

# Static Type and Value Analysis by Abstract Interpretation of Python Programs with Native C Libraries

Raphaël Monat

MPRI lecture  
31 January 2021



# Introduction

---

# Goals of the lecture

## Whoami

- ▶ MPRI 2017-2018
- ▶ PhD 2018-2021
- ▶ Currently ATER

## Lecture

- ▶ A recent take at the PhD, from the other side...
- ▶ Around the analysis of Python.

## Growing popularity

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

---

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

# Dynamic programming 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>

# Dynamic programming 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>

# Dynamic programming 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>

# Dynamic programming 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>

# Dynamic programming 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>

## State of the art

Well-established & industrialized analysis of static programming languages

- ▶ C: Polyspace (1999), Astrée (2003), Frama-C (2008)
- ▶ Java: Julia (2010)

## State of the art

### Well-established & industrialized analysis of static programming languages

- ▶ C: Polyspace (1999), Astrée (2003), Frama-C (2008)
- ▶ Java: Julia (2010)

### Around JavaScript

- ▶ First: Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009
- ▶ Bodin et al. “A trusted mechanised JavaScript specification”. POPL 2014

## State of the art

### Well-established & industrialized analysis of static programming languages

- ▶ C: Polyspace (1999), Astrée (2003), Frama-C (2008)
- ▶ Java: Julia (2010)

### Around JavaScript

- ▶ First: Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009
- ▶ Bodin et al. “A trusted mechanised JavaScript specification”. POPL 2014

### Why Python?

Used a lot in

- ▶ Scientific computing
- ▶ Scripts and automation

# Outline

- 1 Introduction
- 2 A Taste of Python
- 3 The Mopsa Analysis Platform
- 4 Analyzing Python Programs
- 5 Analyzing Python Programs with C Libraries
- 6 Conclusion

# What can we prove?

## Average

```
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
```

# What can we prove?

Average

```
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
```

Highly polymorphic function. More context would help!

# What can we prove?

Average

```
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
```

Average

```
7 average([1,2,3])
8 average([])
9 average(range(2, 10, 3))
10 average({0: 3.14, 1: 2.78})
```

Highly polymorphic function. More context would help!

# What can we prove?

Average

```
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
```

Average

```
7 average([1,2,3])
8 try: average([])
9 except NameError: pass
10 average(range(2, 10, 3))
11 average({0: 3.14, 1: 2.78})
```

Highly polymorphic function. More context would help!

# What can we prove?

Average

```
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
```

Average

```
7 average([1,2,3])
8 try: average([])
9 except NameError: pass
10 average(range(2, 10, 3))
11 average({0: 3.14, 1: 2.78})
```

Highly polymorphic function. More context would help!

Detect raised exceptions interrupting the execution.

# A Taste of Python

---

## No standard

- ▶ CPython is the reference
  - ⇒ manual inspection of the source code and handcrafted tests

# Python's specificities

## 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 __getattribute__(self, attr):  
5         if attr[0] == "_": raise AttributeError("protected")  
6         return object.__getattribute__(self, attr)  
7     a = Protected(42)  
8 a._priv # AttributeError raised
```

# Python's specificities (II)

## 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())
```

# 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())
```

## Previous works on Python 3

Guth. "A formal semantics of Python 3.3". 2013

Implementation within the K framework.

## Previous works on Python 3

Guth. "A formal semantics of Python 3.3". 2013

Implementation within the K framework.

Politz et al. "Python: The full monty". OOPSLA 2013

Complex desugaring into  $\lambda_{\pi}$ .

May incur losses of precision in the abstract interpreter.

## Previous works on Python 3

Guth. "A formal semantics of Python 3.3". 2013

Implementation within the K framework.

Politz et al. "Python: The full monty". OOPSLA 2013

Complex desugaring into  $\lambda_\pi$ .

May incur losses of precision in the abstract interpreter.

Köhl. "An Executable Structural Operational Formal Semantics for Python". 2021

Semantics of Python, using a Python framework, developed concurrently.

## Previous works on Python 3

Guth. "A formal semantics of Python 3.3". 2013

Implementation within the K framework.

Politz et al. "Python: The full monty". OOPSLA 2013

Complex desugaring into  $\lambda_\pi$ .

May incur losses of precision in the abstract interpreter.

Köhl. "An Executable Structural Operational Formal Semantics for Python". 2021

Semantics of Python, using a Python framework, developed concurrently.

Different goal

These works focus on the concrete semantics. This is not our endgoal.

## Previous works on Python 3

Guth. "A formal semantics of Python 3.3". 2013

Implementation within the K framework.

Politz et al. "Py Moving to our own semantics"

Complex desugaring of Python code.

May incur losses of precision.

- ▶ Cost of understanding the code (vs CPython)
- ▶ Trust in the code (CPython's tests?)
- ▶ Insights of the papers

Köhl. "An Executable Formal Semantics for Python". 2021

Semantics of Python, using a Python framework, developed concurrently.

### Different goal

These works focus on the concrete semantics. This is not our endgoal.

# Our approach

## Interpreter-like semantics

Easily convertible to an abstract interpreter.

---

<sup>2</sup>Fromherz, Ouadjaout, and Miné. “Static Value Analysis of Python Programs by Abstract Interpretation”. NFM 2018.

# Our approach

## Interpreter-like semantics

Easily convertible to an abstract interpreter.

## Major extension of the work of Fromherz, Ouadjaout, and Miné<sup>2</sup>

- ▶ Separation between core and builtins
- ▶ 2.3× more cases (**with** statement, bidirectional generators, ...)
- ▶ Improved some cases (+, boolean casts of conditionals, data descriptors, ...)

---

<sup>2</sup>Fromherz, Ouadjaout, and Miné. “Static Value Analysis of Python Programs by Abstract Interpretation”. NFM 2018.

# Our approach

## Interpreter-like semantics

Easily convertible to an abstract interpreter.

## Major extension of the work of Fromherz, Ouadjaout, and Miné<sup>2</sup>

- ▶ Separation between core and builtins
- ▶ 2.3× more cases (**with** statement, bidirectional generators, ...)
- ▶ Improved some cases (+, boolean casts of conditionals, data descriptors, ...)

## Correctness

- ▶ Strived to make it auditable (with links to the source).
- ▶ Tested only through the abstract analysis yet (no concrete execution).

---

<sup>2</sup>Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". NFM 2018.

## Example – attribute access

```
Ecur[(x.s)](cur, e, h)  $\stackrel{\text{def}}{=}$  LOAD_ATTR PyObject_GetAttr (slot_tp_getattr_hook)  
letb (cur, e, h), @x = E[x](cur, e, h) in  
letb (cur, e, h), @c = E[mro_search(type(@x), "__getattribute__")](cur, e, h) in  
letcases (f, e, h), @x.s = E[@c(@x, s)](cur, e, h) in  
match f with  
• exn @exc when isinstance(@exc, AttributeError)  $\Rightarrow$   
  let (f, e, h), @d = E[mro_search(type(@x), "__getattr__")](f, e, h) in  
    if d  $\neq \perp$  then return E[@d(@x, s)](cur, e, h)  
    else return (f, e, h),  $\perp$   
• _  $\Rightarrow$  return (f, e, h), @x.s
```

## Example – attribute access

```
Ecur[(x.s)](cur, e, h)  $\stackrel{\text{def}}{=}$  LOAD_ATTR PyObject_GetAttr (slot_tp_getattr_hook)
letb (cur, e, h), @x = E[x](cur, e, h) in
  ...
Ecur[(object.__getattribute__)(obj, name)](cur, e, h)  $\stackrel{\text{def}}{=}$ 
  tp_field _PyObject_GenericGetAttrWithDict
letb (cur, e, h), @o = E[obj](cur, e, h) in
letb (cur, e, h), @n = E[name](cur, e, h) in
if  $\neg$ isinstance(@n, str) then return S[raise TypeError](cur, e, h), ⊥ else
  let str(n) = fst oh(@n) in
  letcases (f, e, h), @descr = E[mro_search(type(@o), n)](f, e, h) in
    if @descr  $\neq$  ⊥ then
      if hasattr(type(@descr), "__get__")  $\wedge$ 
        (hasattr(type(@descr), "__set__")  $\vee$  hasattr(type(@descr), "__delete__")) then
          return E[type(@descr).__get__(@descr, @o, type(@o))](f, e, h)
```

## Example – attribute access

$E_{cur}[\![x.s]\!](cur, e, h) \stackrel{\text{def}}{=} \text{LOAD_ATTR PyObject_GetAttr}(\text{slot_tp_getattr_hook})$

letb  $(cur, e, h), @_x = E[\![x]\!](cur, e, h)$  in

$E_{cur}[\![\text{object}.__getattribute__}\!(\text{obj}, \text{name})]\!](cur, e, h) \stackrel{\text{def}}{=} \text{tp_field PyObject_GenericGetAttrWithDict}$

letb  $(cur, e, h), @_o = E[\![\text{obj}]\!](cur, e, h)$  in

letb  $(cur, e, h), @_n = E[\![\text{name}]\!]$  in

$E_{cur}[\![\text{type}.__getattribute__}\!(\text{typ}, \text{name})]\!](cur, e, h) \stackrel{\text{def}}{=} \text{tp_field type_getattro}$

letb  $(cur, e, h), @_typ = E[\![\text{typ}]\!](cur, e, h)$  in

letb  $(cur, e, h), @_name = E[\![\text{name}]\!](cur, e, h)$  in

$E_{cur}[\![\text{mro_search}(\text{type}(@_typ), @_name)]]\!](cur, e, h)$  in

letb  $(cur, e, h), @_meta = E[\![\text{mro_search}(\text{type}(@_typ), @_name)]]\!](cur, e, h)$  in

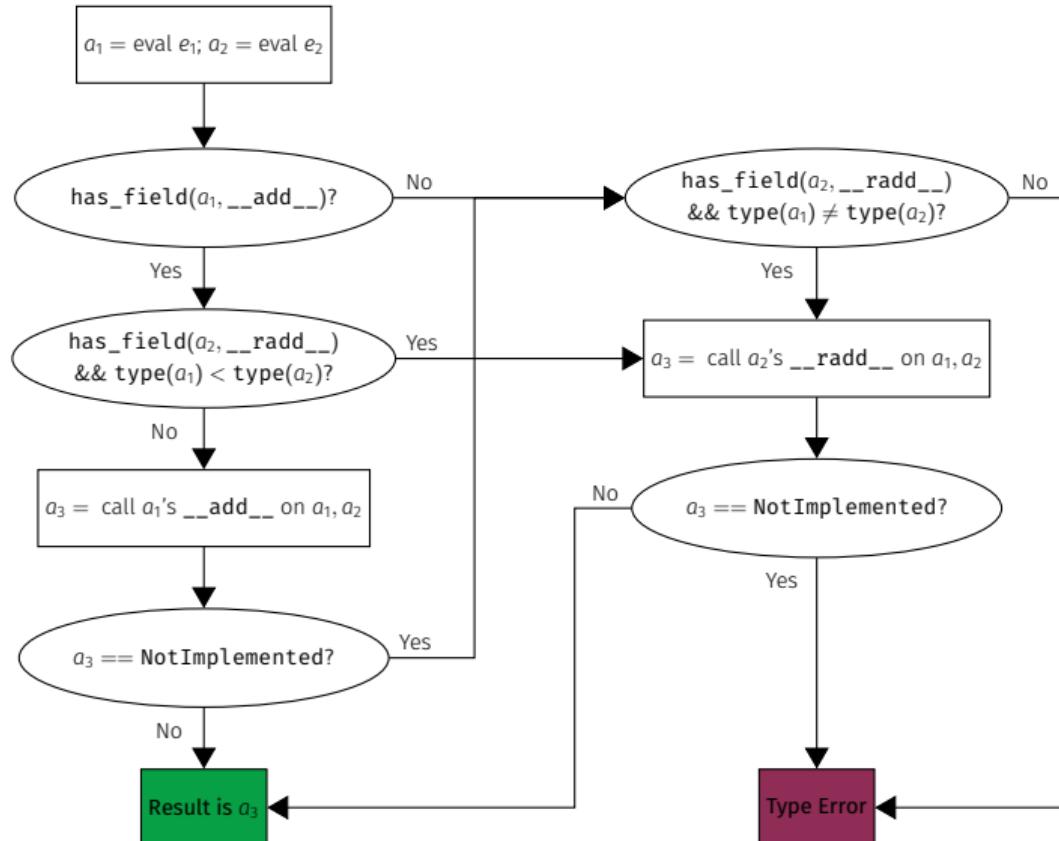
if  $@_meta \neq \perp$  then

if  $\text{hasattr}(\text{type}(@_meta), \text{"__get__"}) \wedge$

$(\text{hasattr}(\text{type}(@_meta), \text{"__set__"}) \vee \text{hasattr}(\text{type}(@_meta), \text{"__delete__"}))$  then

$E_{cur}[\![(\text{cur}, \text{e}, \text{h}), @_tvp, \text{type}(@_typ)]]\!](cur, e, h)$

## Example – binary operators



# Crazy Python

## Custom infix operators

