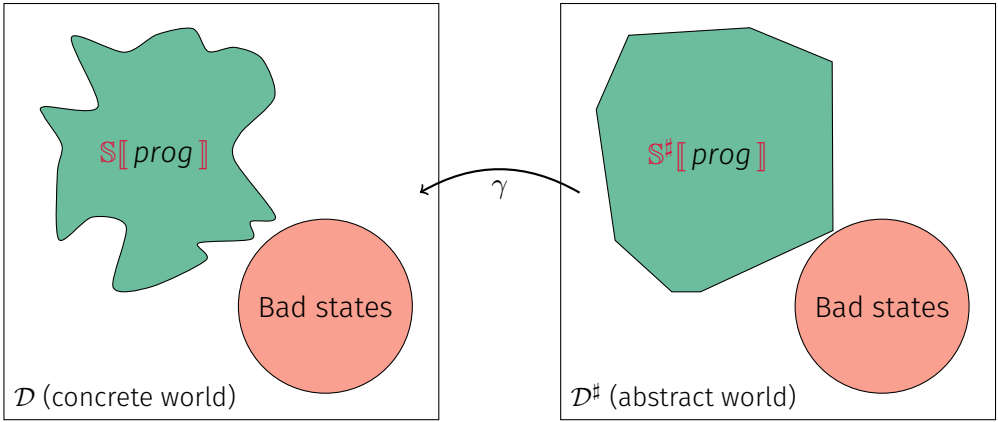


Tests

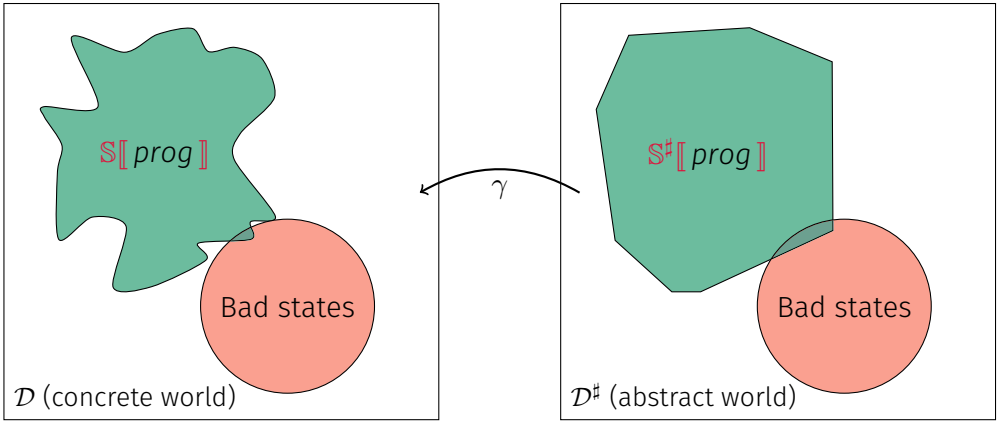


Tests are not sound

Abstract Interpretation for Software Safety

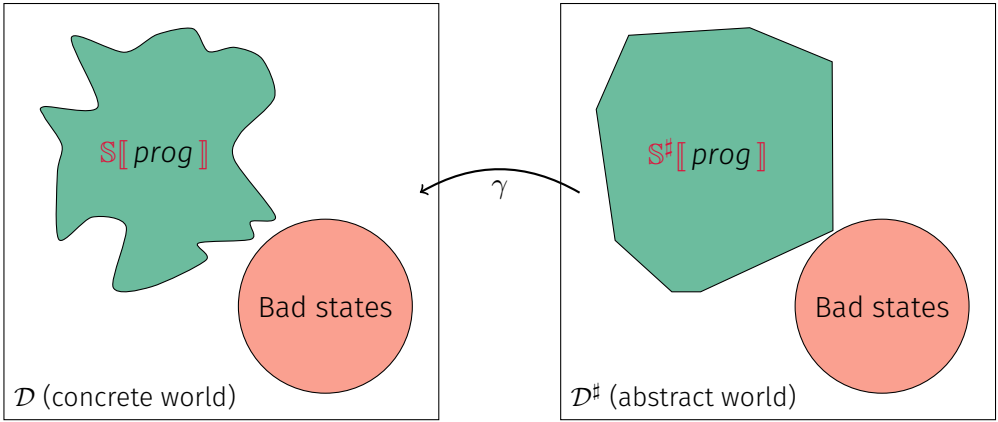


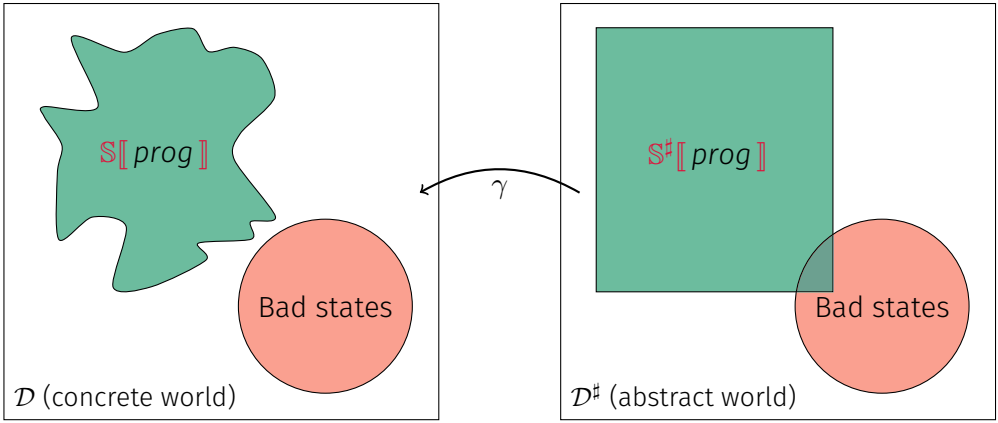
Safe program


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

True alarm

Abstract Interpretation for Software Safety





False alarm (due to imprecisions)

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

- From embedded C software to

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

- From embedded C software to
  - General C software (dynamic allocation, ...)

