

# A Multilanguage Static Analysis of Python/C Programs with Mopsa

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné

SRG, Imperial College, London

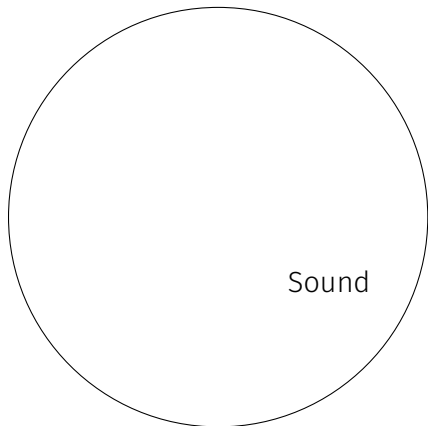
17 March 2022

rmonat.fr



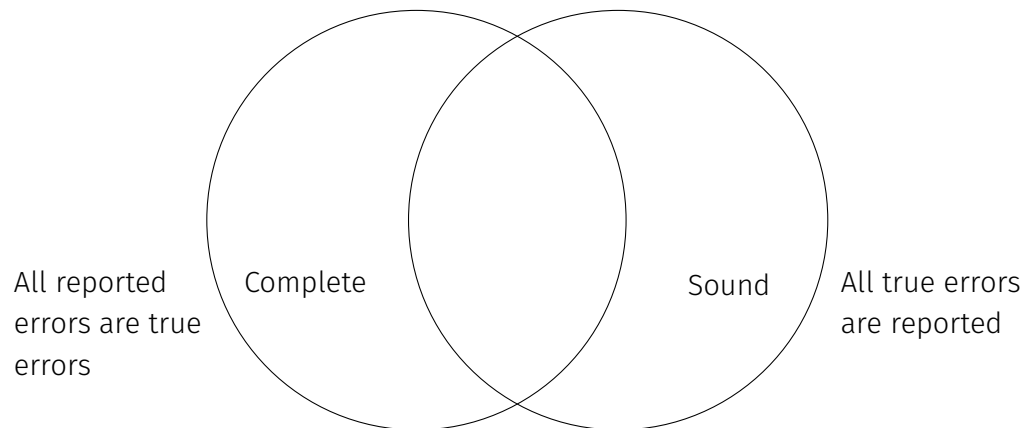
# Introduction

---

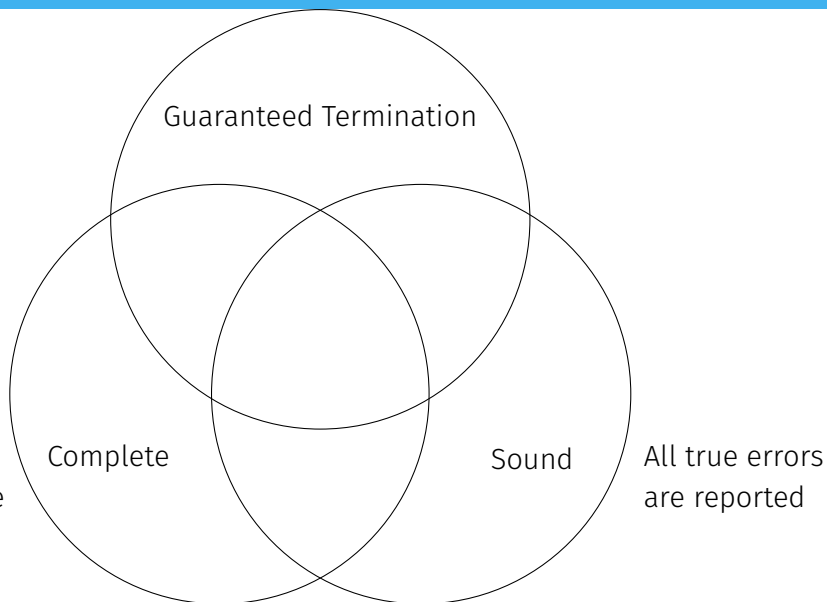


Sound

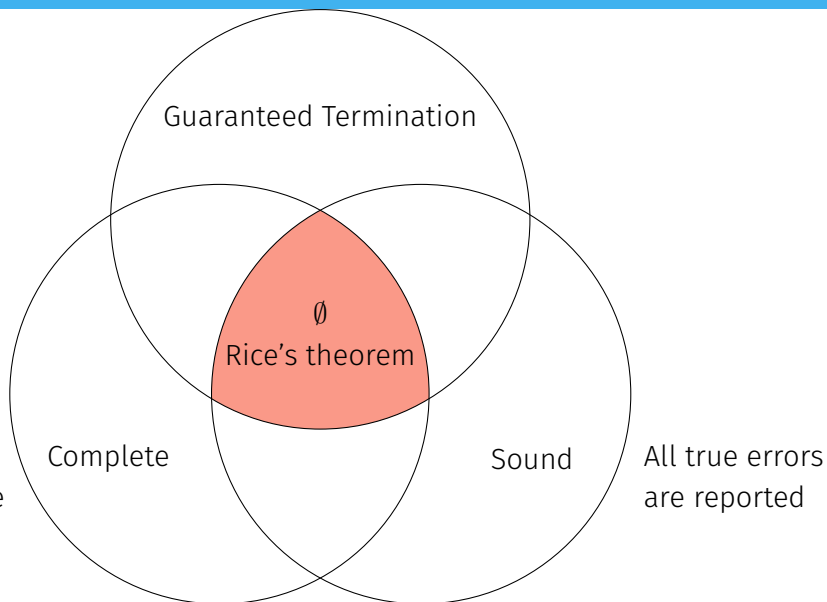
All true errors  
are reported



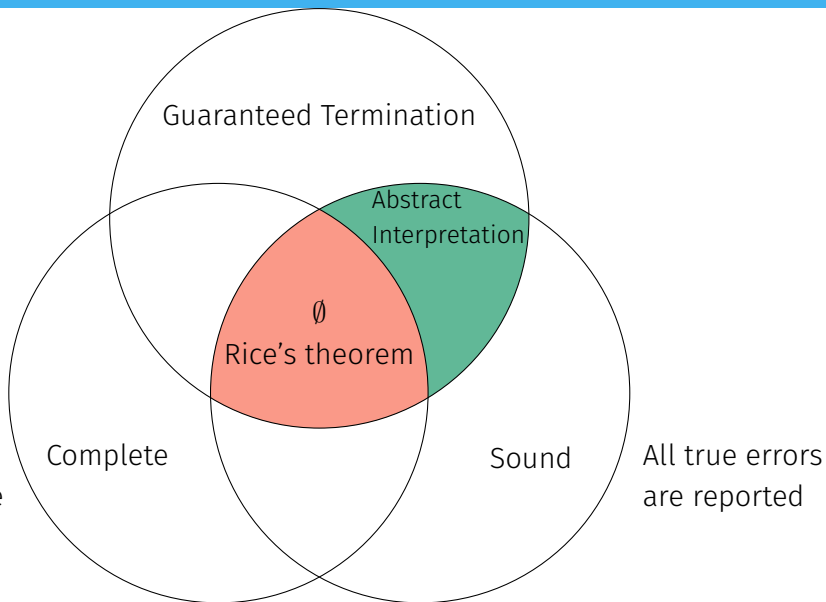
# Program verification



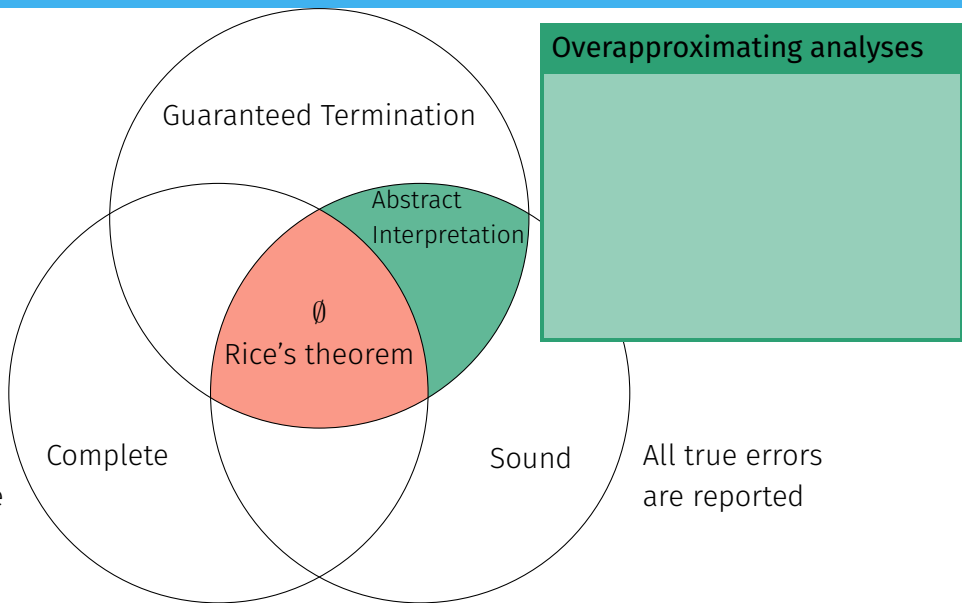
# Program verification



# Program verification

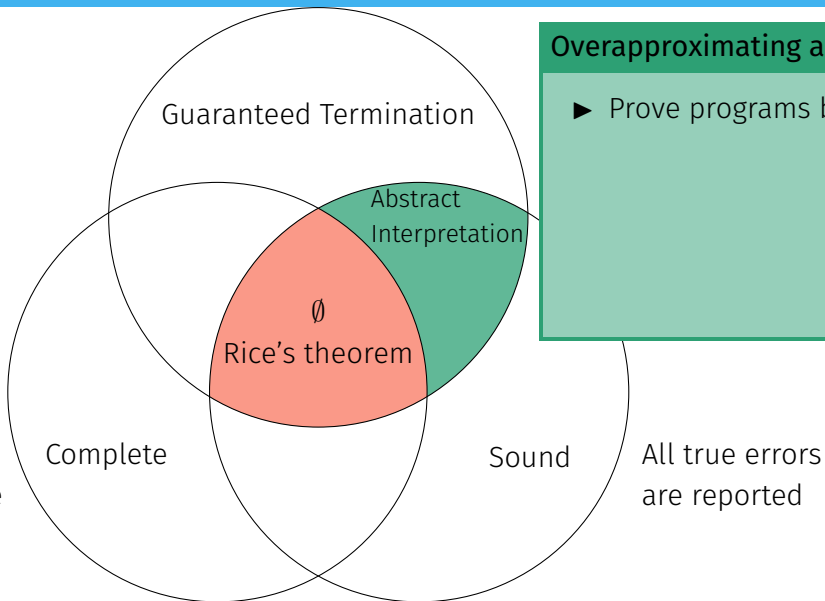


# Program verification

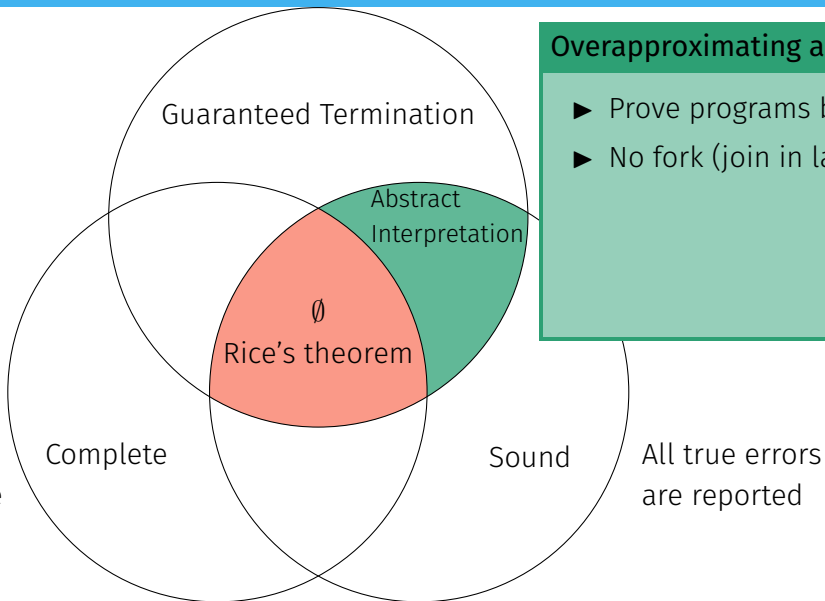




# Program verification

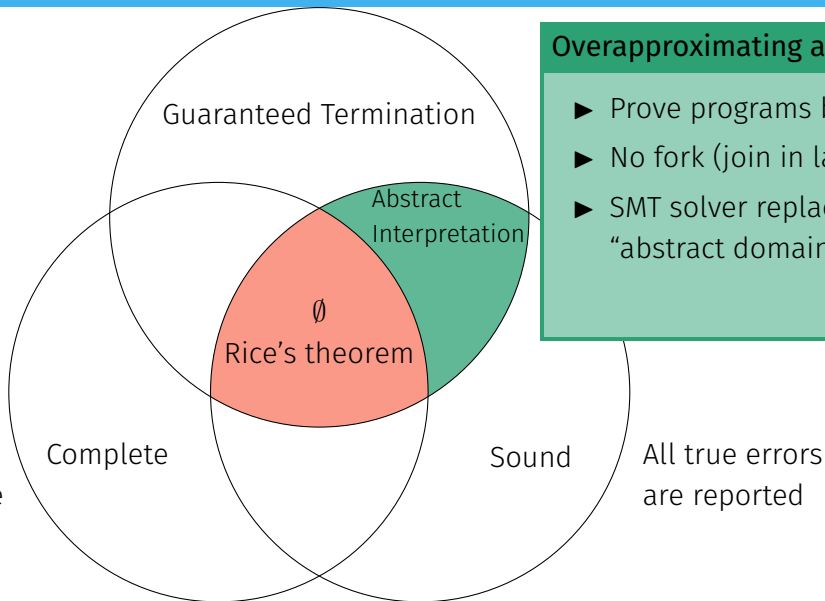


# Program verification



## Overapproximating analyses

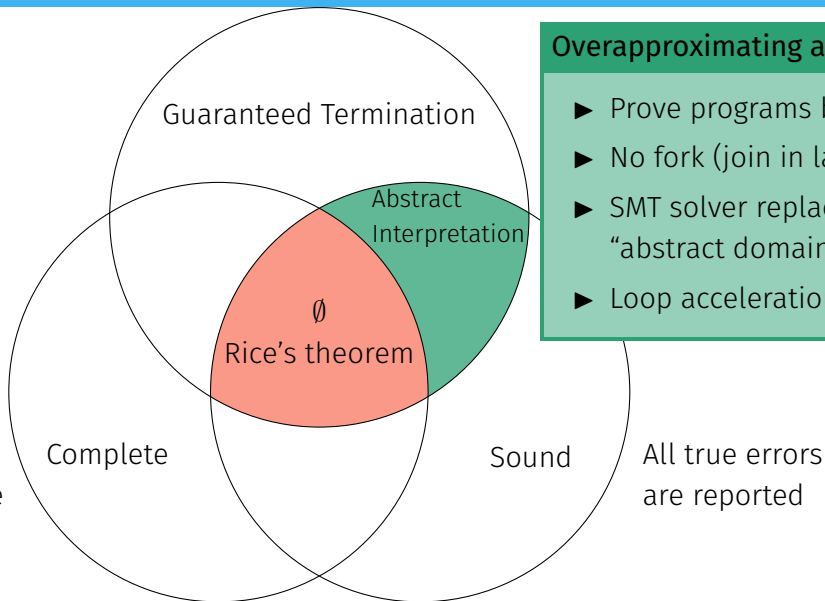
- ▶ Prove programs bug-free
- ▶ No fork (join in lattices)



## Overapproximating analyses

- ▶ Prove programs bug-free
- ▶ No fork (join in lattices)
- ▶ SMT solver replaced by “abstract domains”

