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.
Dynamic programming languages

Growing popularity

JavaScript #1, Python #2 on GitHub

1https://octoverse.github.com/#top-languages
Dynamic programming languages

Growing popularity
JavaScript #1, Python #2 on GitHub¹

New features
▶ Object orientation,
Dynamic programming languages

Growing popularity

JavaScript #1, Python #2 on GitHub\(^1\)

New features

► Object orientation,
► Dynamic typing,
## Dynamic programming languages

### Growing popularity

JavaScript #1, Python #2 on GitHub\(^1\)

### New features

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

\(^1\)https://octoverse.github.com/#top-languages
# Dynamic programming languages

## Growing popularity

JavaScript #1, Python #2 on GitHub\(^1\)

## New features

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

\(^1\)https://octoverse.github.com/#top-languages
### Dynamic programming languages

#### Growing popularity

JavaScript #1, Python #2 on GitHub\(^1\)

#### New features

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

\(^1\)https://octoverse.github.com/#top-languages
Well-established & industrialized analysis of static programming languages

- 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

State of the art

<table>
<thead>
<tr>
<th>Well-established &amp; industrialized analysis of static programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Java: Julia (2010)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Around JavaScript</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Why Python?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used a lot in</td>
</tr>
<tr>
<td>- Scientific computing</td>
</tr>
<tr>
<td>- Scripts and automation</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. A Taste of Python
3. The Mopsis Analysis Platform
4. Analyzing Python Programs
5. Analyzing Python Programs with C Libraries
6. Conclusion
What can we prove?

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

Highly polymorphic function. More context would help!
What can we prove?

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

Highly polymorphic function. More context would help!
What can we prove?

```
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

Average

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

Highly polymorphic function. More context would help!
What can we prove?

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

```
highly polymorphic function. More context would help!
```
### Average

<table>
<thead>
<tr>
<th>No.</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>def average(l):</code></td>
</tr>
<tr>
<td>2</td>
<td><code>m = 0</code></td>
</tr>
<tr>
<td>3</td>
<td><code>for i in range(len(l)):</code></td>
</tr>
<tr>
<td>4</td>
<td><code>m = m + l[i]</code></td>
</tr>
<tr>
<td>5</td>
<td><code>m = m // (i + 1)</code></td>
</tr>
<tr>
<td>6</td>
<td><code>return m</code></td>
</tr>
</tbody>
</table>

### Average

<table>
<thead>
<tr>
<th>No.</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><code>average([1,2,3])</code></td>
</tr>
<tr>
<td>8</td>
<td><code>try: average([])</code></td>
</tr>
<tr>
<td>9</td>
<td><code>except NameError: pass</code></td>
</tr>
<tr>
<td>10</td>
<td><code>average(range(2, 10, 3))</code></td>
</tr>
<tr>
<td>11</td>
<td><code>average({0: 3.14, 1: 2.78})</code></td>
</tr>
</tbody>
</table>

Highly polymorphic function. More context would help!

Detect raised exceptions interrupting the execution.
A Taste of Python
Python’s specificities

<table>
<thead>
<tr>
<th>No standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ CPython is the reference</td>
</tr>
<tr>
<td>→ manual inspection of the source code and handcrafted tests</td>
</tr>
</tbody>
</table>

```python
class Protected:
    def __init__(self, priv):
        self._priv = priv

    def __getattribute__(self, attr):
        if attr[0] == '_': raise AttributeError('protected')
        return object.__getattribute__(self, attr)

a = Protected(42)
a._priv  # AttributeError raised
```
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__

```python
class Protected:
    def __init__(self, priv):
        self._priv = priv
    def __getattr__(self, attr):
        if attr[0] == '_':
            raise AttributeError("protected")
        return object.__getattr__(self, attr)

a = Protected(42)
a._priv  # AttributeError raised
```

Protected attributes
Python’s specificities (II)

Dual type system

▶ Nominal (classes, MRO)

Fspath (from standard library)

class Path:
    def __fspath__(self): return 42

def fspath(p):
    if isinstance(p, (str, bytes)):
        return p
    elif hasattr(p, "__fspath__"):
        r = p.__fspath__()
        if isinstance(r, (str, bytes)):
            return r
        raise TypeError
    fspath("/dev" if random() else Path())

Python’s specificities (II)

Dual type system

- Nominal (classes, MRO)
- Structural (attributes)

---

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)