```
1 class Infix(object):
2     def __init__(self, func): self.func = func
3     def __or__(self, other): return self.func(other)
4     def __ror__(self, other): return Infix(lambda x: self.func(other, x))
5
6 instanceof = Infix(isinstance)
7 b = 5 |instanceof| int
8
9 @Infix
10 def padd(x, y):
11     print("{x} + {y} = {x + y}")
12     return x + y
13 c = 2 |padd| 3
```

---

Credits [tomerfiliba.com/blog/Infix-Operators/](http://tomerfiliba.com/blog/Infix-Operators/)

# The Mopsa Analysis Platform

---

# 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)
```

# 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)
```

Searching for a loop invariant (l. 4)

Environment abstraction

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

Proved safe?

- ▶  $m // (i+1)$
- ▶  $l[i]$

# 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)
```

Searching for a loop invariant (l. 4)

Stateless domains: **list content**,

## Environment abstraction

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

Proved safe?

- ▶  $m // (i+1)$
- ▶  $l[i]$

# 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)

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

# 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)

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

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

# 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)

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

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

# 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

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

## Numeric abstraction (polyhedra)

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

## Attributes abstraction

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

# 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,
14     for i in range(randint(5, 10)))]
15 m = average(l)
```

## Conclusion

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

Proved safe?

- ▶  $m // (i+1)$
- ▶  $l[i].weight$

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

## Environment abstraction

$m \mapsto @_{\text{Task}}^{\#} \quad ; \quad @_{\text{Task}}^{\#}$

- ▶  $l \mapsto \{l[i] \mid 0 \leq i < \underline{\text{len}}(l)\}$
- ▶  $0 \leq i < \underline{\text{len}}(l) \quad 5 \leq \underline{\text{len}}(l) \leq 10$
- ▶  $0 \leq @_{\text{Task}}^{\#} \cdot \underline{\text{weight}} \leq 20$

## Attributes abstraction

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



Modular Open Platform for Static Analysis<sup>3</sup>

---

<sup>3</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>3</sup>

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

---

<sup>3</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>3</sup>

- ▶ One AST to analyze them all
  - 🚩 Multilanguage support
  - 📄 Expressiveness
  - ♻️ Reusability
- ▶ Unified domain signature
  - 📝 Semantic rewriting
  - 🧩 Loose coupling
  - ⌚ Observability

---

<sup>3</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>3</sup>

- ▶ 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>3</sup>Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019.

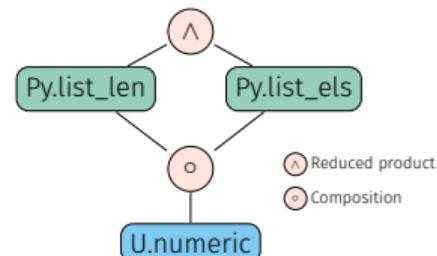


## Modular Open Platform for Static Analysis<sup>3</sup>

- ▶ 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>3</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
```



C.iterators.loops

Rewrite and analyze recursively

Universal.Iterators.Loops

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

Computes fixpoint using widening

# Dynamic, semantic iterators with delegation

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



C.iterators.loops

Rewrite and analyze recursively

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

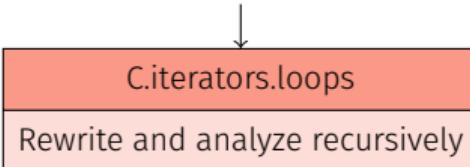
Universal.Iterators.Loops

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

Computes fixpoint using widening

# Dynamic, semantic iterators with delegation

`for(init; cond; incr) body`



`for target in iterable: body`

# Dynamic, semantic iterators with delegation

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



C.iterators.loops

Rewrite and analyze recursively

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



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



Python.Desugar.Loops

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

Universal.Iterators.Loops

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

# Dynamic, semantic iterators with delegation

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



C.iterators.loops

Rewrite and analyze recursively



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

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

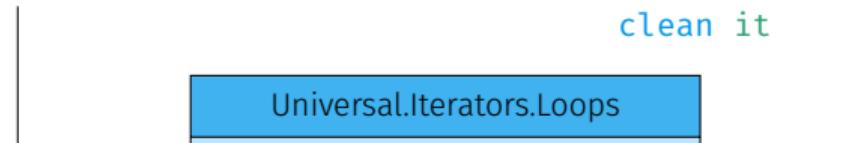


Python.Desugar.Loops

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



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



Universal.Iterators.Loops

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

# Hooks as observers of the analysis

## Idea

Observe analyzer's state before/after any eval/exec

Hook signature

```
1 module type STATELESS_HOOK =
2 sig
3   val name : string
4   val init : 'a ctx -> unit
5
6   val on_before_exec : stmt -> ('a,'a) man -> 'a flow -> unit
7   val on_after_exec : stmt -> ('a,'a) man -> 'a flow -> ('a, unit) cases -> unit
8
9   val on_before_eval : expr -> ('a,'a) man -> 'a flow -> unit
10  val on_after_eval : expr -> ('a,'a) man -> 'a flow -> ('a, expr) cases -> unit
11
12  val on_finish : ('a,'a) man -> 'a flow -> unit
13 end
```

# Example of hooks: Logs

## Logs

- ▶ Display the evaluation tree
- ▶ Optionally, display the abstract state at each point

```
+ S [] r = []; []
+ E [] [] : py []
| + E [] alloc(list, STRONG) : addr []
| o E [] alloc(list, STRONG) : addr [] = @list:3ae881f4d:s : addr done [0.0001s, 1 case]
* reaching dependent_len.py:8.4-6
+ S [] @list:3ae881f4d:s.list_length = 0; []
| + E [] 0 : int []
| o E [] 0 : int [] = 0 : int done [0.0001s, 1 case]
* reaching dependent_len.py:8.4-6
+ S [] @list:3ae881f4d:s.list_length = 0; [] in below(universal.iterators.intraproc)
o S [] @list:3ae881f4d:s.list_length = 0; [] in below(universal.iterators.intraproc) done [0.0001s, 1 case]
| o S [] @list:3ae881f4d:s.list_length = 0; [] done [0.0001s, 1 case]
o E [] [] : py [] = {@list:3ae881f4d:s} : py done [0.0002s, 1 case]
o S [] r = []; [] done [0.0002s, 1 case]
```

## Example of hooks: Coverage

### Coverage

- ▶ Global metric for the analysis' results
- ▶ Good to detect dead code, and soundness issues

```
def sum(l):  
    s = 0  
    for x in l:  
        s += x  
    return s  
  
r2 = sum(['a', 'b', 'c'])  
r1 = sum([1, 2, 3])
```

# Example of hooks: Profiling

## Motivation

- ▶ `perf`, `memtrace` too low-level and global
- ▶ Higher-level view by profiling at the analyzed program's level
- ▶ Inlining and nested loops  $\implies$  analysis time  $\propto$  program size

```
Product
1 def p(l1, l2):
2     r = []
3     for x in l1:
4         for y in l2:
5             r.append((x, y))
6     return r
7
8 r1 = p([1,2,3], [4,5,6])
9 r2 = p(['a', 'b'], ['c', 'd'])
```

## Loop Profiler

```
nested.py:3.4-6.4: 2 times,
                      # iterations (3, 3)
nested.py:4.8-6.4: 6 times,
                      # iterations (3, 1, 1, 3, 1, 1)
```

## Function Profiler

```
p          0.0544s(total)    x2
```

# Analyzing Python Programs

---

## Analyses overview

### Goal

Detect runtime errors: uncaught raised exceptions

# Analyses 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

# Analyses 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

# Attribute abstraction

## Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

# Attribute abstraction

## Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

After line 5: a may have attribute y.

# Attribute abstraction

## Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

After line 5: a may have attribute y.

After line 6, either

- ▶ a must have attribute y
- ▶ or an **AttributeError** has been raised

# Attribute abstraction

## Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

After line 5: a may have attribute y.

After line 6, either

- ▶ a must have attribute y
- ▶ or an **AttributeError** has been raised

⇒ We need both an under and an over-approximation of the attributes.

## Attribute abstraction (II)

Using an under and an over-approximation

$$\text{ObjS}^\sharp = \{ (u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{ \top \}, u \subseteq o \vee o = \top \}$$

## Attribute abstraction (II)

Using an under and an over-approximation

$$\text{ObjS}^\sharp = \{ (u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{ \top \}, u \subseteq o \vee o = \top \}$$

Concretization

$$\gamma_{\text{ObjS}}^\sharp : \begin{cases} \text{ObjS}^\sharp & \rightarrow \mathcal{P}(\mathcal{P}(\text{string})) \\ (u, \top) & \mapsto \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \} \\ (u, o) & \mapsto \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \subseteq o \} \end{cases}$$

## Attribute abstraction (II)

Using an under and an over-approximation

$$\text{ObjS}^\sharp = \{ (u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{ \top \}, u \subseteq o \vee o = \top \}$$

Concretization

$$\gamma_{\text{ObjS}}^\sharp : \begin{cases} \text{ObjS}^\sharp & \rightarrow \mathcal{P}(\mathcal{P}(\text{string})) \\ (u, \top) & \mapsto \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \} \\ (u, o) & \mapsto \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \subseteq o \} \end{cases}$$

Example

$$\gamma_{\text{ObjS}}^\sharp(\{x\}, \{x, y, z\}) = \{ \{x\}, \{x, y\}, \{x, z\}, \{x, y, z\} \}$$

## Attribute abstraction (III)

Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

How to lift the attribute abstraction?

## Attribute abstraction (III)

Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

How to lift the attribute abstraction?  $\mathcal{V} \rightarrow \text{ObjS}^\sharp$ ?

## Attribute abstraction (III)

Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A()
4 a.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

How to lift the attribute abstraction?  $\mathcal{V} \rightarrow \text{ObjS}^\sharp$ ?

After line 5,  $a \mapsto \{x\}, \{x, y\}$ ?

## Attribute abstraction (III)

Non-deterministic attribute handling

```
1 class A: pass
2
3 a = A(); b = a
4 b.x = 42
5 if *: a.y = 3
6 r = a.x * a.y
```

How to lift the attribute abstraction?  $\mathcal{V} \rightarrow \text{ObjS}^\sharp$ ?

After line 5,  $a \mapsto \{x\}, \{x, y\}$ ?

Different variables may point to a same object, we need a memory abstraction:

$\text{Addr}^\sharp \rightarrow \text{ObjS}^\sharp$

## Handling memory

N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses  $\text{Addr}^\sharp$

## Handling memory

N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses  $\text{Addr}^\sharp$ (why finite?)