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

- From embedded C software to
  - General C software (dynamic allocation, ...)
  - Other languages

A Brief History of Abstract Interpretation

1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

- From embedded C software to
  - General C software (dynamic allocation, ...)
  - Other languages
- From full programs to libraries

A Brief History of Abstract Interpretation

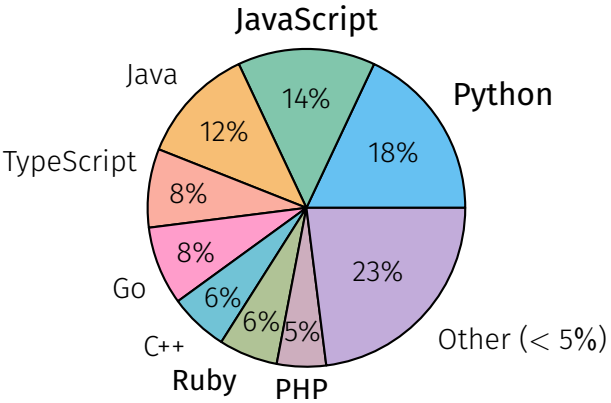
1977: foundational paper by Radhia and Patrick Cousot [CC77]

2010: critical software certification using Astrée [Ber+10]

Now: democratizing static analysis?

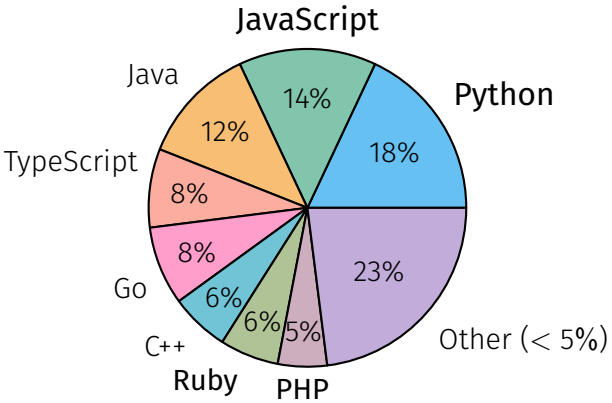
- From embedded C software to
  - General C software (dynamic allocation, ...)
  - Other languages
- From full programs to libraries
- Framework to implement analyses

Dynamic programming languages



Most popular languages on GitHub

Dynamic programming languages



Most popular languages on GitHub

New features

- Dynamic typing
- Dynamic object structure

Outline

1. A Taste of Python
2. A Modern Program Analyzer: Mopsa
3. Analyzing Python Programs
4. Analyzing Python Programs with C Libraries

A Taste of Python

Python's specificities

No standard

CPython is the reference

$\Longrightarrow$ manual inspection of the source code and handcrafted tests

Python's specificities

No standard

CPython is the reference

$\Longrightarrow$ manual inspection of the source code and handcrafted tests

Operator redefinition

- Calls, additions, attribute accesses
- Operators eventually call overloaded `__methods__`

Protected attributes

```
1  class Protected:
2    def __init__(self, priv):
3      self._priv = priv
4    def __getattribute__(self, attr):
5      if attr[0] == "_": raise AttributeError("...")
6      return object.__getattribute__(self, attr)
7
8  a = Protected(42)
9  a._priv # AttributeError raised
```

Python's specificities (II)

Dual type system

- Nominal (classes, MRO [Bar+96])

Fspath (from standard library)

```
1  class Path:
2    def __fspath__(self): return 42
3
4  def fspath(p):
5    if isinstance(p, (str, bytes)):
6      return p
7    elif hasattr(p, "__fspath__"):
8      r = p.__fspath__()
9      if isinstance(r, (str, bytes)):
10       return r
11   raise TypeError
12
13 fspath("/dev" if random() else Path())
```

Python's specificities (II)

Dual type system

- Nominal (classes, MRO [Bar+96])
- **Structural (attributes)**

Fspath (from standard library)

```
1  class Path:
2    def __fspath__(self): return 42
3
4  def fspath(p):
5    if isinstance(p, (str, bytes)):
6      return p
7    elif hasattr(p, "__fspath__"):
8      r = p.__fspath__()
9      if isinstance(r, (str, bytes)):
10       return r
11   raise TypeError
12
13 fspath("/dev" if random() else Path())
```

Python's specificities (II)

Dual type system

- Nominal (classes, MRO [Bar+96])
- Structural (attributes)