```python
class Path:
    def __fspath__(self): return 42

def fspath(p):
    if isinstance(p, (str, bytes)):
        return p
    elif hasattr(p, "__fspath__"):
        r = p.__fspath__()
        if isinstance(r, (str, bytes)):
            return r
        raise TypeError

fspath("/dev" if random() else Path())
```

## Previous works on Python 3

<table>
<thead>
<tr>
<th>Guth. “A formal semantics of Python 3.3”. 2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation within the K framework.</td>
</tr>
</tbody>
</table>

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

Moving to our own semantics

- Cost of understanding the code (vs CPython)
- Trust in the code (CPython's tests?)
- Insights of the papers
### Previous works on Python 3

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Year</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guth.</td>
<td>“A formal semantics of Python 3.3”</td>
<td>2013</td>
<td>Implementation within the K framework.</td>
</tr>
<tr>
<td>Politz et al.</td>
<td>“Python: The full monty”</td>
<td>OOPSLA</td>
<td>Complex desugaring into $\lambda$-\pi. May incur losses of precision in the abstract interpreter.</td>
</tr>
</tbody>
</table>
### Previous works on Python 3

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guth.</td>
<td>“A formal semantics of Python 3.3”</td>
<td>2013</td>
<td>Implementation within the K framework.</td>
</tr>
<tr>
<td>Politz et al.</td>
<td>“Python: The full monty”</td>
<td>OOPSLA</td>
<td>Complex desugaring into $\lambda_\pi$. May incur losses of precision in the abstract interpreter.</td>
</tr>
</tbody>
</table>
### Previous works on Python 3

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guth.</td>
<td>“A formal semantics of Python 3.3”</td>
<td>2013</td>
<td>Implementation within the K framework.</td>
</tr>
<tr>
<td>Politz et al.</td>
<td>“Python: The full monty”</td>
<td>OOPSLA 2013</td>
<td>Complex desugaring into $\lambda_\pi$. May incur losses of precision in the abstract interpreter.</td>
</tr>
</tbody>
</table>

### Different goal

These works focus on the concrete semantics. This is not our endgoal.
### Previous works on Python 3

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guth.</td>
<td>“A formal semantics of Python 3.3”. 2013</td>
<td>Implementation within the K framework.</td>
</tr>
<tr>
<td>Politz et al.</td>
<td>“Python: The full monty”</td>
<td>OOPSLA 2013</td>
</tr>
</tbody>
</table>

#### Moving to our own semantics
- Cost of understanding the code (vs CPython)
- Trust in the code (CPython’s tests?)
- Insights of the papers

#### Different goal
These works focus on the concrete semantics. This is not our endgoal.
Our approach

Interpreter-like semantics
Easily convertable to an abstract interpreter.

---

Our approach

Interpreter-like semantics
Easily convertible to an abstract interpreter.

Major extension of the work of Fromherz, Ouadjaout, and Miné²
- Separation between core and builtins
- 2.3× more cases (with statement, bidirectional generators, ...)
- Improved some cases (+, boolean casts of conditionals, data descriptors, ...)

Our approach

**Interpreter-like semantics**
Easily convertable to an abstract interpreter.

**Major extension of the work of Fromherz, Ouadjaout, and Miné**

- Separation between core and builtins
- $2.3 \times$ 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).

---

Example – attribute access

\[ E_{\text{cur}[x.s]}(\text{cur}, e, h) \overset{\text{def}}{=} \text{LOAD\_ATTR} \ \text{PyObject\_getattr} \ (\text{slot\_tp\_getattr\_hook}) \]

letb (\text{cur}, e, h), @x = E[x](\text{cur}, e, h) in

letb (\text{cur}, e, h), @c = E[\text{mro\_search}(\text{type}(\text{@x}), "\_\_getattr\_\_")](\text{cur}, e, h) in

letcases (f, e, h), @x.s = E[@c(\text{@x}, s)](\text{cur}, e, h) in

match f with

- \text{exn} @exc \text{ when } \text{instance}(@exc, \text{AttributeError}) \Rightarrow
  
  let (f, e, h), @d = E[\text{mro\_search}(\text{type}(\text{@x}), "\_\_getattr\_\_")](f, e, h) in
  if d \neq \bot \text{ then return } E[@d(\text{@x}, s)](\text{cur}, e, h)
  