# Program verification



## Overapproximating analyses

- ▶ Prove programs bug-free
- ▶ No fork (join in lattices)
- ▶ SMT solver replaced by “abstract domains”
- ▶ Loop acceleration

# Conservative static program analysis

average.py

```
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 s
7
8 r1 = average([1, 2, 3])
9 r2 = average(['a', 'b', 'c'])
```

TypeError: unsupported operand type(s) for '+': 'int' and 'str'

argslen.c

```
1 #include <string.h>
2
3 int main(int argc, char *argv[]) {
4     int i = 0;
5     for (char **p = argv; *p; p++) {
6         strlen(*p); // valid string
7         i++; // no overflow
8     }
9     return 0;
10 }
```

No alarm

## Specifications of the analyzer

**Inference** of program properties such as the absence of run-time errors.

**Semantic** based on a formal modelization of the language.

**Automatic** no expert knowledge required.

**Sound** cover all possible executions.

## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

---

<sup>1</sup><https://octoverse.github.com/#top-languages>

## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

## New features

- ▶ Object orientation,

---

<sup>1</sup><https://octoverse.github.com/#top-languages>

## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

## New features

- ▶ Object orientation,
- ▶ Dynamic typing,

---

<sup>1</sup><https://octoverse.github.com/#top-languages>



## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

## New features

- ▶ Object orientation,
- ▶ Dynamic typing,
- ▶ Dynamic object structure,

---

<sup>1</sup><https://octoverse.github.com/#top-languages>

## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

## New features

- ▶ Object orientation,
- ▶ Dynamic typing,
- ▶ Dynamic object structure,
- ▶ Introspection operators,

---

<sup>1</sup><https://octoverse.github.com/#top-languages>

## Growing popularity

JavaScript #1, Python #2 on GitHub<sup>1</sup>

## New features

- ▶ Object orientation,
- ▶ Dynamic typing,
- ▶ Dynamic object structure,
- ▶ Introspection operators,
- ▶ `eval`.