Exceptions

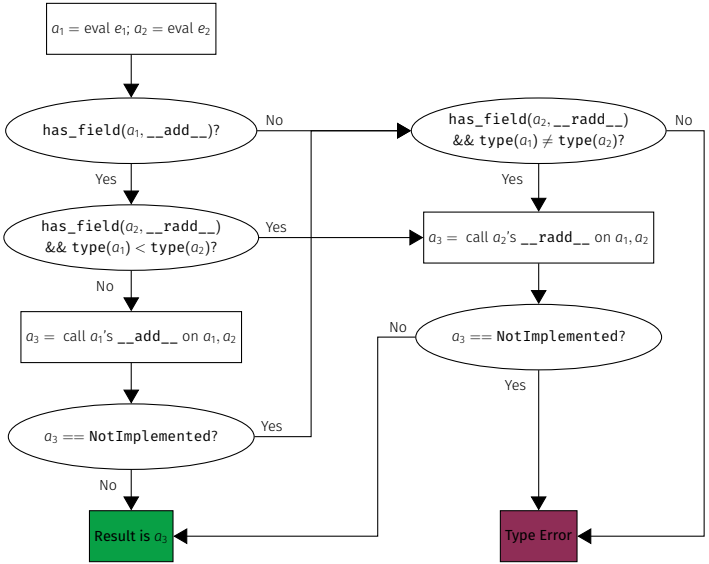
Exceptions rather than specific values

- `1 + "a"` ⇝ `TypeError`
- `l[len(l) + 1]` ⇝ `IndexError`

Fspath (from standard library)

```
1  class Path:
2    def __fspath__(self): return 42
3
4  def fspath(p):
5    if isinstance(p, (str, bytes)):
6      return p
7    elif hasattr(p, "__fspath__"):
8      r = p.__fspath__()
9      if isinstance(r, (str, bytes)):
10       return r
11   raise TypeError
12
13 fspath("/dev" if random() else Path())
```

Example Semantics – binary operators



Crazy Python

Custom infix operators

```
class Infix(object):
    def __init__(self, func): self.func = func
    def __or__(self, other): return self.func(other)
    def __ror__(self, other): return Infix(lambda x: self.func(other, x))

instanceof = Infix(isinstance)
b = 5 |instanceof| int

@Infix
def padd(x, y):
  print(f"{x} + {y} = {x + y}")
  return x + y
c = 2 |padd| 3
```

Credits `tomerfiliba.com/blog/Infix-Operators/`

A Modern Program Analyzer: Mopsa

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

`gitlab.com/mopsa/mopsa-analyzer`

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

Modular Open Platform for Static Analysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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`

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 Open Platform for Static Analysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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 Open Platform for Static Analysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer`

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
- T. Goalard

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
- T. Goalard

Maintainers in bold.

Works around Mopsa

Languages

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

Works around Mopsa

Languages

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

Multilanguage Python+C [MOM21]

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Properties

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Properties

- Absence of RTEs

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Properties

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

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Properties

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

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

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

Properties

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

Works around Mopsa

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b]
Multilanguage Python+C [MOM21]

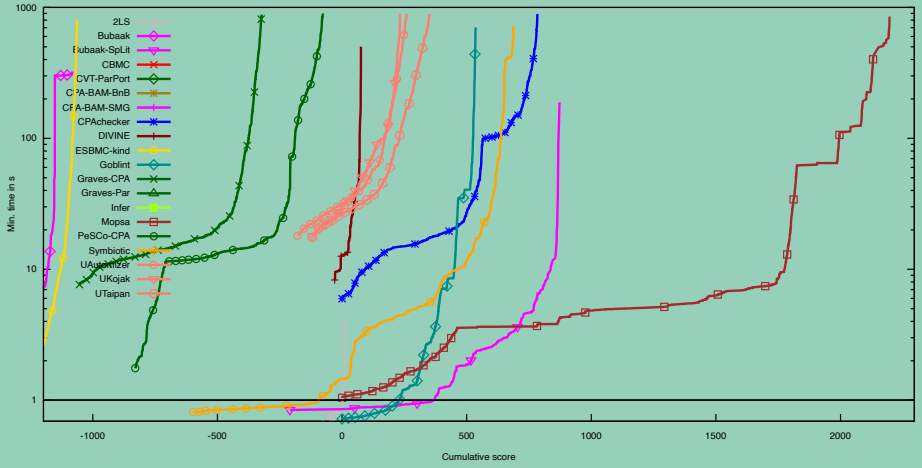
WIP: Michelson [Bau+22], OCaml [VMM23], Catala (date arithmetic [MFM24])...

Properties

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

Software Verification Competition

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



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

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

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

Analysis = composition of abstract domains

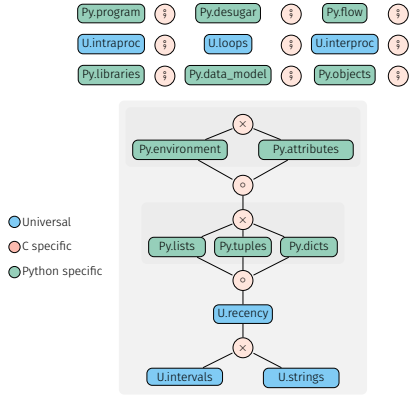
unified domain signature $\Longrightarrow$ iterators are abstract domains

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

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

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



Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

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

1

Iterators to handle multiple languages

Traditional approaches

Desugar/compile programs to an intermediate representation (IR)

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

Mopsa

1

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[1]

[1] 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[1]
- Various programming paradigms supported!

[1] 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[1]
- Various programming paradigms supported!
- All constructs have to be handled – but rewritings are possible

[1]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)

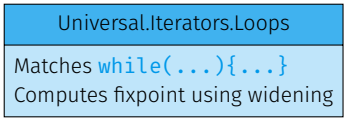
Example: Infer's IR has five (!) constructors

Mopsa

- No loss of precision from the frontend[1]
- 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

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

Dynamic, semantic iterators with delegation

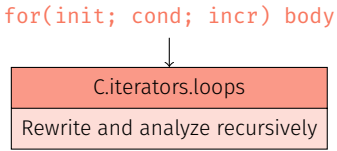


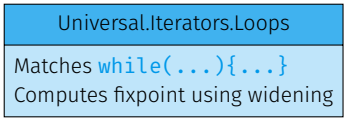
Dynamic, semantic iterators with delegation