  else return (f, e, h), \bot

- \_ \Rightarrow \text{return } (f, e, h), @x.s
Example – attribute access

\[
\mathcal{E}_{\text{cur}}[x.s](\text{cur}, e, h) \overset{\text{def}}{=} \text{LOAD\_ATTR} \ \text{PyObject\_GetAttr}\ (\text{slot\_tp\_getattr\_hook})
\]

\[
\text{letb}\ (\text{cur}, e, h), @x = \mathcal{E}[x](\text{cur}, e, h)\text{ in}
\]

\[
\mathcal{E}_{\text{cur}}[\text{object\._\_getattr\_\_}(\text{obj}, \text{name})](\text{cur}, e, h) \overset{\text{def}}{=} \text{tp\_field\_{\_PyObject\_GenericGetAttrWithDict}}
\]

\[
\text{letb}\ (\text{cur}, e, h), @o = \mathcal{E}[\text{obj}](\text{cur}, e, h)\text{ in}
\]

\[
\text{letb}\ (\text{cur}, e, h), @n = \mathcal{E}[\text{name}](\text{cur}, e, h)\text{ in}
\]

\[
\text{if } \neg\text{isinstance}(\overline{\text{o}_n}, \text{str})\text{ then return } S[\text{raise } \text{TypeError}](\text{cur}, e, h), \bot\text{ else}
\]

\[
\text{let str}(n) = \text{fst}\circ h(\overline{\text{O}_n})\text{ in}
\]

\[
\text{letcases}\ (f, e, h), @\text{descr} = \mathcal{E}[\text{mro\_search}(\text{type}(\overline{\text{O}_o}), n)](f, e, h)\text{ in}
\]

\[
\text{if } @\text{descr} \neq \bot\text{ then}
\]

\[
\text{if } \text{hasattr}(\text{type}(\overline{\text{O}_{\text{descr}}}), "\_\_\_get\_\_\_") \wedge
\]

\[
(\text{hasattr}(\text{type}(\overline{\text{O}_{\text{descr}}}), "\_\_\_set\_\_\_") \lor \text{hasattr}(\text{type}(\overline{\text{O}_{\text{descr}}}), "\_\_\_delete\_\_\_"))\text{ then}
\]

\[
\text{return } \mathcal{E}[\text{type}(\overline{\text{O}_{\text{descr}}}).\_\_\_get\_\_(\overline{\text{O}_{\text{descr}}}, @o, \text{type}(\overline{\text{O}_o}))](f, e, h)
\]
Example – attribute access

\[
E_{\text{cur}}[\times.s](\text{cur}, e, h) \overset{\text{def}}{=} \text{LOAD\_ATTR} \text{PyObject\_GetAttr} (\text{slot\_tp\_getattr\_hook})
\]

\[
\text{letb} (\text{cur}, e, h), @x = E[\times](\text{cur}, e, h) \text{ in}
\]

\[
E_{\text{cur}}[\text{object.__getattribute__}(\text{obj}, \text{name})](\text{cur}, e, h) \overset{\text{def}}{=}
\text{tp\_field \_PyObject\_GenericGetAttrWithDict}
\]

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

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

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

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

\[
\text{letb} (\text{cur}, e, h), @\text{meta} = E[\text{mro\_search}(\text{type}(@\text{typ}), @\text{name})](\text{cur}, e, h) \text{ in}
\]

\[
\text{if} @\text{meta} \neq \bot \text{ then}
\]

\[
\text{if} \hspace{0.5em} \text{hasattr(type}(@\text{meta}), "__get__") \wedge
\]

\[
(\text{hasattr(type}(@\text{meta}), "__set__") \lor \text{hasattr(type}(@\text{meta}), "__delete__")) \hspace{0.5em} \text{then}
\]

\[
E_{\text{cur}}[\text{type.__getattribute__}(\text{typ}, \text{name})](\text{cur}, e, h) \overset{\text{def}}{=}
\text{tp\_field \_type\_getattr}
\]
Example – binary operators

\[ a_1 = \text{eval} e_1; \ a_2 = \text{eval} e_2 \]

has_field($a_1$, \_\_add\_\_)?

\[ \text{has_field}(a_2, \_\_radd\_\_) \]

\&\& \text{type}(a_1) < \text{type}(a_2)?

\[ a_3 = \text{call } a_1 \text{'s } \_\_add\_\_ \text{ on } a_1, a_2 \]

\[ a_3 == \text{NotImplemented} \]

Result is $a_3$
Custom infix operators

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

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

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

Credits tomerfiliba.com/blog/Infix-Operators/
The Mopsa Analysis Platform
Averaging numbers

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m

l = [randint(0, 20)
     for i in range(randint(5, 10))]

m = average(l)
```
```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m

l = [randint(0, 20) for i in range(randint(5, 10))]
m = average(l)
```

### Searching for a loop invariant (l. 4)

#### Environment abstraction

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

### Proved safe?

- $m \div (i+1)$
- $l[i]$
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

\[ m \mapsto \mathbb{int} \quad i \mapsto \mathbb{int} \quad \text{els}(l) \mapsto \mathbb{int} \]

Proved safe?