## Handling memory

N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses  $\text{Addr}^\sharp$ (why finite?)

```
1 class A: pass
2
3 n = random_int()
4 for i in range(n):
5     a = A()
6     a.val = i
```

## Handling memory

N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses  $\text{Addr}^\sharp$ (why finite?)

```
1 class A: pass
2
3 n = random_int()
4 for i in range(n):
5     a = A()
6     a.val = i
```

Need to handle unbounded allocations.

## Allocation site abstraction

- ▶  $\text{Addr}^\# = \text{Loc}$
- ▶ Weak updates on abstract addresses

Attribute addition

```
1 class A: pass
2
3 a = A()
4 a.x = 42
```

After line 3:  $\text{@}_3^\# \mapsto \emptyset, \emptyset$

Weak update line 4:

$$(\text{@}_3^\# \mapsto \{x\}, \{x\}) \sqcup^\# (\text{@}_3^\# \mapsto \emptyset, \emptyset) =$$

## Allocation site abstraction

- ▶  $\text{Addr}^\# = \text{Loc}$
- ▶ Weak updates on abstract addresses

Attribute addition

```
1 class A: pass
2
3 a = A()
4 a.x = 42
```

After line 3:  $\text{@}_3^\# \mapsto \emptyset, \emptyset$

Weak update line 4:

$$(\text{@}_3^\# \mapsto \{x\}, \{x\}) \sqcup^\# (\text{@}_3^\# \mapsto \emptyset, \emptyset) = \text{@}_3^\# \mapsto \emptyset, \{x\}$$

## Allocation site abstraction

- ▶  $\text{Addr}^\# = \text{Loc}$
- ▶ Weak updates on abstract addresses

Attribute addition

```
1 class A: pass
2
3 a = A()
4 a.x = 42
```

After line 3:  $\text{@}_3^\# \mapsto \emptyset, \emptyset$

Weak update line 4:

$$(\text{@}_3^\# \mapsto \{x\}, \{x\}) \sqcup^\# (\text{@}_3^\# \mapsto \emptyset, \emptyset) = \text{@}_3^\# \mapsto \emptyset, \{x\}$$

## How to perform precise operations?

### The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization
- ▶ Twofold partitioning:

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization
- ▶ Twofold partitioning:
  - by allocation site  $l \in \text{Loc}$

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization
- ▶ Twofold partitioning:
  - by allocation site  $l \in \text{Loc}$
  - through a recency criterion:  $(l, r)$  most recent allocation (with strong updates)  
 $(l, o)$  older addresses (summarized)

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization
- ▶ Twofold partitioning:
  - by allocation site  $l \in \text{Loc}$
  - through a recency criterion:  $(l, r)$  most recent allocation (with strong updates)  
 $(l, o)$  older addresses (summarized)
- ▶ Initially designed for analysis of low-level code (binaries, C)

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## The recency abstraction<sup>4</sup>

- ▶ Precise analysis of object initialization
- ▶ Twofold partitioning:
  - by allocation site  $l \in \text{Loc}$
  - through a recency criterion:  $(l, r)$  most recent allocation (with strong updates)  
 $(l, o)$  older addresses (summarized)
- ▶ Initially designed for analysis of low-level code (binaries, C)
- ▶ Also used in Type Analysis for JavaScript<sup>5</sup>

---

<sup>4</sup>Balakrishnan and Reps. “Recency-Abstraction for Heap-Allocated Storage”. SAS 2006.

<sup>5</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

## Handling memory | Recency abstraction (II)

```
Task creation
```

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 m = [1, 2]
7 l = [Task(i) for i in m]
8 l.append(Task(3))
```

### Type analysis

Nominal types used in abstract  
addresses. No need for allocation-site  
in Tasks. But helpful for lists!

# Handling memory | Recency abstraction (II)

```
Task creation
```

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 m = [1, 2]
7 l = [Task(i) for i in m]
8 l.append(Task(3))
```

## Type analysis

Nominal types used in abstract addresses. No need for allocation-site in **Tasks**. But helpful for lists!

## Value analysis

Use allocation sites for **range** objects.

## Handling memory | Recency abstraction (II)

```
Task creation
```

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 m = [1, 2]
7 l = [Task(i) for i in m]
8 l.append(Task(3))
```

### Type analysis

Nominal types used in abstract addresses. No need for allocation-site in Tasks. But helpful for lists!

### Value analysis

Use allocation sites for range objects.

## Variable allocation policies

- ▶ Type-based (nominal) and/or location-based partitioning.
- ▶ Different configurations depending on type/value analysis.

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

$$\left\{ @^{\sharp}(\text{Task}, r) \mapsto \emptyset, \emptyset \right.$$

Allocation:

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

$\{ @\#(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \}$

Initialization:

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, r) \mapsto \emptyset, \emptyset \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

Allocation:  $@\sharp(\text{Task}, r) \rightsquigarrow @\sharp(\text{Task}, o)$

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:  $\text{@}^\sharp(\text{Task}, r) \rightsquigarrow \text{@}^\sharp(\text{Task}, o)$

$$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \emptyset, \emptyset \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:  $\text{@}^\sharp(\text{Task}, r) \rightsquigarrow \text{@}^\sharp(\text{Task}, o)$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, r) \mapsto \emptyset, \emptyset \end{array} \right.$

$\left\{ \begin{array}{l} \text{@}^\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$

## Handling memory | Recency abstraction (III)

```
1 class Task:  
2     def __init__(self, weight):  
3         if weight < 0: raise ValueError  
4         self.weight = weight  
5  
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

$$\left\{ \begin{array}{l} @\sharp(\text{Task}, r) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\ @\sharp(\text{Task}, o) \mapsto \{ \text{weight} \}, \{ \text{weight} \} \end{array} \right.$$

Analysis of  $l[0].x = 3$ ? What about  $l[3].x = 3$ ?

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

$\left\{ @^\sharp(\text{Task}, r) \right.$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

$$\left\{ @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \right.$$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \\ @^\sharp(\text{Task}, r) \\ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \end{array} \right.$$

Allocation:  $@^\sharp(\text{Task}, r) \rightsquigarrow @^\sharp(\text{Task}, o)$

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \\ @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [1, 1] \\ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \end{array} \right.$$

Initialization:

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:  $\mathbb{O}^\sharp(\text{Task}, r) \rightsquigarrow \mathbb{O}^\sharp(\text{Task}, o)$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [1, 1] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \sqcup [1, 1] \right.$$

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\left\{ @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [1, 1] \right.$$

$$\left\{ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [4, 4] \right.$$

$$\left\{ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [1, 2] \right.$$

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:  $\mathbb{O}^\sharp(\text{Task}, r) \rightsquigarrow \mathbb{O}^\sharp(\text{Task}, o)$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [1, 1] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [4, 4] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [1, 2] \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, r) \right.$$

$$\left\{ \mathbb{O}^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [1, 2] \sqcup [4, 4] \right.$$

## Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes:  $\text{Addr}^\sharp \times \text{string} \subseteq \mathcal{V}$

```
1 class Task:
2     def __init__(self, weight):
3         if weight < 0: raise ValueError
4         self.weight = weight
5
6 l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \end{array} \right.$$

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [1, 1] \\ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \end{array} \right.$$

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [4, 4] \\ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [1, 2] \end{array} \right.$$

$$\left\{ \begin{array}{l} @^\sharp(\text{Task}, r) \cdot \text{weight} \mapsto [5, 5] \\ @^\sharp(\text{Task}, o) \cdot \text{weight} \mapsto [1, 4] \end{array} \right.$$

## List abstraction

- ▶ Summarization of the content (auxiliary variable)
- ▶ Auxiliary length variable

## List abstraction

- ▶ Summarization of the content (auxiliary variable)
- ▶ Auxiliary length variable

## Dictionaries in Python

- ▶ Keys can be any object (JavaScript: strings or symbols)
- ▶ Key/value summarization currently used
- ▶ To be combined with an attribute-like abstraction for strings?

## Python List Abstraction

- ▶ Smash each list into one weak, abstract contents variable.
- ▶ The contents variable is built upon the list's abstract address.
- ▶ Delegate to memory and numeric domains.

```
E#[[e1, ..., en]loc]s  $\stackrel{\text{def}}{=}$ 
let s, @ = E#[ alloc(List loc) ]s in
let contents = mk_var @ "contents" in
S#[ contents  $\stackrel{\text{weak}}{=}$  en ]o...o S#[ contents = e1 ]s, @
```

## Python List Abstraction

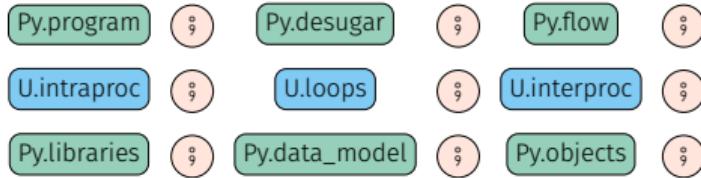
- ▶ Smash each list into one weak, abstract contents variable.
- ▶ The contents variable is built upon the list's abstract address.
- ▶ Delegate to memory and numeric domains.