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

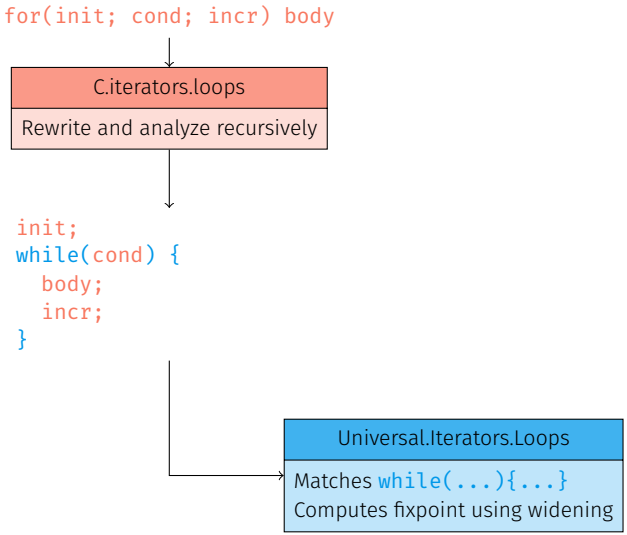
Universal.Iterators.Loops
Matches `while(...){...}` Computes fixpoint using widening

Dynamic, semantic iterators with delegation

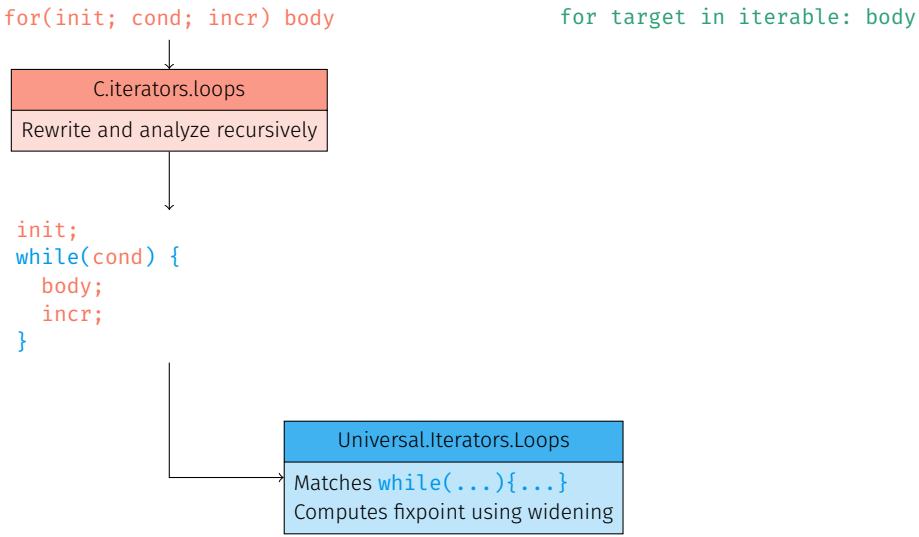




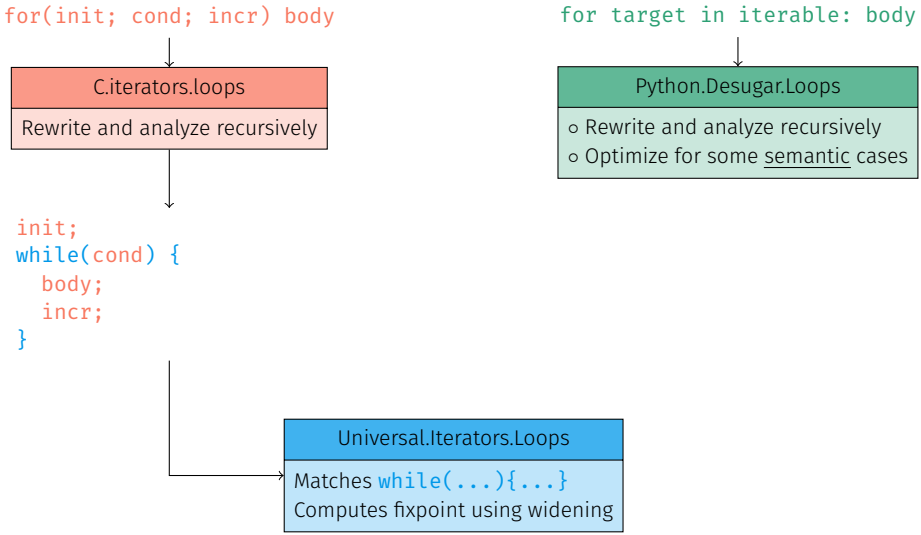
Dynamic, semantic iterators with delegation



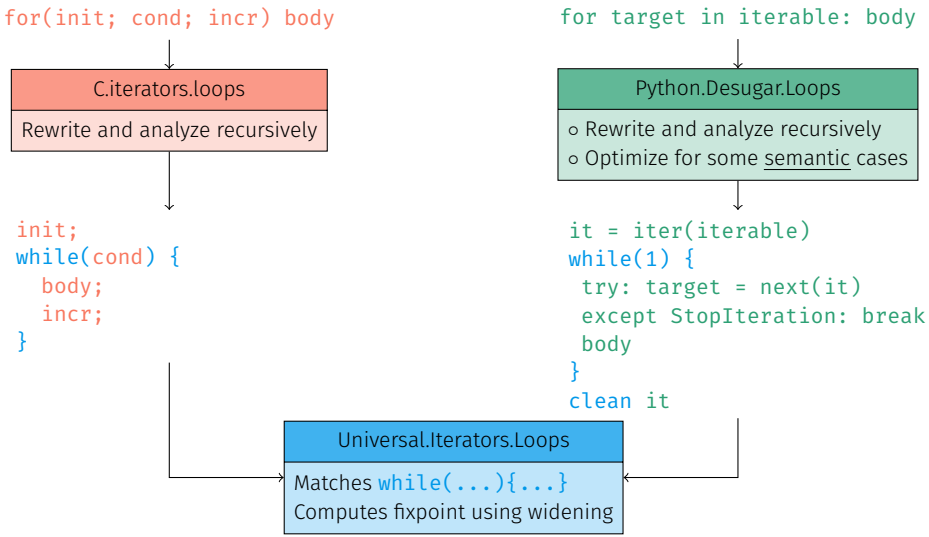
Dynamic, semantic iterators with delegation