---

<sup>1</sup><https://octoverse.github.com/#top-languages>

## No standard

- ▶ CPython is the reference

  - ⇒ manual inspection of the source code and handcrafted tests

## No standard

- ▶ CPython is the reference
  - ⇒ 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 __getattr__(self, attr):
5         if attr[0] == "_": raise AttributeError("...")
6         return object.__getattr__(self, attr)
7
8 a = Protected(42)
9 a._priv # AttributeError raised
```

### Dual type system

- ▶ Nominal (classes, MRO)

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)
- ▶ 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())
```

---

Barrett et al. "A Monotonic Superclass Linearization for Dylan". OOPSLA 1996

# Python's specificities (II)

## Dual type system

- ▶ Nominal (classes, MRO)
- ▶ 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())
```

---

Barrett et al. "A Monotonic Superclass Linearization for Dylan". OOPSLA 1996



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

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

- ▶ To bring better performance (numpy)

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

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

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

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

Pitfalls

## 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)

## 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)

## 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

# Outline

- 1 Introduction
- 2 Mopsa
- 3 Concrete Multilanguage Semantics
- 4 Implementation & Experimental Evaluation
- 5 Conclusion



Mopsa

---

# A program analysis workflow

## 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)
```

# A program analysis workflow

## 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?

- ▶  $m // (i+1)$
- ▶  $l[i]$

Searching for a loop invariant (l. 4)

## Environment abstraction

$$m \mapsto @_{\text{int}}^{\#} \quad i \mapsto @_{\text{int}}^{\#}$$

# A program analysis workflow

## 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?

- ▶  $m // (i+1)$
- ▶  $l[i]$

Searching for a loop invariant (l. 4)

Stateless domains: **list content**,

## Environment abstraction

$$m \mapsto @_{\text{int}}^{\#} \quad i \mapsto @_{\text{int}}^{\#} \quad \underline{\text{els}}(l) \mapsto @_{\text{int}}^{\#}$$

# A program analysis workflow

## 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?

- ▶  $m // (i+1)$
- ▶  $l[i]$

Searching for a loop invariant (l. 4)

Stateless domains: list content,

## Environment abstraction

$$m \mapsto @_{\text{int}}^{\#} \quad i \mapsto @_{\text{int}}^{\#} \quad \underline{\text{els}}(l) \mapsto @_{\text{int}}^{\#}$$

## Numeric abstraction (intervals)

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

# A program analysis workflow

## 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)`
- ▶ `l[i]`

Searching for a loop invariant (l. 4)

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

## Environment abstraction

$$m \mapsto @_{\text{int}}^{\#} \quad i \mapsto @_{\text{int}}^{\#} \quad \underline{\text{els}}(l) \mapsto @_{\text{int}}^{\#}$$

## Numeric abstraction (intervals)

$$m \in [0, +\infty) \quad \underline{\text{els}}(l) \in [0, 20]$$
$$\underline{\text{len}}(l) \in [5, 10] \quad i \in [0, 10]$$

# A program analysis workflow

## 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)`
- ▶ `l[i]`

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

## Environment abstraction

$$m \mapsto @_{\text{int}}^{\#} \quad i \mapsto @_{\text{int}}^{\#} \quad \underline{\text{els}}(l) \mapsto @_{\text{int}}^{\#}$$

## Numeric abstraction (polyhedra)

$$m \in [0, +\infty) \quad \underline{\text{els}}(l) \in [0, 20]$$
$$0 \leq i < \underline{\text{len}}(l) \quad 5 \leq \underline{\text{len}}(l) \leq 10$$

# A program analysis workflow

## Averaging tasks

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6     def average(l):
7         m = 0
8         for i in range(len(l)):
9             m = m + l[i].weight
10            m = m // (i + 1)
11        return m
12
13 l = [Task(randint(0, 20))
14      for i in range(randint(5, 10))]
15 m = average(l)
```

Proved safe?

- ▶  $m // (i+1)$
- ▶  $l[i].weight$

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

## Environment abstraction

$$\begin{aligned} m &\mapsto @_{int}^\# & i &\mapsto @_{int}^\# & \underline{\text{els}(l)} &\mapsto @_{Task}^\# \\ @_{Task}^\# \cdot \text{weight} &\mapsto @_{int}^\# \end{aligned}$$

## Numeric abstraction (polyhedra)

$$\begin{aligned} m &\in [0, +\infty) \\ 0 &\leq i < \underline{\text{len}(l)} & 5 &\leq \underline{\text{len}(l)} \leq 10 \\ 0 &\leq @_{Task}^\# \cdot \text{weight} \leq 20 \end{aligned}$$

## Attributes abstraction

$$@_{Task}^\# \mapsto (\{\text{weight}\}, \emptyset)$$



# A program analysis workflow

## Averaging tasks

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 def average(l):
7     m = 0
8     for i in range(len(l)):
9         m = m + l[i].weight
10        m = m // (i + 1)
11    return m
12
13 l = [Task(randint(0, 10), randint(1, 10))
14      for i in range(randint(5, 10))]
15 m = average(l)
```

## Conclusion

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

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

## Environment abstraction

$m \mapsto @^{\#} \dots i \dots @^{\#} \dots @^{\#} \text{Task}$

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

$0 \leq \underline{@^{\#}_{\text{Task}} \cdot \text{weight}} \leq 20$

## Attributes abstraction

$@^{\#}_{\text{Task}} \mapsto (\{\text{weight}\}, \emptyset)$

Proved safe?

- ▶  $m // (i+1)$
- ▶  $l[i].weight$



Modular Open Platform for Static Analysis<sup>2</sup>

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

---

<sup>2</sup>Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.