▶ \( m \ // (i+1) \)

▶ \( l[i] \)
Workflow

Averaging numbers

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

Searching for a loop invariant (l. 4)

Stateless domains: list content,

Environment abstraction

\[
m \mapsto \mathbb{\#}_{\text{int}} \quad i \mapsto \mathbb{\#}_{\text{int}} \quad \text{els}(l) \mapsto \mathbb{\#}_{\text{int}}
\]

Numeric abstraction (intervals)

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

Proved safe?

- \( m \div (i+1) \)
- \( l[i] \)
Workflow

Averaging numbers

```
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
    m = m // (i + 1)
    return m
```

Stateless domains: list content, `list length`

Proved safe?

▶ `m // (i+1)`
▶ `l[i]`

Searching for a loop invariant (l. 4)

Environment abstraction

\[ m \mapsto @\text{int}^\# \quad i \mapsto @\text{int}^\# \quad \text{els}(l) \mapsto @\text{int}^\# \]

Numeric abstraction (intervals)

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

## Averaging numbers

```python
def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i]
        m = m // (i + 1)
    return m
```

```python
l = [randint(0, 20) for i in range(randint(5, 10))]
m = average(l)
```

### Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

### Environment abstraction

\[ m \mapsto \mathbb{Z}_+ \quad i \mapsto \mathbb{Z}_+ \quad \text{els}(l) \mapsto \mathbb{Z}_+ \]

### Numeric abstraction (polyhedra)

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

\[ 0 \leq i < \text{len}(l) \quad 5 \leq \text{len}(l) \leq 10 \]

---

Proved safe?

- \( m // (i+1) \)
- \( l[i] \)
Workflow

Averaging tasks

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

    def average(l):
        m = 0
        for i in range(len(l)):
            m = m + l[i].weight
        m = m // (i + 1)
        return m

l = [Task(randint(0, 20))
     for i in range(randint(5, 10))]

m = average(l)
```

Searching for a loop invariant (l. 4)
Stateless domains: list content, list length

### Environment abstraction

- $m \mapsto \int$  
- $i \mapsto \int$  
- $\text{els}(l) \mapsto \int$
- $\text{Task} \cdot \text{weight} \mapsto \int$

### Numeric abstraction (polyhedra)

- $m \in [0, +\infty)$
- $0 \leq i < \text{len}(l)$
- $5 \leq \text{len}(l) \leq 10$
- $0 \leq \text{Task} \cdot \text{weight} \leq 20$

### Attributes abstraction

- $\text{Task} \mapsto (\{\text{weight}\}, \emptyset)$

---

Proved safe?

- $m // (i+1)$
- $l[i].weight$
Workflow

Averaging tasks

class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

l = [Task(randint(0, 20)) for i in range(randint(5, 10))]
m = average(l)

Searching for a loop invariant (l. 4)
Stateless domains: list content, list length

Environment abstraction

\[ m \mapsto \overline{\text{Task}} \]
\[ i \mapsto \overline{\text{int}} \]
\[ m \mapsto \overline{\text{int}} \]
\[ \text{els}(l) \mapsto \overline{\text{Task}} \cdot \text{Task} \]
\[ \text{weight} \mapsto \overline{\text{int}} \]

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

Attributes abstraction

\[ \overline{\text{Task}} \mapsto (\{ \text{weight} \}, \emptyset) \]

Proved safe?

- \( m \div (i+1) \)
- \( l[i].weight \)
Overview

Modular Open Platform for Static Analysis

---

\[ \text{Py.list_len} \circ \text{U.numeric} \]
Overview

Modular Open Platform for Static Analysis

One AST to analyze them all
- Multilanguage support
- Expressiveness
- Reusability

\[ \text{Py.list_len} \circ \text{U.numeric} \]

\[ \text{Reduced product} \circ \text{Composition} \]

---

\(^3\)Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.
Overview

Modular Open Platform for Static Analysis

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability

- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability

---

Overview

**Modular Open Platform for Static Analysis**

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability

- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability

- **DAG of abstract domains**
  - Composition
  - Cooperation

---

Modular Open Platform for Static Analysis

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability

- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability

- DAG of abstract domains
  - Composition
  - Cooperation

---

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

Python.Desugar.Loops
◦ Rewrite and analyze recursively
◦ Optimize for some semantic cases

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

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

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

Python.Desugar.Loops
◦ Rewrite and analyze recursively
◦ Optimize for some semantic cases
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
  - Rewrite and analyze recursively
  - Optimize for some semantic cases

- Universal.Enumerate.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
◦ Rewrite and analyze recursively
◦ Optimize for some semantic cases

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

Universal.Iterators.Loops
Matches while(...){}...
Computes fixpoint using widening
Hooks as observers of the analysis

Idea

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

Hook signature