Dynamic, semantic iterators with delegation



Dynamic, semantic iterators with delegation



Expression rewriting

$[\![m = m + l[i].weight]\!]^{\sharp}_{env} \sigma^{\sharp}$

Expression rewriting

$[\![m = m + l[i].weight]\!]^\sharp_{env}\sigma^\sharp$

$\hookrightarrow \mathbb{E}^\sharp_{binop}[\![m + l[i].weight]\!]\sigma^\sharp$

Expression rewriting

$$[\![m = m + l[i].weight]\!]^\sharp_{env}\sigma^\sharp$$

$$\hookrightarrow \mathbb{E}^\sharp_{binop}[\![m + l[i].weight]\!]\sigma^\sharp$$

$$\hookrightarrow \mathbb{E}^\sharp_{env}[\![m]\!]\sigma^\sharp$$

$$\longleftarrow \langle \mathbb{O}^\sharp_{\texttt{int}^\sharp}, \underline{\text{int}}(m) \rangle, \sigma^\sharp$$

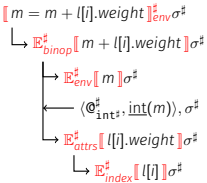
Expression rewriting

$[\![m = m + l[i].weight]\!]^\sharp_{env}\sigma^\sharp$

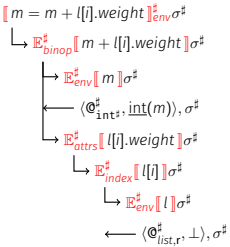
↳ $\mathbb{E}^\sharp_{binop}[\![m + l[i].weight]\!]\sigma^\sharp$

- → $\mathbb{E}^\sharp_{env}[\![m]\!]\sigma^\sharp$
- ← $\langle \mathfrak{C}^\sharp_{\texttt{int}^\sharp}, \underline{\text{int}}(m) \rangle, \sigma^\sharp$
- → $\mathbb{E}^\sharp_{attrs}[\![l[i].weight]\!]\sigma^\sharp$

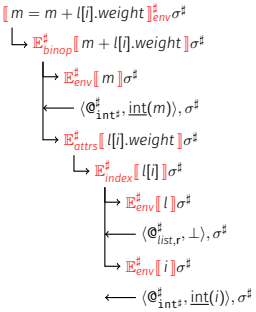
Expression rewriting



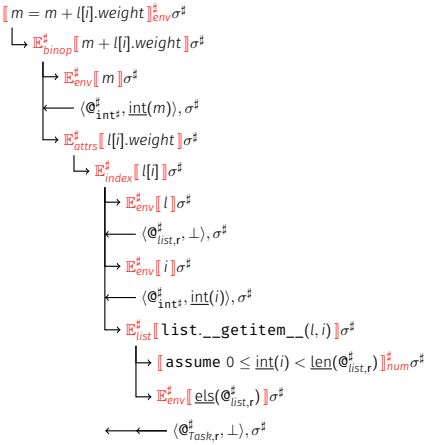
Expression rewriting



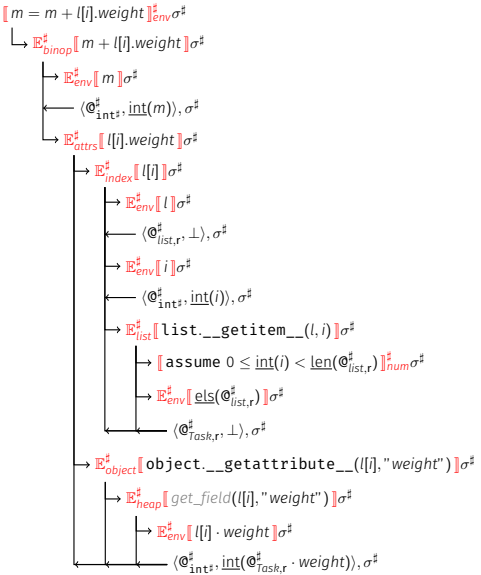
Expression rewriting



Expression rewriting



Expression rewriting



Analyzing Python Programs

Analysis Overview

Goal

Detect runtime errors: uncaught raised exceptions

Analysis Overview

Goal

Detect runtime errors: uncaught raised exceptions

Supported constructs

Our analysis supports:

- Objects
- Exceptions
- Dynamic typing
- Introspection
- Permissive semantics
- Dynamic attributes
- Generators
- `super`
- Metaclasses

Analysis Overview

Goal

Detect runtime errors: uncaught raised exceptions

Supported constructs

Our analysis supports:

- Objects
- Exceptions
- Dynamic typing
- Introspection
- Permissive semantics
- Dynamic attributes
- Generators
- `super`
- Metaclasses

Unsupported constructs

- Recursive functions
- `eval`
- Finalizers

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Searching for a loop invariant (l. 4)

“Nominal type” abstraction

$m : \texttt{int} \quad i : \texttt{int}$

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Searching for a loop invariant (l. 4)

Stateless domains: **list content**,

“Nominal type” abstraction
m : `int` i : `int` els(l) : `int`

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Searching for a loop invariant (l. 4)

Stateless domains: list content,

"Nominal type" abstraction

$m : \texttt{int} \quad i : \texttt{int} \quad \underline{\text{els}}(l) : \texttt{int}$