## Modular Open Platform for Static Analysis<sup>2</sup>

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

- ▶ One AST to analyze them all
  - 🚩 Multilanguage support
  - 📄 Expressiveness
  - ♻️ Reusability

---

<sup>2</sup>Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.



## Modular Open Platform for Static Analysis<sup>2</sup>

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

▶ One AST to analyze them all



Multilanguage support



Expressiveness



Reusability

▶ Unified domain signature



Semantic rewriting



Loose coupling



Observability

---

<sup>2</sup>Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.



## Modular Open Platform for Static Analysis<sup>2</sup>

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

▶ One AST to analyze them all



Multilanguage support



Expressiveness



Reusability

▶ Unified domain signature



Semantic rewriting



Loose coupling



Observability

▶ DAG of abstract domains



Composition



Cooperation

---

<sup>2</sup>Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.



## Modular Open Platform for Static Analysis<sup>2</sup>

[gitlab.com/mopsa/mopsa-analyzer](https://gitlab.com/mopsa/mopsa-analyzer)

▶ One AST to analyze them all

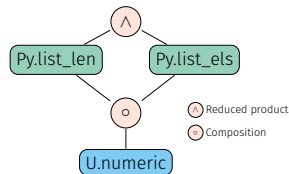
- ▶ Multilanguage support
- ▶ Expressiveness
- ▶ Reusability

▶ Unified domain signature

- ▶ Semantic rewriting
- ▶ Loose coupling
- ▶ Observability

▶ DAG of abstract domains

- ▶ Composition
- ▶ Cooperation



<sup>2</sup>Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.

# 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

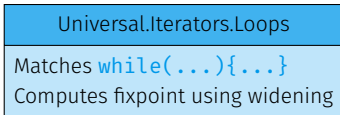
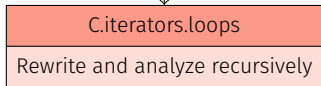
Matches `while(...){...}`

Computes fixpoint using widening



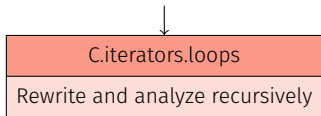
# Dynamic, semantic iterators with delegation

`for(init; cond; incr) body`

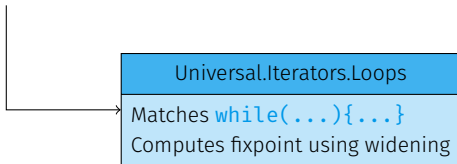


# Dynamic, semantic iterators with delegation

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



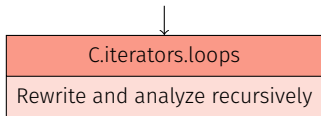
```
init;  
while(cond) {  
    body;  
    incr;  
}  
clean init
```



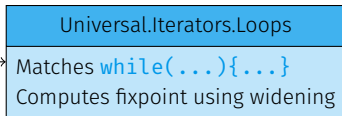
# Dynamic, semantic iterators with delegation

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

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

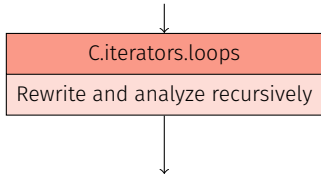


```
init;  
while(cond) {  
    body;  
    incr;  
}  
clean init
```

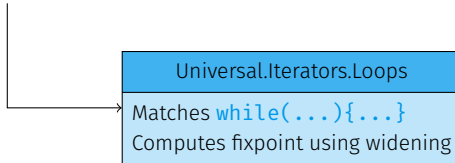


# Dynamic, semantic iterators with delegation

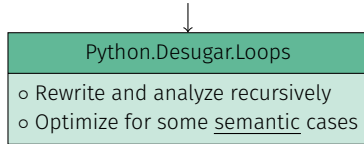
```
for(init; cond; incr) body
```



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

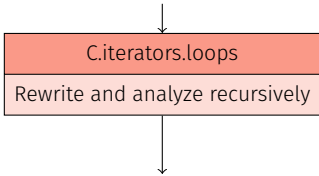


```
for target in iterable: body
```



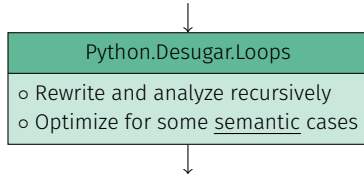
# Dynamic, semantic iterators with delegation

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

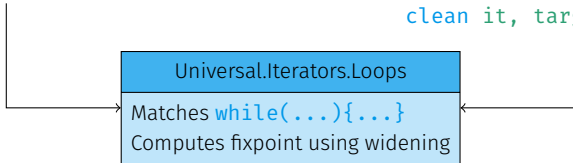


```
init;  
while(cond) {  
    body;  
    incr;  
}  
clean init
```

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



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



# Concrete Multilanguage Semantics

---

## Combining C and Python – example

counter.c

```
1 typedef struct {
2     PyObject_HEAD;
3     int count;
4 } Counter;
5
6 static PyObject*
7 CounterIncr(Counter *self, PyObject *args)
8 {
9     int i = 1;
10    if(!PyArg_ParseTuple(args, "|i", &i))
11        return NULL;
12
13    self->count += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
19 {
20     return Py_BuildValue("i", self->count);
21 }
```

count.py

```
1 from counter import Counter
2 from random import randrange
3
4 c = Counter()
5 power = randrange(128)
6 c.incr(2**power-1)
7 c.incr()
8 r = c.get()
```

## Combining C and Python – example

counter.c

```
1 typedef struct {
2     PyObject_HEAD;
3     int count;
4 } Counter;
5
6 static PyObject*
7 CounterIncr(Counter *self, PyObject *args)
8 {
9     int i = 1;
10    if(!PyArg_ParseTuple(args, "|i", &i))
11        return NULL;
12
13    self->count += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
19 {
20     return Py_BuildValue("i", self->count);
21 }
```

count.py

```
1 from counter import Counter
2 from random import randrange
3
4 c = Counter()
5 power = randrange(128)
6 c.incr(2**power-1)
7 c.incr()
8 r = c.get()
```