```
E#[[e1, ..., en]loc]s  $\stackrel{\text{def}}{=}$ 
  let s, @ = E#[ alloc(List loc) ]s in
    let contents = mk_var @ "contents" in
      S#[ contents  $\stackrel{\text{weak}}{=}$  en ]o ... o S#[ contents = e1 ]s, @
```

Demo!

- ▶ Dynamicity:  
type inference first
- ▶ Flow-sensitive
- ▶ Context-sensitive

# Types | Analysis



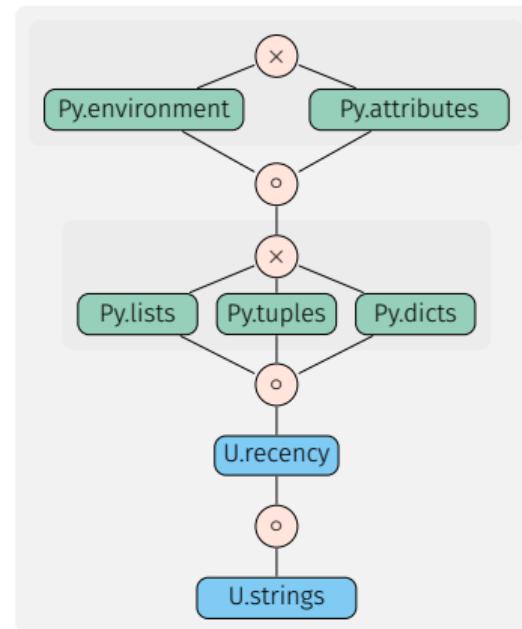
Sequence

Cartesian product

Composition

Universal

Python specific



- ▶ Dynamicity:  
type inference first
- ▶ Flow-sensitive
- ▶ Context-sensitive

- ▶ Similar in essence to TAJS.<sup>6</sup>
- ▶ Dataflow analysis by Fritz and Hage.<sup>7</sup>
- ▶ Typpete: SMT-based type inference.<sup>8</sup>
- ▶ Pytype, type inference tool used by Google.<sup>9</sup>
- ▶ RPython: efficient compilation of a static subset of Python.<sup>10</sup>
- ▶ Value analysis by Fromherz et al.<sup>11</sup>

---

<sup>6</sup>Jensen, Møller, and Thiemann. “Type Analysis for JavaScript”. SAS 2009.

<sup>7</sup>Fritz and Hage. “Cost versus precision for approximate typing for Python”. PEPM 2017.

<sup>8</sup>Hassan, Urban, Eilers, and Müller. “MaxSMT-Based Type Inference for Python 3”. CAV 2018.

<sup>9</sup>Kramm et al. Pytype. 2019.

<sup>10</sup>Ancona, Ancona, Cuni, and Matsakis. “RPython: a step towards reconciling dynamically and statically typed OO languages”. DLS 2007.

<sup>11</sup>Fromherz, Ouadjaout, and Miné. “Static Value Analysis of Python Programs by Abstract Interpretation”. NFM 2018.

# Types | Experimental evaluation

Name	LOC	Mopsa	⚠	Fritz & Hage	Pypyte	Typpete	Fromherz et al.	RPython
bellman_ford.py	61	0.24s	0†	1.4s	0.99s	1.4s	2.4m	7.1s
float.py	63	82ms	0†	1.7s	0.92s	1.3s	0.84s	5.6s
coop_concat.py	64	43ms	0†	1.8s	0.81s	1.3s	20ms	⌚⌚
crafting.py	132	0.41s	0†🔑	1.6s	0.97	1.7s	⌚⌚	⌚⌚
nbody.py	157	0.80s	1†🔑‡	1.7s	1.3s	⌚⌚	⌚⌚	⌚⌚
chaos.py	324	2.3s	0†‡	13s	11s	⌚⌚	⌚⌚	⌚⌚
scimark.py	416	0.55s	2†	8.5s	4.4s	⌚⌚	⌚⌚	⌚⌚
richards.py	426	5.0s	2†‡	38s	2.4s	⌚⌚	⌚⌚	7.8s
unpack_seq.py	458	4.2s	0‡	1.1s	7.4s	2.7s	14s	⌚⌚
go.py	461	15s	32†‡	8.5s	3.4s	⌚⌚	⌚⌚	⌚⌚
hexiom.py	674	22s	25†🔑‡	⌚⌚	4.2s	⌚⌚	⌚⌚	⌚⌚
regex_v8.py	1792	15s	0†	4.9s	⌚	1.7m	⌚⌚	⌚⌚
processInput.py	1417	4.8s	7†🔑‡	2.4s	11s	⌚⌚	⌚⌚	⌚⌚
choose.py	2562	46s	17†‡	1.7s	15s	⌚⌚	⌚⌚	⌚⌚

⌚ unsupported by the analyzer (crash) ⌄ timeout (after 1h)

Smashed exceptions: KeyError 🔑, IndexError †, ValueError ‡

Name	LOC	Mopsa	⚠	Fritz & Hage	Ptytype	Typpete	Fromherz et al.	RPython
bellman_ford.py	61	0.24s	0†	1.4s	0.99s	1.4s	2.4m	7.1s
float.py	63	82ms	0†	1.7s	0.92s	1.3s	0.84s	5.6s
coop_concat.py	64	43ms	0†	1.8s	0.81s	1.3s	20ms	⌚⌚
crafting.py	132	0.41s	0†🔑	1.6s	0.71s	1.6s	⌚⌚	⌚⌚
nbody.py	157						⌚⌚	⌚⌚
chaos.py	324						⌚⌚	⌚⌚
scimark.py	416						⌚⌚	⌚⌚
richards.py	426						⌚⌚	7.8s
unpack_seq.py	458						14s	⌚⌚
go.py	461						⌚⌚	⌚⌚
hexiom.py	674						⌚⌚	⌚⌚
regex_v8.py	1792	15s	0†	4.9s	⌚	1.7m	⌚⌚	⌚⌚
processInput.py	1417	4.8s	7†🔑*	2.4s	11s	⌚⌚	⌚⌚	⌚⌚
choose.py	2562	46s	17†🔑*	1.7s	15s	⌚⌚	⌚⌚	⌚⌚

## Conclusion

- ▶ Handling Python's dynamicity
- ▶ Good scalability (w.r.t. other semantic tools)

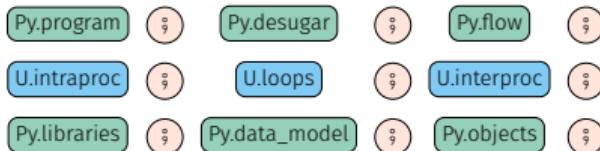
⌚ unsupported by the analyzer (crash) ⌂ timeout (after 1h)  
 Smashed exceptions: KeyError 🔑, IndexError †, ValueError \*

## Types $\rightsquigarrow$ values | Configurations

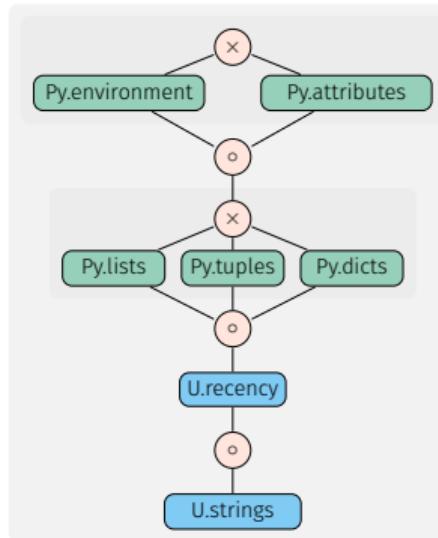
Thanks to Mopsa, switching from types to values is straightforward!

# Types $\rightsquigarrow$ values | Configurations

Thanks to Mopsa, switching from types to values is straightforward!

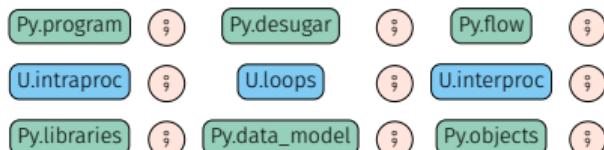
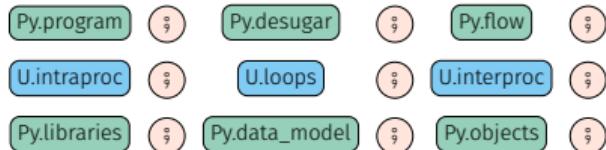


- ∅ Sequence
- × Cartesian product
- Composition
- Universal
- Python specific

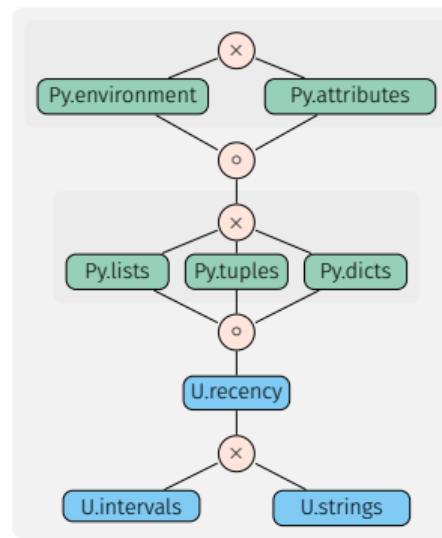
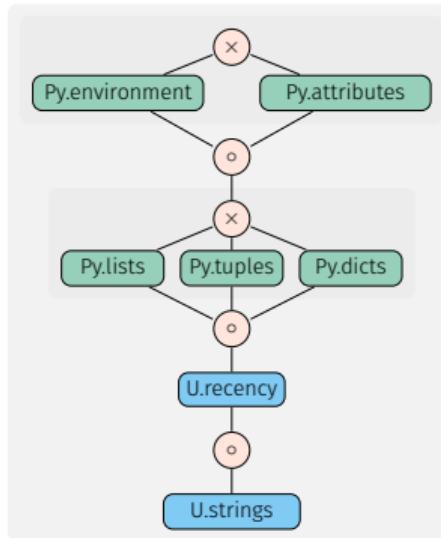


# Types $\rightsquigarrow$ values | Configurations

Thanks to Mopsa, switching from types to values is straightforward!



- (?) Sequence
- (x) Cartesian product
- (o) Composition
- (blue) Universal
- (green) Python specific



# Types $\rightsquigarrow$ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- **ValueError** (l. 3)

# Types $\rightsquigarrow$ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- ▶ **ValueError** (l. 3)
- ▶ **IndexError** (l. 9)

# Types ↵ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- ▶ **ValueError** (l. 3)
- ▶ **IndexError** (l. 9)
- ▶ **ZeroDivisionError** (l. 10)

# Types ↵ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- ▶ **ValueError** (l. 3)
- ▶ **IndexError** (l. 9)
- ▶ **ZeroDivisionError** (l. 10)
- ▶ **NameError** (l. 10)

# Types $\rightsquigarrow$ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- ▶ **ValueError** (l. 3)
- ▶ **IndexError** (l. 9)
- ▶ **ZeroDivisionError** (l. 10)
- ▶ **NameError** (l. 10)

## Non-relational value analysis

**IndexError** (l. 9)

# Types $\rightsquigarrow$ values | Comparing the analyses