Numeric abstraction (intervals)

$m \in [0, +\infty) \qquad \underline{\text{els}}(l) \in [0, 20]$

$i \in [0, +\infty)$

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Searching for a loop invariant (l. 4)

Stateless domains: list content, **list length**

"Nominal type" abstraction

$m : \texttt{int} \quad i : \texttt{int} \quad \underline{\text{els}}(l) : \texttt{int}$

Numeric abstraction (intervals)

$m \in [0, +\infty) \qquad \underline{\text{els}}(l) \in [0, 20]$

$i \in \mathbf{[0, 10]} \qquad \underline{\mathbf{len}}(l) \in \mathbf{[5, 10]}$

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

"Nominal type" abstraction

$m : \texttt{int} \quad i : \texttt{int} \quad \underline{\text{els}}(l) : \texttt{int}$

Numeric abstraction (polyhedra)

$m \in [0, +\infty) \qquad \underline{\text{els}}(l) \in [0, 20]$

$0 \leq i < \underline{\text{len}}(l) \qquad 5 \leq \underline{\text{len}}(l) \leq 10$

Analysis Domains Required

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(ra
10 m = average(l)
```

Proved safe?

- `m // (i+1)`
- `m + l[i]`

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

“Nominal type” abstraction

$0 \leq i < \underline{\text{len}}(l) \qquad 5 \leq \underline{\text{len}}(l) \leq 10$

Conclusion

- Different domains depending on the precision
- Use of auxiliary variables (underlined)

Comparison of the type and value analyses [MOM20b]

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Type analysis

- `IndexError` (l. 4)

Comparison of the type and value analyses [MOM20b]

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Type analysis

- `IndexError` (l. 4)
- `ZeroDivisionError` (l. 5)

Comparison of the type and value analyses [MOM20b]

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Type analysis

- `IndexError` (l. 4)
- `ZeroDivisionError` (l. 5)
- `NameError` (l. 5)

Comparison of the type and value analyses [MOM20b]

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Type analysis

- `IndexError` (l. 4)
- `ZeroDivisionError` (l. 5)
- `NameError` (l. 5)

Non-relational value analysis

`IndexError` (l. 5)

Comparison of the type and value analyses [MOM20b]

Averaging numbers

```
1  def average(l):
2    m = 0
3    for i in range(len(l)):
4      m = m + l[i]
5    m = m // (i + 1)
6    return m
7
8  l = [randint(0, 20)
9    for i in range(randint(5, 10))]
10 m = average(l)
```

Type analysis

- `IndexError` (l. 4)
- `ZeroDivisionError` (l. 5)
- `NameError` (l. 5)

Non-relational value analysis

`IndexError` (l. 5)

Relational value analysis

No alarm!

Benchmarks

Name	LOC	Type Analysis					Non-relational Value Analysis				
		Time	Mem.	Exceptions detected			Time	Mem.	Exceptions detected		
				Type	Index	Key			Type	Index	Key
nbody.py	157	1.5s	3MB	0	22	1	5.7s	9MB	0	**1**	1
scimark.py	416	1.4s	12MB	1	1	0	3.4s	27MB	1	**0**	0
richards.py	426	13s	112MB	1	4	0	17s	149MB	1	**2**	0
unpack_seq.py	458	8.3s	7MB	0	0	0	9.4s	6MB	0	0	0
go.py	461	27s	345MB	33	20	0	2.0m	1.4GB	33	20	0
hexiom.py	674	1.1m	525MB	0	46	3	4.7m	3.2GB	0	**21**	3
regex_v8.py	1792	23s	18MB	0	2053	0	1.3m	56MB	0	**145**	0
processInput.py	1417	10s	64MB	7	7	1	12s	85MB	7	**4**	1
choose.py	2562	1.1m	1.6GB	12	22	7	2.9m	3.7GB	12	**13**	7
Total	9294	4.0m	2.8GB	59	2214	12	13m	9.1GB	59	228	12

Benchmarks

Name	LOC	Type Analysis					Non-relational Value Analysis				
		Time	Mem.	Exceptions detected Type	Index	Key	Time	Mem.	Exceptions detected Type	Index	Key
nbody.py	157	1.5								**1**	1
scimark.py	416	1.4								**0**	0
richards.py	426	13								**2**	0
unpack_seq.py	458	8.3								0	0
go.py	461	27								20	0
hexiom.py	674	1.1m								**21**	3
regex_v8.py	1792	23s								**145**	0
processInput.py	1417	10s								**4**	1
choose.py	2562	1.1m			22	7	2.9m	3.7GB	12	**13**	7
Total	9294	4.0m	2.8GB	59	2214	12	13m	9.1GB	59	228	12

Conclusion

The non-relational value analysis

- does not remove false type alarms
- significantly reduces index errors
- is $\simeq 3\times$ costlier

Benchmarks

Name	LOC	Type Analysis					Non-relational Value Analysis				
		Time	Mem.	Exceptions detected Type	Index	Key	Time	Mem.	Exceptions detected Type	Index	Key
nbody.py	157	1.5								1	1
scimark.py	416	1.4								0	0
richards.py	426	13								2	0
unpack_seq.py	458	8.3								0	0
go.py	461	27								20	0
hexiom.py	674	1.1m								21	3
regex_v8.py	1792	23s								145	0
processInput.py	1417	10s								4	1
choose.py	2562	1.1m			22	7	2.9m	3.7GB	12	13	7
Total	9294	4.0m	2.8GB	59	2214	12	13m	9.1GB	59	228	12

Conclusion

The non-relational value analysis

- does not remove false type alarms
- significantly reduces index errors
- is $\simeq 3\times$ costlier

Heuristic packing and relational analyses

- Static packing, using function's scope
- Rules out all 145 alarms of `regex_v8.py` (1792 LOC) at 2.5× cost

Selectivity of the non-relational value analysis

Name	Attributes	Types	Indexes	Keys	Values	Overflows	Divisions
scimark.py	746/746	844/844	**2**/5		**29**/30	**21**/43	**20**/21
richards.py	**352**/353	389/389	**2**/4		**2**/3		2/2
unpack_seq.py	807/807	1210/1210			1/1		
go.py	**664**/697	728/728	**2**/20		7/7	**6**/12	**4**/6
hexiom.py	598/598	672/672	**10**/32	**0**/3	**23**/24		
regex_v8.py	7357/7357	8349/8349	**1913**/2057		63/63		
processInput.py	**617**/619	**790**/792	12/12	**0**/1	**0**/1	2/2	
choose.py	**2519**/2521	**2997**/2999	**28**/39	**4**/8	**9**/24	**7**/17	

Selectivity of the analysis on some classes of exceptions

Selectivity = Number of proved safe operations / Total number of checks

An empty cell denotes a program where the kind of exception cannot happen

Analyzing Python Programs with C Libraries

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Pitfalls

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Pitfalls

- Different values (arbitrary-precision integers in Python, bounded in C)

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Pitfalls

- Different values (arbitrary-precision integers in Python, bounded in C)
- Different runtime-errors (exceptions in Python)

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Pitfalls

- Different values (arbitrary-precision integers in Python, bounded in C)
- Different runtime-errors (exceptions in Python)
- Garbage collection

Combining C and Python – motivation

One in five of the top 200 Python libraries contains C code

- To bring better performance (numpy)
- To provide library bindings (pygit2)

Pitfalls

- Different values (arbitrary-precision integers in Python, bounded in C)
- Different runtime-errors (exceptions in Python)
- Garbage collection
- Less approaches to detect multi-language attacks [MBO22]

Combining C and Python – example

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, "|i", &i))
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->count);
}
```