▶  $\text{power} \leq 30 \Rightarrow r = 2^{\text{power}}$



# Combining C and Python – example

counter.c

```
1 typedef struct {
2     PyObject_HEAD;
3     int count;
4 } Counter;
5
6 static PyObject*
7 CounterIncr(Counter *self, PyObject *args)
8 {
9     int i = 1;
10    if(!PyArg_ParseTuple(args, "|i", &i))
11        return NULL;
12
13    self->count += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
19 {
20     return Py_BuildValue("i", self->count);
21 }
```

count.py

```
1 from counter import Counter
2 from random import randrange
3
4 c = Counter()
5 power = randrange(128)
6 c.incr(2**power-1)
7 c.incr()
8 r = c.get()
```

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

# Combining C and Python – example

counter.c

```
1 typedef struct {
2     PyObject_HEAD;
3     int count;
4 } Counter;
5
6 static PyObject*
7 CounterIncr(Counter *self, PyObject *args)
8 {
9     int i = 1;
10    if(!PyArg_ParseTuple(args, "|i", &i))
11        return NULL;
12
13    self->count += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
19 {
20     return Py_BuildValue("i", self->count);
21 }
```

count.py

```
1 from counter import Counter
2 from random import randrange
3
4 c = Counter()
5 power = randrange(128)
6 c.incr(2**power-1)
7 c.incr()
8 r = c.get()
```

- ▶  $\text{power} \leq 30 \Rightarrow r = 2^{\text{power}}$
- ▶  $\text{power} = 31 \Rightarrow r = -2^{31}$
- ▶  $32 \leq \text{power} \leq 64$ : OverflowError:  
signed integer is greater than maximum
- ▶  $\text{power} \geq 64$ : 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  $\implies$  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  $\implies$  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  $\implies$  missed errors
- ▶ Only types
- ▶ Typedhed: 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
- ▶ Typedhed: type annotations for the standard library, used in previous work: Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020.

# 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

```
1 typedef struct {
2     PyObject_HEAD;
3     int count;
4 } Counter;
5
6 static PyObject*
7 CounterIncr(Counter *self, PyObject *args)
8 {
9     int i = 1;
10    if(!PyArg_ParseTuple(args, "|i", &i))
11        return NULL;
12
13    self->count += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
19 {
20     return Py_BuildValue("i", self->count);
21 }
```

count.py

```
1 from counter import Counter
2 from random import randrange
3
4 c = Counter()
5 power = randrange(128)
6 c.incr(2**power-1)
7 c.incr()
8 r = c.get()
```

# Analysis result

counter.c	count.py
<pre>1 typedef struct { 2     PyObject_HEAD; 3     int count; 4 } Counter; 5 6 static PyObject* 7 CounterIncr(Counter *self, 8 { 9     int i = 1; 10    if(!PyArg_ParseTuple(a 11        return NULL; 12 13    self-&gt;count += i; 14    Py_RETURN_NONE; 15 } 16 17 static PyObject* 18 CounterGet(Counter *self) 19 { 20     return Py_BuildValue(" 21 }</pre>	<pre>1 from counter import Counter 2 from random import randrange</pre> <p>△ Check #430: ./counter.c: In function 'CounterIncr': ./counter.c:13.2-18: warning: Integer overflow</p> <p>13: self-&gt;count += i; ^^^^^^^^^^^^^^^^^^</p> <p>'(self-&gt;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</p> <p>X Check #506: count.py: In function 'PyErr_SetString': count.py:6.0-14: error: OverflowError exception</p> <p>6: c.incr(2**p-1) ^^^^^^^^^^^^^^^^^^</p> <p>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]: PyTuple_int from count.py:7.0-14: CounterIncr +1 other callstack</p>

## Concrete definition

- ▶ Builds upon the Python and C semantics

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views



## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views
- ▶ Boundary functions when objects switch views for the first time

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views
- ▶ Boundary functions when objects switch views for the first time

## Limitations

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views
- ▶ Boundary functions when objects switch views for the first time

## Limitations

- ▶ Garbage collection not handled

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views
- ▶ Boundary functions when objects switch views for the first time

## Limitations

- ▶ Garbage collection not handled
- ▶ C access to Python objects only through the API (verified by Mopsa)

## Concrete definition

- ▶ Builds upon the Python and C semantics
- ▶ Defines the API: calls between languages, value conversions
- ▶ Shared memory, with disjoint, complementary views
- ▶ Boundary functions when objects switch views for the first time

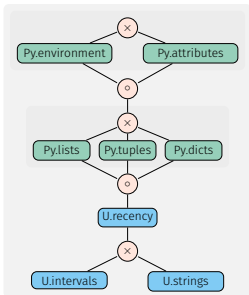
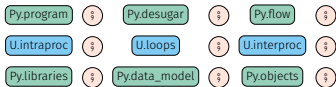
## Limitations

- ▶ Garbage collection not handled
- ▶ C access to Python objects only through the API (verified by Mopsa)
- ▶ Manual modelization from CPython's source code

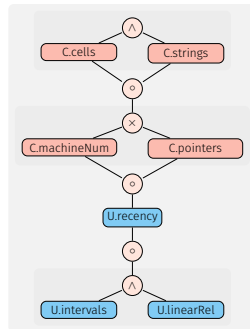
## Implementation & Experimental Evaluation