```
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 = []
14 for i in range(randint(5, 10)):
15     l.append(Task(randint(0, 20)))
16 m = average(l)
```

## Type analysis

- ▶ **ValueError** (l. 3)
- ▶ **IndexError** (l. 9)
- ▶ **ZeroDivisionError** (l. 10)
- ▶ **NameError** (l. 10)

## Non-relational value analysis

**IndexError** (l. 9)

## Relational value analysis

No alarm!

## Types $\rightsquigarrow$ values | Comparing the analyses (II)

Name	LOC	Type Analysis						Non-relational Value Analysis						
		Time	Mem.	Exceptions detected			Time	Mem.	Exceptions detected			Type	Index	Key
				Type	Index	Key			Type	Index	Key			
nbbody.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			

## Types ↵ values | Comparing the analyses (II)

Name	LOC	Type Analysis			Non-relational Value Analysis				
		Time	Mem.	Exceptions detected	Time	Mem.	Exceptions detected	Index	Key
		Type	Index	Key					
nbbody.py	157	1.5s	—	—	—	—	—	1	1
scimark.py	416	1.4s	—	—	—	—	—	0	0
richards.py	426	13s	—	—	—	—	—	2	0
unpack_seq.py	458	8.3s	—	—	—	—	—	0	0
go.py	461	27s	—	—	—	—	—	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	—	—	—	—	—	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

## Types $\rightsquigarrow$ values | Comparing the analyses (II)

Name	LOC	Type Analysis			Non-relational Value Analysis				
		Time	Mem.	Exceptions detected	Time	Mem.	Exceptions detected	Index	Key
Type	Index	Key	Type	Index	Key	Index	Key	Index	Key
nbbody.py	157	1.5s	—	—	—	—	—	1	1
scimark.py	416	1.4s	—	—	—	—	—	0	0
richards.py	426	13s	—	—	—	—	—	2	0
unpack_seq.py	458	8.3s	—	—	—	—	—	0	0
go.py	461	27s	—	—	—	—	—	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	—	—	—	—	—	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  $2.5\times$  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

## Two soundnesses

- ▶ Modelization of the semantics from CPython
- ▶ Implementation of this semantics within Mopsa

# Soundness

## Two soundnesses

- ▶ Modelization of the semantics from CPython
- ▶ Implementation of this semantics within Mopsa

## Our approach

- ▶ Test only in the abstract
- ▶ Issue of overapproximations and unproved assertions

## Unsupported constructs

- ▶ eval
- ▶ Recursive functions
- ▶ Finalizers

# Soundness

## Two soundnesses

- ▶ Modelization of the semantics from CPython
- ▶ Implementation of this semantics within Mopsa

## Our approach

- ▶ Test only in the abstract
- ▶ Issue of overapproximations and unproved assertions

## Unsupported constructs

- ▶ eval
- ▶ Recursive functions
- ▶ Finalizers

## Tests from previous works

- ▶ 450/586 tests supported
- ▶ 268/586 assertions proved

## Official tests from CPython

- ▶ 325/416 tests supported (17 chosen files)
- ▶ 389/702 assertions proved

# 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 object representations (Python objects, C structs)

# 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 object representations (Python objects, C structs)
- ▶ 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 object representations (Python objects, C structs)
- ▶ Different runtime-errors (exceptions in Python)
- ▶ Garbage collection

# Combining C and Python – first look

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API
- ▶ To detect runtime errors (in C, Python, and the “glue”)

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API
- ▶ To detect runtime errors (in C, Python, and the “glue”)
- ▶ Observations

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API
- ▶ To detect runtime errors (in C, Python, and the “glue”)
- ▶ Observations
  - allocated objects are shared in the memory

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API
- ▶ To detect runtime errors (in C, Python, and the “glue”)
- ▶ Observations
  - allocated objects are shared in the memory
  - but each language has different abstractions

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

## A combined static analysis of C/Python<sup>12</sup>

- ▶ Targeting C extensions using the CPython API
  - ▶ To detect runtime errors (in C, Python, and the “glue”)
  - ▶ Observations
    - allocated objects are shared in the memory
    - but each language has different abstractions
- ⇒ Share universal domains and synchronize abstractions

---

<sup>12</sup>Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.

# 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  $\Rightarrow$  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
- ▶ 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
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,  13: self-&gt;count += i; 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("</pre>	<pre>1 from counter import Counter 2 from random import randrange 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 339 340 341 342 343 344 345 346 347 348 349 349 350 351 352 353 354 355 356 357 358 359 359 360 361 362 363 364 365 366 367 368 369 369 370 371 372 373 374 375 376 377 378 379 379 380 381 382 383 384 385 386 387 387 388 389 389 390 391 392 393 394 395 396 397 397 398 399 399 400 401 402 403 404 405 406 407 408 409 409 410 411 412 413 414 415 416 417 418 419 419 420 421 422 423 424 425 426 427 428 429 429 430 431 432 433 434 435 436 437 438 439 439 440 441 442 443 444 445 446 447 447 448 449 449 450 451 452 453 454 455 456 457 458 459 459 460 461 462 463 464 465 466 467 468 469 469 470 471 472 473 474 475 476 477 478 479 479 480 481 482 483 484 485 486 487 487 488 489 489 490 491 492 493 494 495 496 497 497 498 499 499 500 501 502 503 504 505 506 507 508 509 509 510 511 512 513 514 515 516 517 518 519 519 520 521 522 523 524 525 526 527 528 529 529 530 531 532 533 534 535 536 537 538 539 539 540 541 542 543 544 545 546 547 547 548 549 549 