```ocaml
module type STATELESSHOOK = 
  sig
  val name : string
  val init : 'a ctx -> unit
  val on_before_exec : stmt -> ('a,'a) man -> 'a flow -> unit
  val on_after_exec : stmt -> ('a,'a) man -> 'a flow -> ('a, unit) cases -> unit
  val on_before_eval : expr -> ('a,'a) man -> 'a flow -> unit
  val on_after_eval : expr -> ('a,'a) man -> 'a flow -> ('a, expr) cases -> unit
  val on_finish : ('a,'a) man -> 'a flow -> unit
end
```
Example of hooks: Logs

Logs

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

```plaintext
+ 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 [ ]; int ]
      o E [ ]; 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]
    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

```python
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 $\Rightarrow$ analysis time $\not \propto$ 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

Unsupported constructs

- Recursive functions
- `eval`
- Finalizers
**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

```python
class A: pass
a = A()
a.x = 42
if *:
a.y = 3
r = a.x * a.y
```

After line 5:

After line 6, either ▶a must have attribute ▶ or an AttributeError has been raised ⇒ We need both an under and an over-approximation of the attributes.
Attribute abstraction

Non-deterministic attribute handling

```python
class A: pass

a = A()
a.x = 42
if *: a.y = 3
r = a.x * a.y
```

After line 5: a may have attribute 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

```python
class A:
    pass

a = A()
a.x = 42

if *:
a.y = 3

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.
Using an under and an over-approximation

\[ \text{ObjS}^\# = \{(u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{ \top \}, u \subseteq o \lor o = \top \} \]
Attribute abstraction (II)

Using an under and an over-approximation

\[ \text{ObjS}^\# = \{ (u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{ \top \}, u \subseteq o \lor o = \top \} \]

Concretization

\[ \gamma^\#_{\text{ObjS}} : \begin{cases} \text{ObjS}^\# & \rightarrow & \mathcal{P}(\mathcal{P}(\text{string})) \\ (u, \top) & \mapsto & \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \} \\ (u, o) & \mapsto & \{ s \in \mathcal{P}(\text{string}) \mid u \subseteq s \subseteq o \} \end{cases} \]

Example

\[ \gamma^\#_{\text{ObjS}}(\{ x \}, \{ x, y, z \}) = \{ \{ x \}, \{ x, y \}, \{ x, z \}, \{ x, y, z \} \} \]
Attribute abstraction (II)

Using an under and an over-approximation

\[
\text{ObjS}^\# = \left\{ (u, o) \mid u \in \mathcal{P}(\text{string}), o \in \mathcal{P}(\text{string}) \cup \{\top\}, u \subseteq o \lor o = \top \right\}
\]

Concretization

\[
\gamma^\#_{\text{ObjS}} : \begin{cases}
\text{ObjS}^\# & \rightarrow \mathcal{P}(\mathcal{P}(\text{string})) \\
(u, \top) & \mapsto \{s \in \mathcal{P}(\text{string}) \mid u \subseteq s\} \\
(u, o) & \mapsto \{s \in \mathcal{P}(\text{string}) \mid u \subseteq s \subseteq o\}
\end{cases}
\]

Example

\[
\gamma^\#_{\text{ObjS}}(\{x\}, \{x, y, z\}) = \{\{x\}, \{x, y\}, \{x, z\}, \{x, y, z\}\}
\]
Attribute abstraction (III)

Non-deterministic attribute handling

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

How to lift the attribute abstraction?
**Attribute abstraction (III)**

Non-deterministic attribute handling

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

How to lift the attribute abstraction? $\forall \rightarrow \text{ObjS}^?$
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? \( \mathcal{V} \rightarrow \text{ObjS}^\# \)?