count.py

```python
from counter import Counter
from random import randrange

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

Combining C and Python – example

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, "|i", &i))
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->count);
}
```

count.py

```python
from counter import Counter
from random import randrange

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

- $\texttt{power} \leq 30 \Rightarrow \texttt{r} = 2^{\texttt{power}}$

Combining C and Python – example

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, "|i", &i))
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->count);
}
```

count.py

```python
from counter import Counter
from random import randrange

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

- $\texttt{power} \leq 30 \Rightarrow \texttt{r} = 2^{\texttt{power}}$
- $32 \leq \texttt{power} \leq 64$: OverflowError: signed integer is greater than maximum
- $\texttt{power} \geq 64$: OverflowError: Python int too large to convert to C long

Combining C and Python – example

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, "|i", &i))
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->count);
}
```

count.py

```python
from counter import Counter
from random import randrange

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

- $\texttt{power} \leq 30 \Rightarrow \texttt{r} = 2^{\texttt{power}}$
- $\texttt{power} = 31 \Rightarrow \texttt{r} = -2^{31}$
- $32 \leq \texttt{power} \leq 64$: OverflowError: signed integer is greater than maximum
- $\texttt{power} \geq 64$: OverflowError: Python int too large to convert to C long

How to analyze multilanguage programs?

Type annotations

```
class Counter:
  def __init__(self): ...
  def incr(self, i: int = 1): ...
  def get(self) -> int: ...
```

How to analyze multilanguage programs?

Type annotations

```
class Counter:
  def __init__(self): ...
  def incr(self, i: int = 1): ...
  def get(self) -> int: ...
```

- No raised exceptions $\Longrightarrow$ missed errors

How to analyze multilanguage programs?

Type annotations

```
class Counter:
  def __init__(self): ...
  def incr(self, i: int = 1): ...
  def get(self) -> int: ...
```

- No raised exceptions $\Longrightarrow$ missed errors
- Only types

How to analyze multilanguage programs?

Type annotations

```
class Counter:
  def __init__(self): ...
  def incr(self, i: int = 1): ...
  def get(self) -> int: ...
```

- No raised exceptions $\Longrightarrow$ missed errors
- Only types
- Typeshed: type annotations for the standard library

How to analyze multilanguage programs?

Type annotations

```
class Counter:
  def __init__(self): ...
  def incr(self, i: int = 1): ...
  def get(self) -> int: ...
```

- No raised exceptions $\implies$ missed errors
- Only types
- Typeshed: type annotations for the standard library, used in the single-language analysis before

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

```
class Counter:
  def __init__(self):
    self.count = 0
  def get(self):
    return self.count
  def incr(self, i=1):
    self.count += i
```

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

```
class Counter:
  def __init__(self):
    self.count = 0
  def get(self):
    return self.count
  def incr(self, i=1):
    self.count += i
```

- No integer wrap-around in Python

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

```
class Counter:
  def __init__(self):
    self.count = 0
  def get(self):
    return self.count
  def incr(self, i=1):
    self.count += i
```

- No integer wrap-around in Python
- Some effects can't be written in pure Python (e.g., read-only attributes)

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

Our approach

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

Our approach

- Analyze both the C and Python sources

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

Our approach

- Analyze both the C and Python sources
- Switch from one language to the other just as the program does

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

Our approach

- Analyze both the C and Python sources
- Switch from one language to the other just as the program does
- Reuse previous analyses of C and Python

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion
- Not sound: some effects are not taken into account

Our approach

- Analyze both the C and Python sources
- Switch from one language to the other just as the program does
- Reuse previous analyses of C and Python
- Detect runtime errors in Python, in C, and at the boundary

Analysis result

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, "|i", &i))
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->count);
}
```

count.py

```python
from counter import Counter
from random import randrange

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

Analysis result

counter.c