549 550 551 552 553 554 555 556 557 558 559 559 560 561 562 563 564 565 566 567 567 568 569 569 569 570 571 572 573 574 575 576 577 578 579 579 580 581 582 583 584 585 586 587 587 588 589 589 589 590 591 592 593 594 595 596 597 597 598 599 599 599 600 601 602 603 604 605 606 607 608 609 609 610 611 612 613 614 615 616 617 617 618 619 619 619 620 621 622 623 624 625 626 627 627 628 629 629 629 630 631 632 633 634 635 636 637 637 638 639 639 639 640 641 642 643 644 645 646 646 647 648 648 648 649 650 651 652 653 654 655 656 657 657 658 659 659 659 660 661 662 663 664 665 666 667 667 668 669 669 669 670 671 672 673 674 675 676 677 677 678 679 679 679 680 681 682 683 684 685 686 687 687 688 689 689 689 690 691 692 693 694 695 696 697 697 698 699 699 699 700 701 702 703 704 705 706 707 707 708 709 709 709 710 711 712 713 714 715 716 716 717 718 718 718 719 720 721 722 723 724 725 726 726 727 728 728 728 729 730 731 732 733 734 735 736 736 737 738 738 738 739 740 741 742 743 744 745 745 746 747 747 747 748 749 749 749 750 751 752 753 754 755 756 756 757 758 758 758 759 760 761 762 763 764 765 766 766 767 768 768 768 769 770 771 772 773 774 775 776 776 777 778 778 778 779 780 781 782 783 784 785 786 786 787 788 788 788 789 790 791 792 793 794 795 795 796 797 797 797 798 799 799 799 800 801 802 803 804 805 806 806 807 808 808 808 809 810 811 812 813 814 815 815 816 817 817 817 818 819 819 819 820 821 822 823 824 825 826 826 827 828 828 828 829 830 831 832 833 834 835 835 836 837 837 837 838 839 839 839 840 841 842 843 844 845 845 846 847 847 847 848 849 849 849 850 851 852 853 854 855 855 856 857 857 857 858 859 859 859 860 861 862 863 864 865 865 866 867 867 867 868 869 869 869 870 871 872 873 874 875 875 876 877 877 877 878 879 879 879 880 881 882 883 884 885 885 886 887 887 887 888 889 889 889 890 891 892 893 894 894 895 896 896 896 897 898 898 898 899 900 901 902 903 903 904 905 905 905 906 907 907 907 908 909 909 909 910 911 912 913 914 914 915 916 916 916 917 918 918 918 919 920 921 922 923 923 923 924 925 925 925 926 927 927 927 928 929 929 929 930 931 932 933 934 934 935 936 936 936 937 938 938 938 939 940 941 942 943 943 943 944 945 945 945 946 947 947 947 948 949 949 949 950 951 952 953 954 954 955 956 956 956 957 958 958 958 959 960 961 962 963 963 963 964 965 965 965 966 967 967 967 968 969 969 969 970 971 972 973 974 974 974 975 976 976 976 977 978 978 978 979 980 981 982 983 983 983 984 985 985 985 986 987 987 987 988 989 989 989 990 991 992 993 993 993 994 995 995 995 996 997 997 997 998 999 999 999 1000 1001 1002 1003 1003 1003 1004 1005 1005 1005 1006 1007 1007 1007 1008 1009 1009 1009 1010 1011 1012 1013 1013 1013 1014 1015 1015 1015 1016 1017 1017 1017 1018 1019 1019 1019 1020 1021 1022 1023 1023 1023 1024 1025 1025 1025 1026 1027 1027 1027 1028 1029 1029 1029 1030 1031 1032 1033 1033 1033 1034 1035 1035 1035 1036 1037 1037 1037 1038 1039 1039 1039 1040 1041 1042 1043 1043 1043 1044 1045 1045 1045 1046 1047 1047 1047 1048 1049 1049 1049 1050 1051 1052 1053 1053 1053 1054 1055 1055 1055 1056 1057 1057 1057 1058 1059 1059 1059 1060 1061 1062 1063 1063 1063 1064 1065 1065 1065 1066 1067 1067 1067 1068 1069 1069 1069 1070 1071 1072 1073 1073 1073 1074 1075 1075 1075 1076 1077 1077 1077 1078 1079 1079 1079 1080 1081 1082 1083 1083 1083 1084 1085 1085 1085 1086 1087 1087 1087 1088 1089 1089 1089 1090 1091 1092 1093 1093 1093 1094 1095 1095 1095 1096 1097 1097 1097 1098 1099 1099 1099 1100 1101 1102 1103 1103 1103 1104 1105 1105 1105 1106 1107 1107 1107 1108 1109 1109 1109 1110 1111 1112 1113 1113 1113 1114 1115 1115 1115 1116 1117 1117 1117 1118 1119 1119 1119 1120 1121 1122 1123 1123 1123 1124 1125 1125 1125 1126 1127 1127 1127 1128 1129 1129 1129 1130 1131 1132 1133 1133 1133 1134 1135 1135 1135 1136 1137 1137 1137 1138 1139 1139 1139 1140 1141 1142 1143 1143 1143 1144 1145 1145 1145 1146 1147 1147 1147 1148 1149 1149 1149 1150 1151 1152 1153 1153 1153 1154 1155 1155 1155 1156 1157 1157 1157 1158 1159 1159 1159 1160 1161 1162 1163 1163 1163 1164 1165 1165 1165 1166 1167 1167 1167 1168 1169 1169 1169 1170 1171 1172 1173 1173 1173 1174 1175 1175 1175 1176 1177 1177 1177 1178 1179 1179 1179 1180 1181 1182 1183 1183 1183 1184 1185 1185 1185 1186 1187 1187 1187 1188 1189 1189 1189 1190 1191 1192 1193 1193 1193 1194 1195 1195 1195 1196 1197 1197 1197 1198 1199 1199 1199 1200 1201 1202 1203 1203 1203 1204 1205 1205 1205 1206 1207 1207 1207 1208 1209 1209 1209 1210 1211 1212 1213 1213 1213 1214 1215 1215 1215 1216 1217 1217 1217 1218 1219 1219 1219 1220 1221 1222 1223 1223 1223 1224 1225 1225 1225 1226 1227 1227 1227 1228 1229 1229 1229 1230 1231 1232 1233 1233 1233 1234 1235 1235 1235 1236 1237 1237 1237 1238 1239 1239 1239 1240 1241 1242 1243 1243 1243 1244 1245 1245 1245 1246 1247 1247 1247 1248 1249 1249 1249 1250 1251 1252 1253 1253 1253 1254 1255 1255 1255 1256 1257 1257 1257 1258 1259 1259 1259 1260 1261 1262 1263 1263 1263 1264 1265 1265 1265 1266 1267 1267 1267 1268 1269 1269 1269 1270 1271 1272 1273 1273 1273 1274 1275 1275 1275 1276 1277 1277 1277 1278 1279 1279 1279 1280 1281 1282 1283 1283 1283 1284 1285 1285 1285 1286 1287 1287 1287 1288 1289 1289 1289 1290 1291 1292 1293 1293 1293 1294 1295 1295 1295 1296 1297 1297 1297 1298 1299 1299 1299 1300 1301 1302 1303 1303 1303 1304 1305 1305 1305 1306 1307 1307 1307 1308 1309 1309 1309 1310 1311 1312 1313 1313 1313 1314 1315 1315 1315 1316 1317 1317 1317 1318 1319 1319 1319 1320 1321 1322 1323 1323 1323 1324 1325 1325 1325 1326 1327 1327 1327 1328 1329 1329 1329 1330 1331 1332 1333 1333 1333 1334 1335 1335 1335 1336 1337 1337 1337 1338 1339 1339 1339 1340 1341 1342 1343 1343 1343 1344 1345 1345 1345 1346 1347 1347 1347 1348 1349 1349 