---

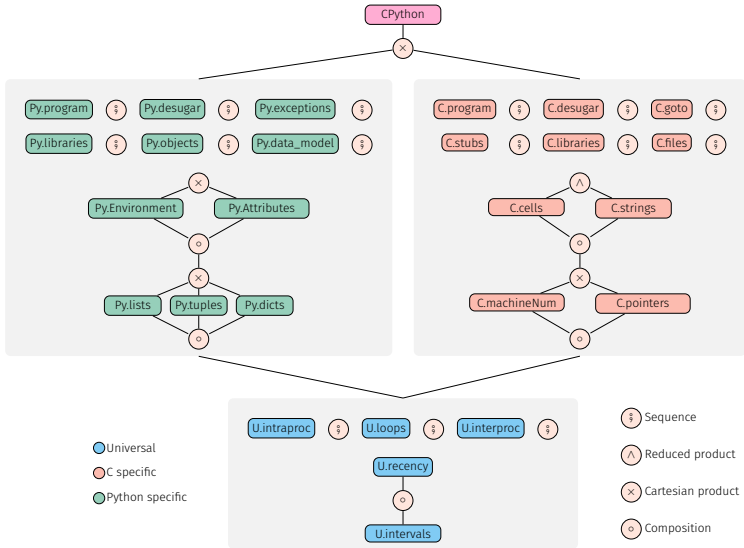
# From distinct Python and C analyses...



- Universal
- C
- Python specific
- ⋮ Sequence
- ∧ Reduced product
- × Cartesian product
- Composition

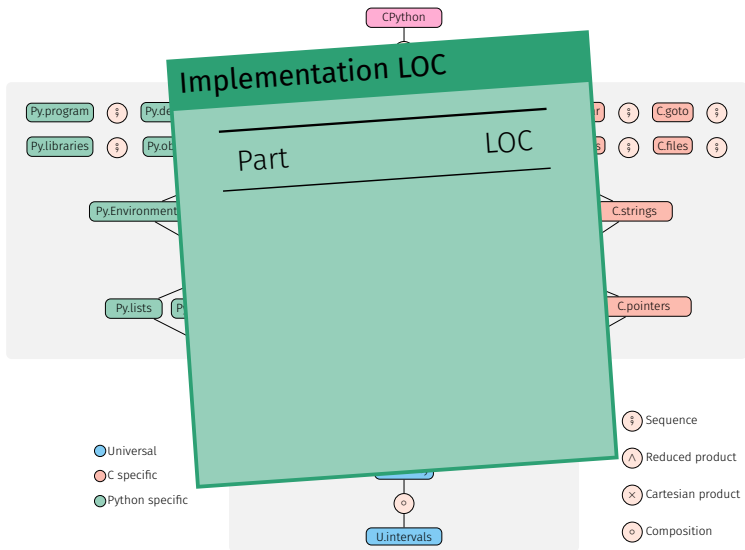


# ... to a multilanguage analysis!

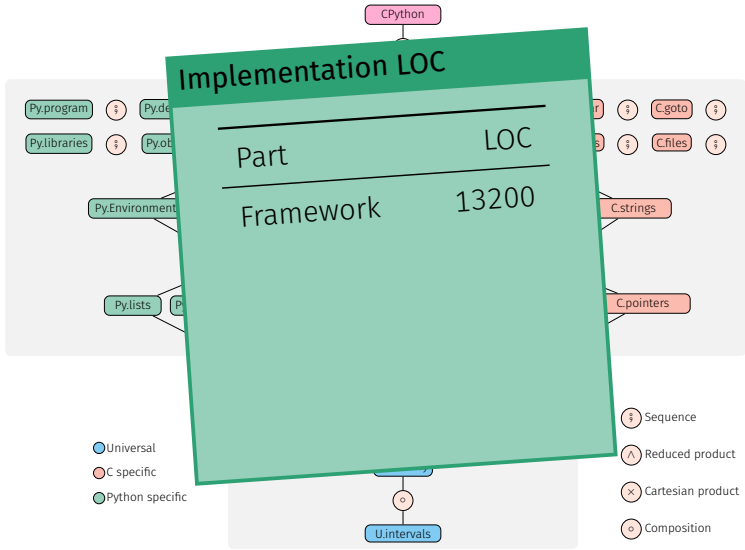




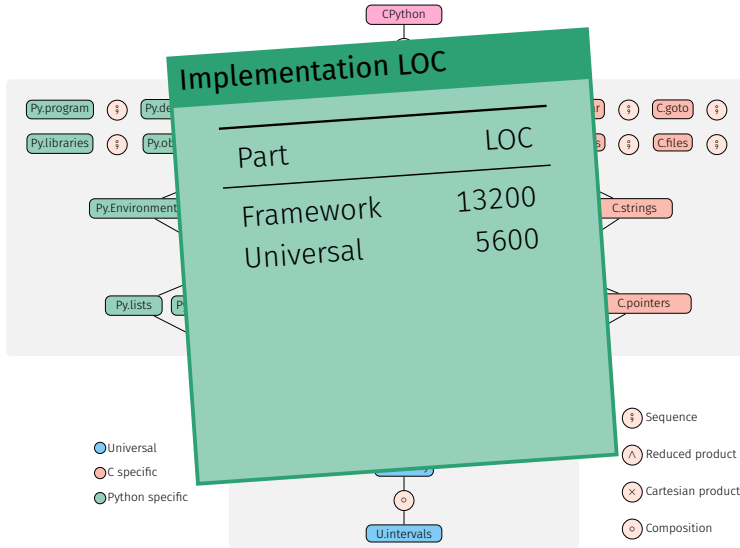
# ... to a multilanguage analysis!