```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject*
CounterIncr(Counter *self,
{
    int i = 1;
    if(!PyArg_ParseTuple(a
       return NULL;

    self->count += i;
    Py_RETURN_NONE;
}

static PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("
}
```

count.py

```python
from counter import Counter
```

⚠Check #430:
./counter.c: In function '**CounterIncr**':
./counter.c:13.2-18: warning: Integer overflow

```
 13:    self->count += i;
        ^^^^^^^^^^^^^^^^
```

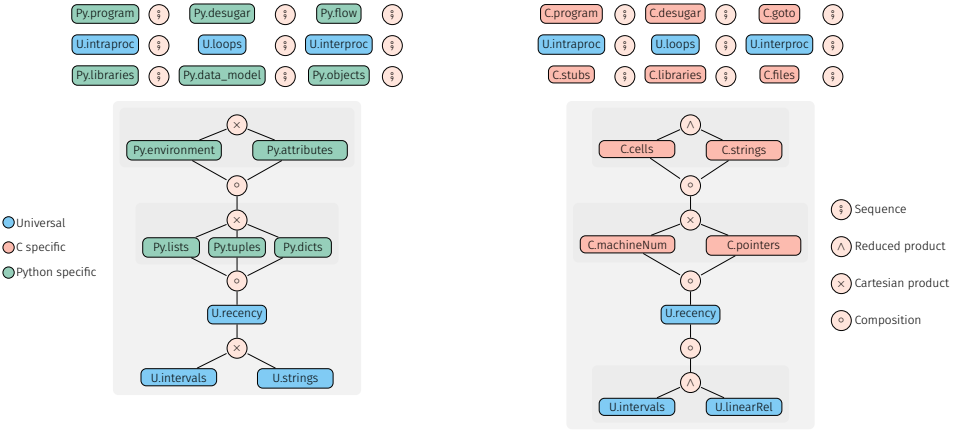
'**(self->count + i)**' has value [0,2147483648] that is larger than the range of '**signed int**' = [-2147483648,2147483647]
Callstack:
    from count.py:8.0-8: CounterIncr

✗Check #506:
count.py: In function '**PyErr_SetString**':
count.py:6.0-14: error: OverflowError exception

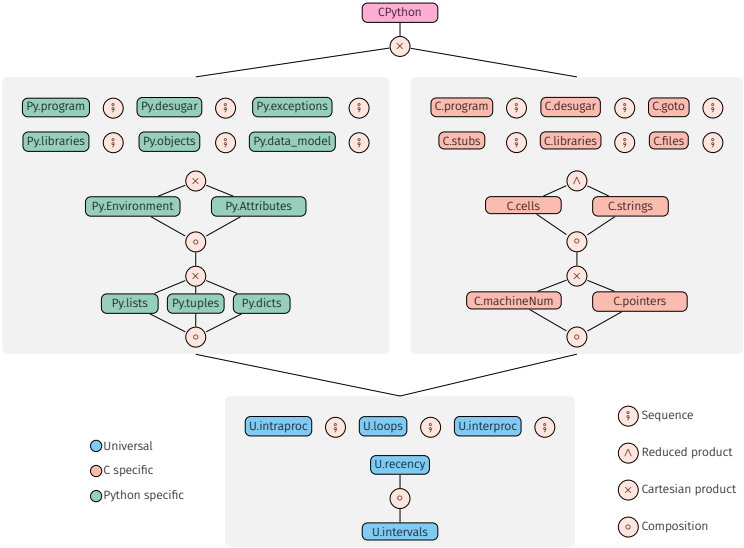
```
 6: c.incr(2**p-1)
    ^^^^^^^^^^^^^^
```

Uncaught Python exception: OverflowError: signed integer is greater than maximum
Uncaught Python exception: OverflowError: Python int too large to convert to C long
Callstack:
    from ./counter.c:17.6-38::convert_single[0]: PyParseTuple_int
    from count.py:7.0-14: CounterIncr
+1 other callstack

From distinct Python and C analyses...



From distinct Python and C analyses… to a multilanguage analysis!



Benchmarks

Corpus selection

- Popular, real-world libraries available on GitHub, averaging 412 stars.
- Whole-program analysis: we use the tests provided by the libraries.

Library	C + Py. Loc	Tests	/test	# proved checks / # 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)

Conclusion

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

`gitlab.com/mopsa/mopsa-analyzer`

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

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

`gitlab.com/mopsa/mopsa-analyzer`

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

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

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

`gitlab.com/mopsa/mopsa-analyzer`

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

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

Conclusion

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

`gitlab.com/mopsa/mopsa-analyzer`

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

One AST to rule them all

- Multilanguage support
- Expressiveness
- Reusability

Unified domain signature

- Semantic rewriting
- Loose coupling
- Observability

DAG of abstractions

- Relational domains
- Composition
- Cooperation

Multilanguage Static Analysis of Python/C Programs with Mopsa

Raphaël Monat – SyCoMoRES team

rmonat.fr

AWS

5 November 2025

Inria

References – I

[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.

[Bar+96] Kim Barrett et al. **“A Monotonic Superclass Linearization for Dylan”.** In: 1996.

[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.

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

References – II

[CC77] P. Cousot and R. Cousot. **“Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”.** In: ACM, 1977, pp. 238–252.

[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.

References – III

[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.

[MBO22] Samuel Mergendahl, Nathan Burow, and Hamed Okhravi. **“Cross-Language Attacks”.** In: The Internet Society, 2022.

References – IV

[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.

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

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

References – V

[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.

[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.

References – VI

[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 – VII

[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.