1349 1350 1351 1352 1353 1353 1353 1354 1355 1355 1355 1356 1357 1357 1357 1358 1359 1359 1359 1360 1361 1362 1363 1363 1363 1364 1365 1365 1365 1366 1367 1367 1367 1368 1369 1369 1369 1370 1371 1372 1373 1373 1373 1374 1375 1375 1375 1376 1377 1377 1377 1378 1379 1379 1379 1380 1381 1382 1383 1383 1383 1384 1385 1385 1385 1386 1387 1387 1387 1388 1389 1389 1389 1390 1391 1392 1393 1393 1393 1394 1395 1395 1395 1396 1397 1397 1397 1398 1399 1399 1399 1400 1401 1402 1403 1403 1403 1404 1405 1405 1405 1406 1407 1407 1407 1408 1409 1409 1409 1410 1411 1412 1413 1413 1413 1414 1415 1415 1415 1416 1417 1417 1417 1418 1419 1419 1419 1420 1421 1422 1423 1423 1423 1424 1425 1425 1425 1426 1427 1427 1427 1428 1429 1429 1429 1430 1431 1432 1433 1433 1433 1434 1435 1435 1435 1436 1437 1437 1437 1438 1439 1439 1439 1440 1441 1442 1443 1443 1443 1444 1445 1445 1445 1446 1447 1447 1447 1448 1449 1449 1449 1450 1451 1452 1453 1453 1453 1454 1455 1455 1455 1456 1457 1457 1457 1458 1459 1459 1459 1460 1461 1462 1463 1463 1463 1464 1465 1465 1465 1466 1467 1467 1467 1468 1469 1469 1469 1470 1471 1472 1473 1473 1473 1474 1475 1475 1475 1476 1477 1477 1477 1478 1479 1479 1479 1480 1481 1482 1483 1483 1483 1484 1485 1485 1485 1486 1487 1487 1487 1488 1489 1489 1489 1490 1491 1492 1493 1493 1493 1494 1495 1495 1495 1496 1497 1497 1497 1498 1499 1499 1499 1500 1501 1502 1503 1503 1503 1504 1505 1505 1505 1506 1507 1507 1507 1508 1509 1509 1509 1510 1511 1512 1513 1513 1513 1514 1515 1515 1515 1516 1517 1517 1517 1518 1519 1519 1519 1520 1521 1522 1523 1523 1523 1524 1525 1525 1525 1526 1527 1527 1527 1528 1529 1529 1529 1530 1531 1532 1533 1533 1533 1534 1535 1535 1535 1536 1537 1537 1537 1538 1539 1539 1539 1540 1541 1542 1543 1543 1543 1544 1545 1545 1545 1546 1547 1547 1547 1548 1549 1549 1549 1550 1551 1552 1553 1553 1553 1554 1555 1555 1555 1556 1557 1557 1557 1558 1559 1559 1559 1560 1561 1562 1563 1563 1563 1564 1565 1565 1565 1566 1567 1567 1567 1568 1569 1569 1569 1570 1571 1572 1573 1573 1573 1574 1575 1575 1575 1576 1577 1577 1577 1578 1579 1579 1579 1580 1581 1582 1583 1583 1583 1584 1585 1585 1585 1586 1587 1587 1587 1588 1589 1589 1589 1590 1591 1592 1593 1593 1593 1594 1595 1595 1595 1596 1597 1597 1597 1598 1599 1599 1599 1600 1601 1602 1603 1603 1603 1604 1605 1605 1605 1606 1607 1607 1607 1608 1609 1609 1609 1610 1611 1612 1613 1613 1613 1614 1615 1615 1615 1616 1617 1617 1617 1618 1619 1619 1619 1620 1621 1622 1623 1623 1623 1624 1625 1625 1625 1626 1627 1627 1627 1628 1629 1629 1629 1630 1631 1632 1633 1633 1633 1634 1635 1635 1635 1636 1637 1637 1637 1638 1639 1639 1639 1640 1641 1642 1643 1643 1643 1644 1645 1645 1645 1646 1647 1647 1647 1648 1649 1649 1649 1650 1651 1652 1653 1653 1653 1654 1655 1655 1655 1656 1657 1657 1657 1658 1659 1659 1659 1660 1661 1662 1663 1663 1663 1664 1665 1665 1665 1666 1667 1667 1667 1668 1669 1669 1669 1670 1671 1672 1673 1673 1673 1674 1675 1675 1675 1676 1677 1677 1677 1678 1679 1679 1679 1680 1681 1682 1683 1683 1683 1684 1685 1685 1685 1686 1687 1687 1687 1688 1689 1689 1689 1690 1691 1692 1693 1693 1693 1694 1695 1695 1695 1696 1697 1697 1697 1698 1699 1699 1699 1700 1701 1702 1703 1703 1703 1704 1705 1705 1705 1706 1707 1707 1707 1708 1709 1709 1709 1710 1711 1712 1713 1713 1713 1714 1715 1715 1715 1716 1717 1717 1717 1718 1719 1719 1719 1720 1721 1722 1723 1723 1723 1724 1725 1725 1725 1726 1727 1727 1727 1728 1729 1729 1729 1730 1731 1732 1733 1733 1733 1734 1735 1735 1735 1736 1737 1737 1737 1738 1739 1739 1739 1740 1741 1742 1743 1743 1743 1744 1745 1745 1745 1746 1747 1747 1747 1748 1749 1749 1749 1750 1751 1752 1753 1753 1753 1754 1755 1755 1755 1756 1757 1757 1757 1758 1759 1759 1759 1760 1761 1762 1763 1763 1763 1764 1765 1765 1765 1766 1767 1767 1767 1768 1769 1769 1769 1770 1771 1772 1773 1773 1773 1774 1775 1775 1775 1776 1777 1777 1777 1778 1779 1779 1779 1780 1781 1782 1783 1783 1783 1784 1785 1785 1785 1786 1787 1787 1787 1788 1789 1789 1789 1790 1791 1792 1793 1793 1793 1794 1795 1795 1795 1796 1797 1797 1797 1798 1799 1799 1799 1800 1801 1802 1803 1803 1803 1804 1805 1805 1805 1806 1807 1807 1807 1808 1809 1809 1809 1810 1811 1812 1813 1813 1813 1814 1815 1815 1815 1816 1817 1817 1817 1818 1819 1819 1819 1820 1821 1822 1823 1823 1823 1824 1825</pre>

## 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 heap, with disjoint, complementary views

## Concrete definition

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

## Limitations

# Multilanguage semantics

## Concrete definition

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

## Limitations

- ▶ Garbage collection not handled

# Multilanguage semantics

## Concrete definition

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

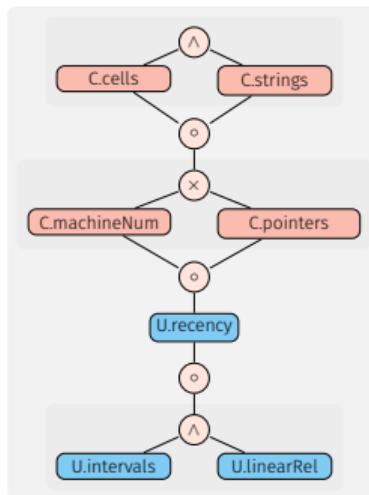
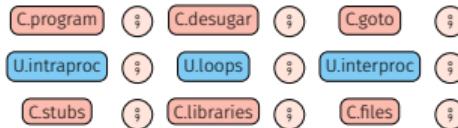
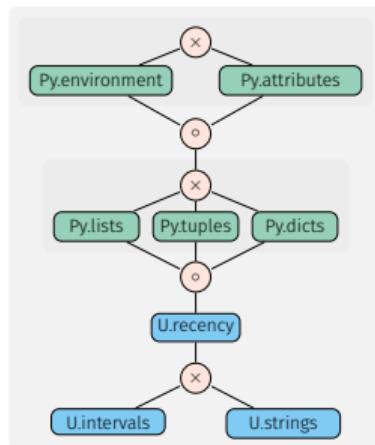
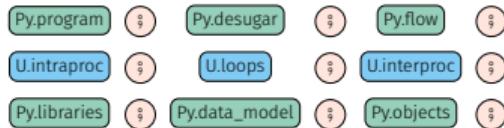
## Concrete definition

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

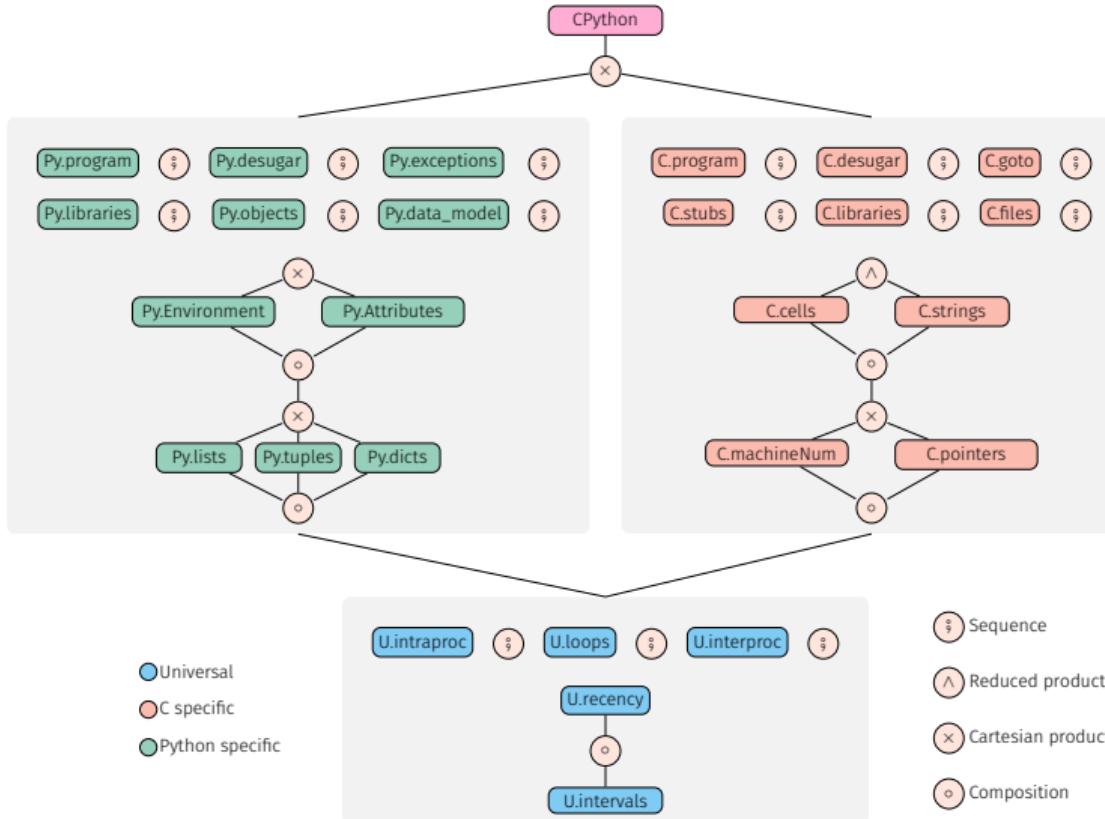
## Limitations

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

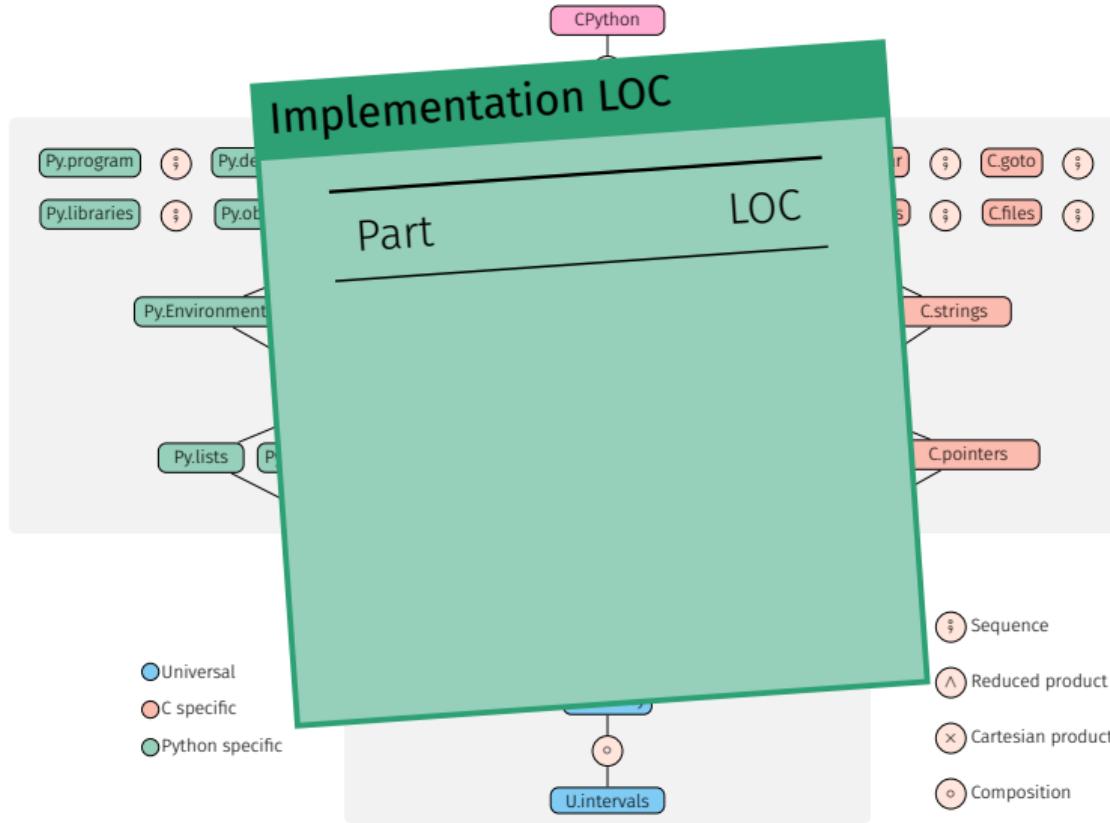
# From distinct Python and C analyses...



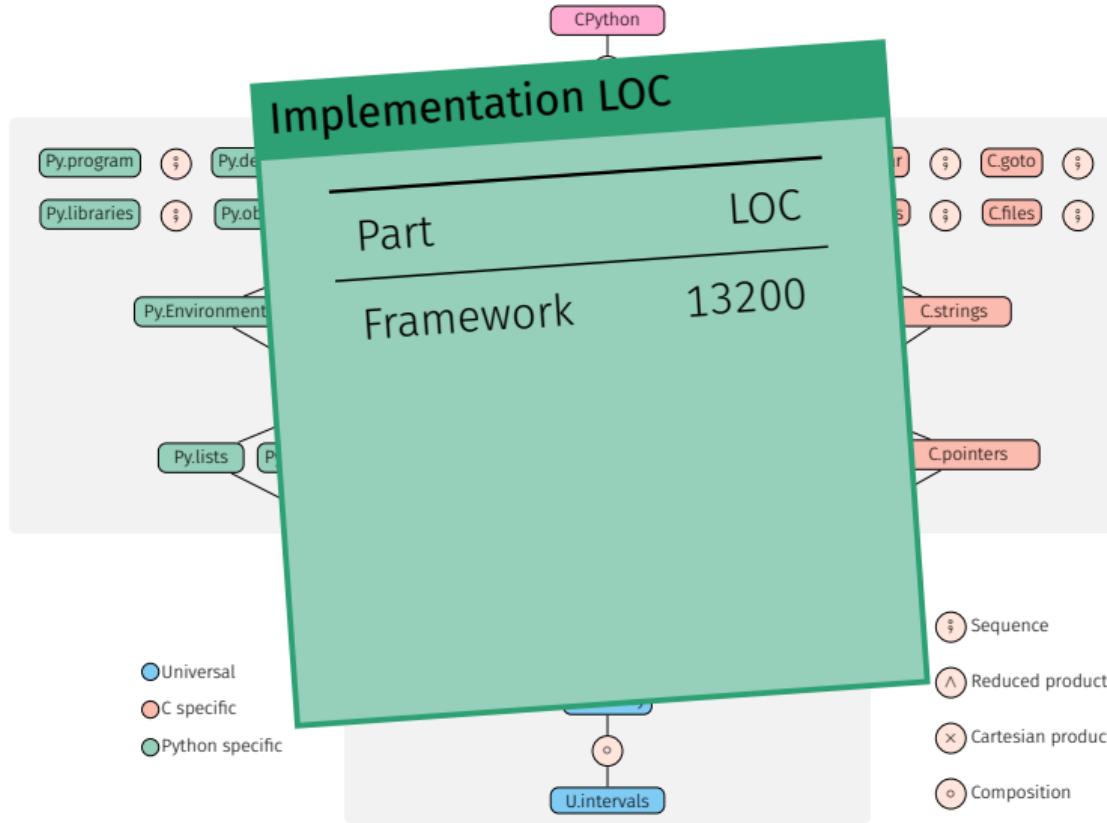
# ... to a multilanguage analysis!



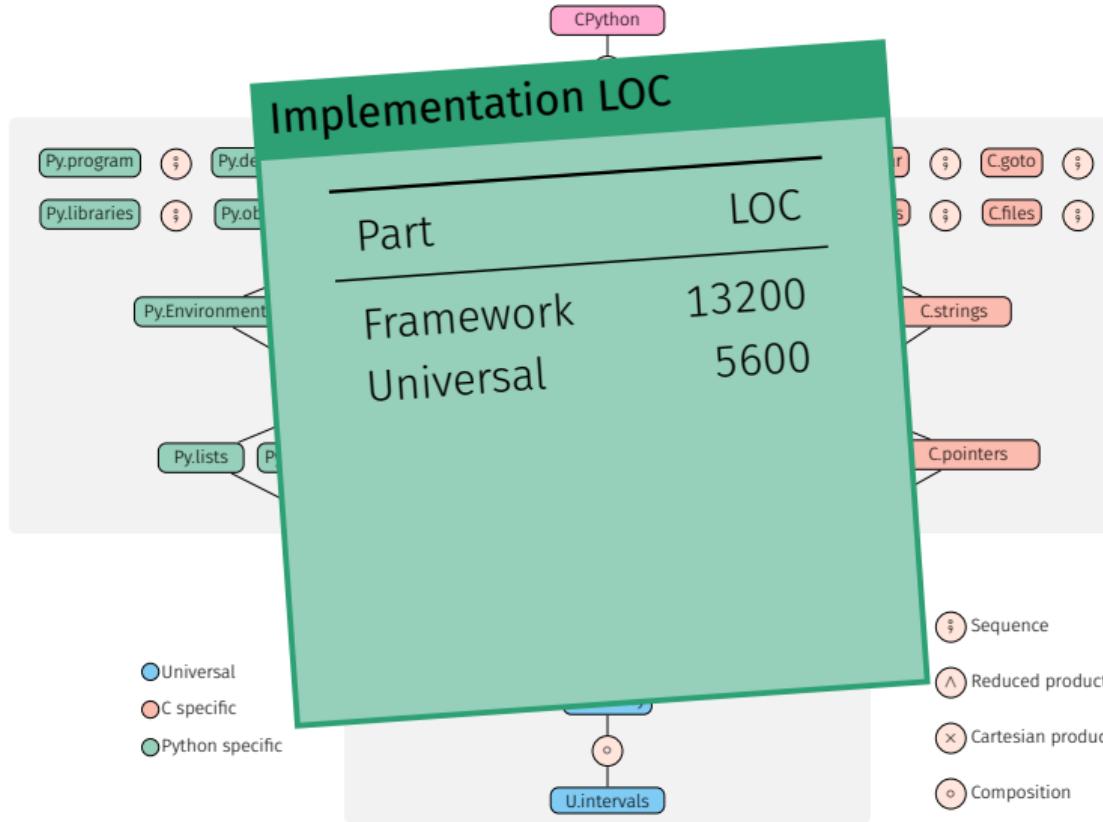
... to a multilanguage analysis!



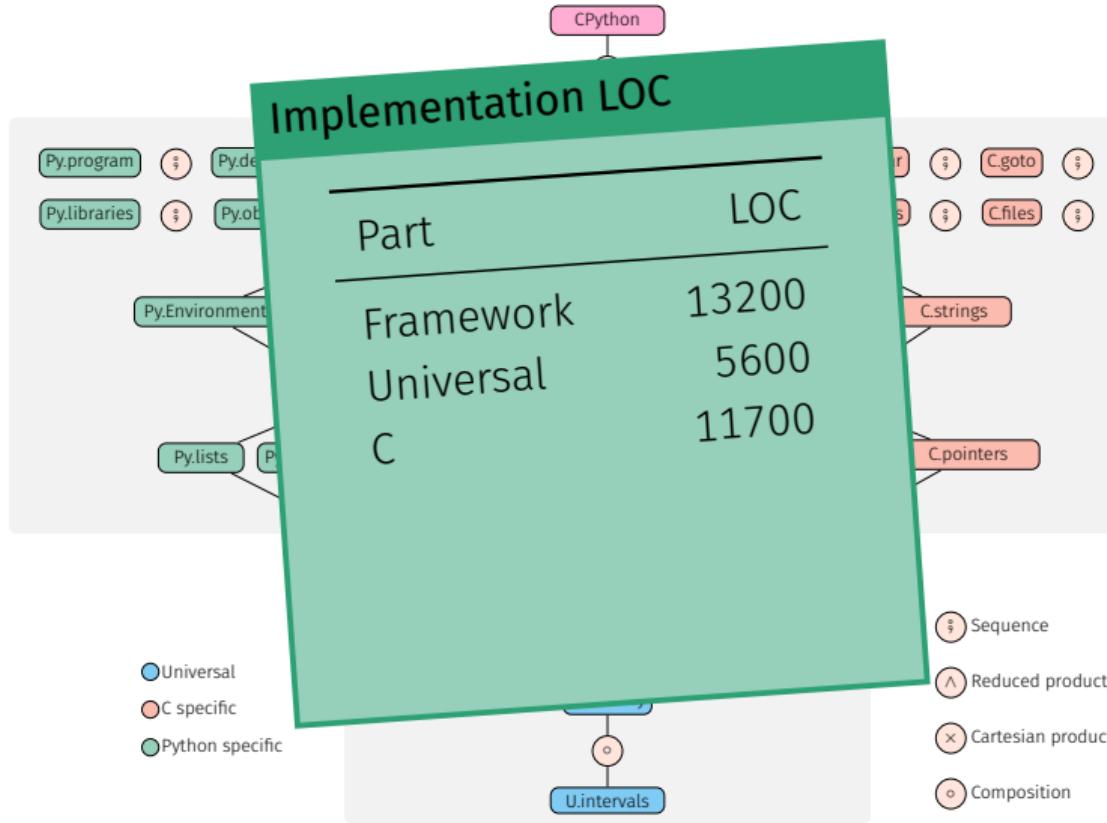
... to a multilanguage analysis!



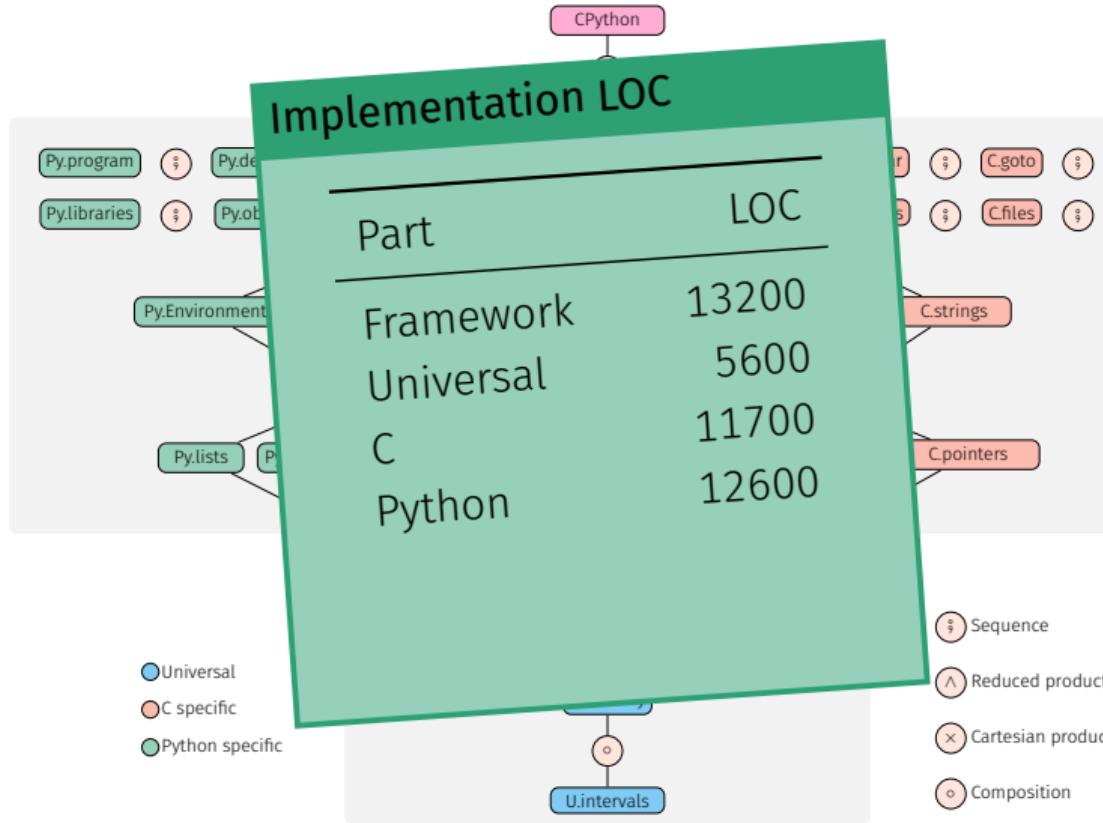
... to a multilanguage analysis!



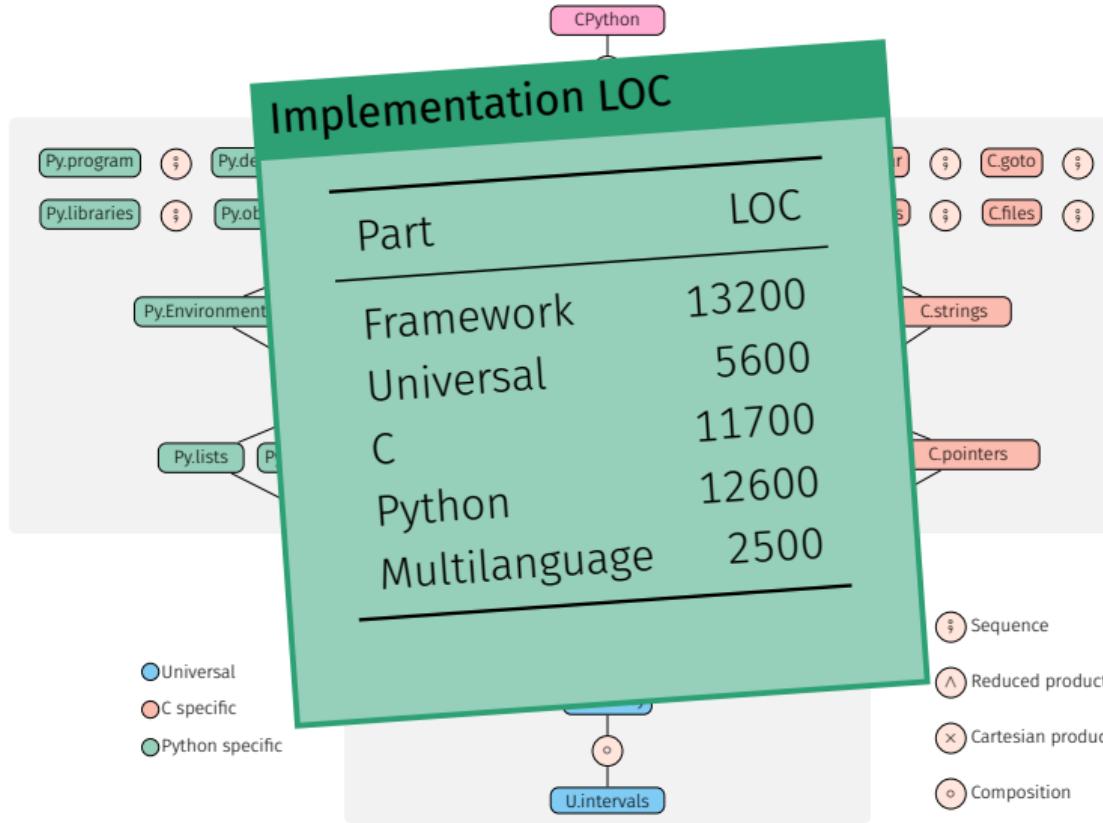
... to a multilanguage analysis!



... to a multilanguage analysis!



... to a multilanguage analysis!



# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
```