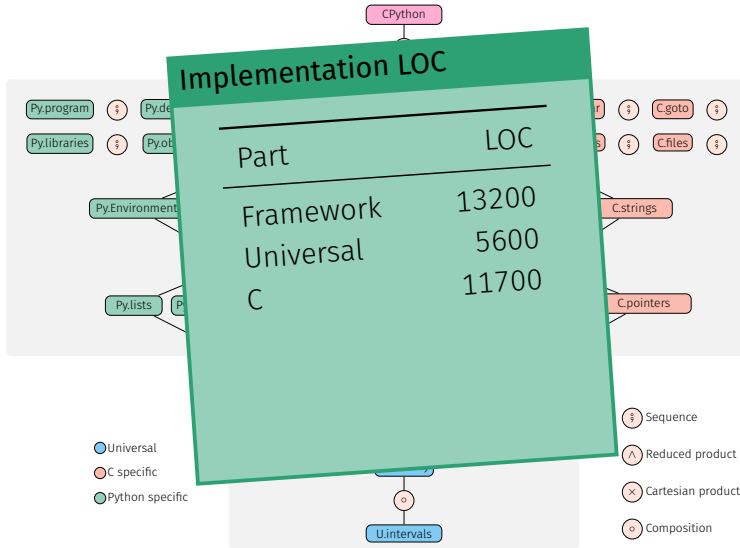
# ... to a multilanguage analysis!



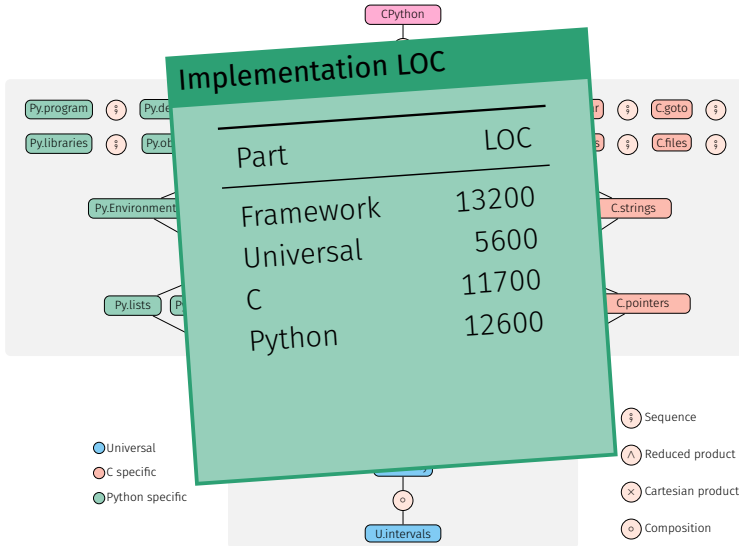
# ... to a multilanguage analysis!



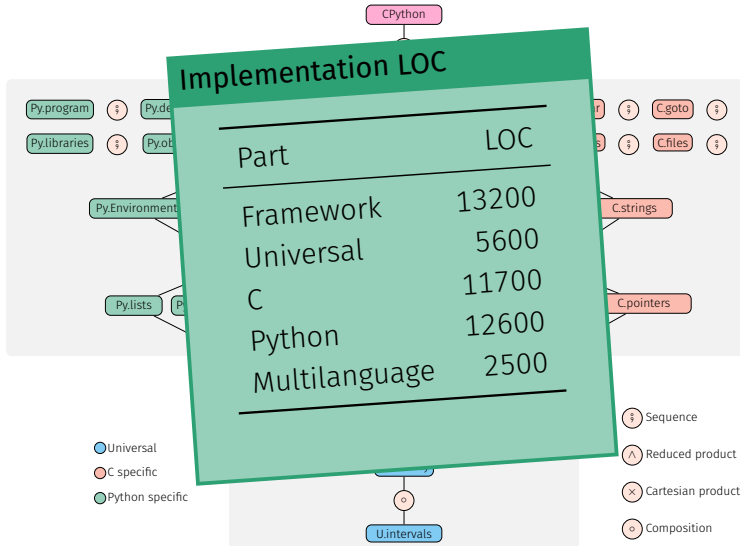
# ... to a multilanguage analysis!



# ... to a multilanguage analysis!



# ... to a multilanguage analysis!



## 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	Tests	🕒	🔴	🟢	Assertions	Py ↔ C
noise	722	675	15/15	18s	99.6% (4952)	100.0% (1738)	0/21	6.5
ahocorasick	3541	1336	46/92	54s	93.1% (1785)	98.0% (4937)	30/88	5.4
levenshtein	5441	357	17/17	1.5m	79.9% (3106)	93.2% (1719)	0/38	2.7
cdistance	1433	912	28/28	1.9m	95.3% (1832)	98.3% (11884)	88/207	8.7
llist	2829	1686	167/194	4.2m	99.0% (5311)	98.8% (30944)	235/691	51.7
bitarray	3244	2597	159/216	4.2m	96.3% (4496)	94.6% (21070)	100/378	14.8

$\frac{\text{safe C checks}}{\text{total C checks}} \%$

total C checks

average # transitions between Python and C per test

## Conclusion

---



# Contribution: multilanguage Python/C analysis

## Difficulties

- ▶ Concrete semantics
- ▶ Memory interaction

---

Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021

# Contribution: multilanguage Python/C analysis

## Difficulties

- ▶ Concrete semantics
- ▶ Memory interaction

## Previous works

- ▶ Type/exceptions analyses for the JNI
- ▶ No detection of runtime errors in C

---

Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021

# Contribution: multilanguage Python/C analysis

## Difficulties

- ▶ Concrete semantics
- ▶ Memory interaction

## Previous works

- ▶ Type/exceptions analyses for the JNI
- ▶ No detection of runtime errors in C

## Results

- ▶ Careful separation of the states and modelization of the API
- ▶ Lightweight domain on top of off-the-shelf C and Python analyses
- ▶ Shared underlying abstractions (numeric, recency)
- ▶ Scale to small, real-world libraries (using client code)

---

Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021

### Multilanguage analyses

- ▶ Other interoperability frameworks (Cffi, Swig, Cython)
- ▶ Bigger applications, need compositional analyses

## Multilanguage analyses

- ▶ Other interoperability frameworks (Cffi, Swig, Cython)
- ▶ Bigger applications, need compositional analyses

## Library analyses

- ▶ Library analysis without client code
- ▶ Infer Typedshed's annotations

# A Multilanguage Static Analysis of Python/C Programs with Mopsa

## Questions

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné

SRG, Imperial College, London

17 March 2022

rmonat.fr