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()
```

Python

Attributes  
@CounterCls  $\mapsto \{get, incr\}$

Environment  
Counter  $\mapsto \{@CounterCls\}$   
@CounterCls.get  $\mapsto \{\text{&c function CounterGet}\}$   
@CounterCls.incr  $\mapsto \{\text{&c function CounterIncr}\}$

Universal

Heap (Recency)  
@CounterCls @CounterIncr  
@CounterGet

Intervals

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
```

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()
```

Python

Attributes  
@CounterCls  $\mapsto \{get, incr\}$

Environment  
Counter  $\mapsto \{@CounterCls\}$   
@CounterCls.get  $\mapsto \{\text{@c function CounterGet}\}$   
@CounterCls.incr  $\mapsto \{\text{@c function CounterIncr}\}$

Universal

Heap (Recency)  
@CounterCls @CounterIncr  
@CounterGet

Intervals

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
```

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()
```

Python

## Attributes

```
@CounterCls ↔ {get, incr}
```

## Environment

```
Counter ↔ {@CounterCls}
@CounterCls.get ↔
{@c function CounterGet}
@CounterCls.incr ↔
{@c function CounterIncr}
```

Universal

## Heap (Recency)

```
@CounterCls @CounterIncr
@CounterGet @I{CounterCls}
```

## Intervals

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
{@I{CounterCls},8,ptr} : {CounterCls}
```

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()
```

Python

## Attributes

```
@CounterCls ↔ {get, incr}
```

## Environment

```
Counter ↔ {@CounterCls}
@CounterCls.get ↔
{@c function CounterGet}
@CounterCls.incr ↔
{@c function CounterIncr}
```

Universal

Heap (Recency)  
@CounterCls @CounterIncr  
@CounterGet @I{CounterCls}

Intervals

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
{@I{CounterCls},8,ptr} : {CounterCls}
```

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()
```

Python

## Attributes

```
@CounterCls ↔ {get, incr}
```

## Environment

```
Counter ↔ {@CounterCls}
@CounterCls.get ↔
{@c function CounterGet}
@CounterCls.incr ↔
{@c function CounterIncr}
```

Universal

Heap (Recency)  
@CounterCls @CounterIncr  
@CounterGet @I{CounterCls}

## Intervals

```
<@I{CounterCls},16,s32> ↔ [0, 0]
```

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
{@I{CounterCls},8,ptr} : {CounterCls}
```

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()
```

Python

## Attributes

```
@CounterCls ↪ {get, incr}
@I{CounterCls} ↪ Ø
```

## Environment

```
Counter ↪ {@CounterCls}
@CounterCls.get ↪
{@c function CounterGet}
@CounterCls.incr ↪
{@c function CounterIncr}
```

Universal

Heap (Recency)  
@CounterCls @CounterIncr  
@CounterGet @I{CounterCls}

## Intervals

```
<@I{CounterCls},16,s32> ↪ [0, 0]
```

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
{@I{CounterCls},8,ptr} : {CounterCls}
```

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()
```

Python

## Attributes

```
@CounterCls ↪ {get, incr}
@I{CounterCls} ↪ Ø
```

## Environment

```
Counter ↪ {@CounterCls}
@CounterCls.get ↪
{@c function CounterGet}
@CounterCls.incr ↪
{@c function CounterIncr}
c ↪ {@I{CounterCls}}
```

Universal

## Heap (Recency)

```
@CounterCls @CounterIncr
@CounterGet @I{CounterCls}
```

## Intervals

```
{@I{CounterCls},16,s32} ↪ [0, 0]
```

# Analysis of the multilanguage 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->counter += i;
14    Py_RETURN_NONE;
15 }
16
17 static PyObject*
18 CounterGet(Counter *self)
```

C

## Pointers

```
<CounterCls,8,ptr> : {PyType_Type}
<CounterCls,232,ptr> : {Counter_methods}
{@I{CounterCls},8,ptr} : {CounterCls}
```

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()
```

Python

## Attributes

```
@CounterCls ↔ {get, incr}
@I{CounterCls} ↔ Ø
```

## Environment

```
Counter ↔ {@CounterCls}
@CounterCls.get ↔
{@c function CounterGet}
@CounterCls.incr ↔
{@c function CounterIncr}
c ↔ {@I{CounterCls}}
power ↔ {@I{int}}
```

Universal

## Heap (Recency)

```
@CounterCls @CounterIncr
@CounterGet @I{CounterCls}
```

## Intervals

```
{@I{CounterCls},16,s32} ↔ [0, 0]
power ↔ [0, 127]
```

## 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

### Theoretical frameworks

- ▶ Matthews and Findler<sup>13</sup> boundary functions as value conversions between two languages.
- ▶ Buro, Crole, and Mastroeni<sup>14</sup> generic framework for combining analyses of different languages.

---

<sup>13</sup> Matthews and Findler. "Operational semantics for multi-language programs". 2009.

<sup>14</sup> Buro, Crole, and Mastroeni. "On Multi-language Abstraction - Towards a Static Analysis of Multi-language Programs". SAS 2020.

## Related work (II)

### Around the Java Native Interface (JNI)

Static translation of some of C's effects, injected back into the Java analysis.

- ▶ Effects of C code on Java heap modelized using JVML<sup>15</sup>
- ▶ Type inference of Java objects in C code<sup>16</sup>
- ▶ Extraction of C callbacks to Java<sup>17</sup>
  
- ▶ Modular analyses
- ▶ No numeric information
- ▶ Missing C runtime errors

---

<sup>15</sup>Tan and Morrisett. "Ilea: inter-language analysis across Java and C". OOPSLA 2007.

<sup>16</sup>Furr and Foster. "Checking type safety of foreign function calls". 2008.

<sup>17</sup>Lee, Lee, and Ryu. "Broadening Horizons of Multilingual Static Analysis: Semantic Summary Extraction from C Code for JNI Program Analysis". ASE 2020.

## Conclusion

---

# Contribution: concrete semantics of Python

## Difficulties

- ▶ Size of the semantics
- ▶ CPython's source code

# Contribution: concrete semantics of Python

## Difficulties

- ▶ Size of the semantics
- ▶ CPython's source code

## Previous works

- ▶ Executable semantics of Python
- ▶ Handcrafted tests

# Contribution: concrete semantics of Python

## Difficulties

- ▶ Size of the semantics
- ▶ CPython's source code

## Previous works

- ▶ Executable semantics of Python
- ▶ Handcrafted tests

## Our results

- ▶ Semantics suitable for abstract interpretation
- ▶ Written and explained in the manuscript (70 cases)
- ▶ Backreferences to the source code
- ▶ Preliminary tests using CPython's suite

# Contribution: type & value analyses of Python

## Difficulties

- ▶ Dynamicity
- ▶ Dual type system
- ▶ Size of the semantics

---

<sup>18</sup> Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020.

<sup>19</sup> Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020.

# Contribution: type & value analyses of Python

## Difficulties

- ▶ Dynamicity
- ▶ Dual type system
- ▶ Size of the semantics

## Previous works

- ▶ JS: type and constant analysis
- ▶ Python: no scalability or support of dynamicity

---

<sup>18</sup> Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020.

<sup>19</sup> Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020.

# Contribution: type & value analyses of Python

## Difficulties

- ▶ Dynamicity
- ▶ Dual type system
- ▶ Size of the semantics

## Previous works

- ▶ JS: type and constant analysis
- ▶ Python: no scalability or support of dynamicity

## Our results

- ▶ Type analysis<sup>18</sup>,
- ▶ Numeric value analysis & new sensitivities for the recency abstraction<sup>19</sup>
- ▶ Relational value analysis with packing (manuscript)
- ▶ Scale to small, real-world benchmarks

<sup>18</sup> Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020.

<sup>19</sup> Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020.

## 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

## Our 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

## Some future works

### Executable concrete semantics

- ▶ Split soundness testing (CPython – concrete semantics – analyzer)
- ▶ Use skeletal semantics or interaction trees framework
- ▶ Conformance tests

## Some future works

### Executable concrete semantics

- ▶ Split soundness testing (CPython – concrete semantics – analyzer)
- ▶ Use skeletal semantics or interaction trees framework
- ▶ Conformance tests

### Dictionary abstractions

- ▶ Beyond key/value summarization
- ▶ Empirical study of dictionary use (use of non-string keys)

## Some future works

### Executable concrete semantics

- ▶ Split soundness testing (CPython – concrete semantics – analyzer)
- ▶ Use skeletal semantics or interaction trees framework
- ▶ Conformance tests

### Dictionary abstractions

- ▶ Beyond key/value summarization
- ▶ Empirical study of dictionary use (use of non-string keys)

### Multilanguage library analyses

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

# Static Type and Value Analysis by Abstract Interpretation of Python Programs with Native C Libraries

## Questions

xkcd.com/353

Raphaël Monat

MPRI lecture  
31 January 2021