After line 5, \( a \mapsto \{ x \}, \{ x, y \} \)?
Attribute abstraction (III)

Non-deterministic attribute handling

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

How to lift the attribute abstraction? $\forall \rightarrow \text{ObjS}^\#$?

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

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

$\text{Addr}^\# \rightarrow \text{ObjS}^\#$
Handling memory

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

A finite set of abstract addresses $\text{Addr}^\#$
N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses $\text{Addr}^\#$ (why finite?)
N.B: already mentionned by Xavier Rival in lesson 10.

A **finite** set of abstract addresses \texttt{Addr} (why finite?)

```python
1 class A: pass
2 3 n = random_int()
4 for i in range(n):
5   a = A()
6   a.val = i
```
N.B: already mentionned by Xavier Rival in lesson 10.

A finite set of abstract addresses $\text{Addr}^\#$ (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.
Handling memory | Allocation-site abstraction

### Allocation site abstraction

- \( \text{Addr}^\# = \text{Loc} \)
- **Weak updates** on abstract addresses

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

After line 3: \(@_3^\# \mapsto \emptyset, \emptyset\)

Weak update line 4:

\((@_3^\# \mapsto \{x\}, \{x\}) \sqcup^\# (@_3^\# \mapsto \emptyset, \emptyset) =\)
Handling memory | Allocation-site abstraction

Allocation site abstraction

- \( Addr^\# = Loc \)
- Weak updates on abstract addresses

Attribute addition

```python
class A: pass
a = A()
a.x = 42
```

After line 3: \( @^3 \mapsto \emptyset, \emptyset \)

Weak update line 4:

\[
(\langle @^3 \mapsto \{ x \}, \{ x \} \rangle \uplus^\# (\langle @^3 \mapsto \emptyset, \emptyset \rangle) = \langle @^3 \mapsto \emptyset, \{ x \} \rangle)
\]
Handling memory | Allocation-site abstraction

Allocation site abstraction

★ \texttt{Addr}^\# = \texttt{Loc}

★ \textbf{Weak updates} on abstract addresses

Attribute addition

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

After line 3: \(\odot^-3 \mapsto \emptyset, \emptyset\)

Weak update line 4:

\[
(\odot^-3 \mapsto \{ x \}, \{ x \}) \sqcup^-\# (\odot^-3 \mapsto \emptyset, \emptyset) = \odot^-3 \mapsto \emptyset, \{ x \}
\]

How to perform precise operations?
The recency abstraction

- Precise analysis of object initialization

---

Handling memory | Recency abstraction

The recency abstraction

- 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

---

The recency abstraction

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

---

The recency abstraction

- 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)

---

The recency abstraction

- 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)

---

### The recency abstraction

- 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

---

Handling memory | Recency abstraction (II)

Task creation

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

m = [1, 2]
l = [Task(i) for i in m]
l.append(Task(3))
```

Type analysis

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

m = [1, 2]
l = [Task(i) for i in m]
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.
Task creation

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

m = [1, 2]
l = [Task(i) for i in m]
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.
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

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

l = [Task(2), Task(1), Task(4), Task(5)]

Allocation:

\{ @#(Task, r) \mapsto \emptyset, \emptyset \}
Handling memory | Recency abstraction (III)

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

Initialization:

\[
\{\text{@#}(\text{Task}, r) \mapsto \{\text{weight}\}, \{\text{weight}\}\}
\]
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]

Allocation: @#(Task, r) ~@#(Task, o)

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

{ @#(Task, r) \mapsto \emptyset, \emptyset 

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

Analysis of l[0]. x = 3

What about l[3]. x = 3
### Handling memory | Recency abstraction (III)

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

**Initialization:**

\[
\begin{align*}
\text{@} # (\text{Task}, r) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\text{@} # (\text{Task}, r) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\text{@} # (\text{Task}, o) & \mapsto \{ \text{weight} \}, \{ \text{weight} \}
\end{align*}
\]
```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation: $\odot^\#(\text{Task}, r) \rightsquigarrow \odot^\#(\text{Task}, o)$

\[
\begin{align*}
\odot^\#(\text{Task}, r) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\odot^\#(\text{Task}, r) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\odot^\#(\text{Task}, o) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\odot^\#(\text{Task}, r) & \mapsto \emptyset, \emptyset \\
\odot^\#(\text{Task}, o) & \mapsto \{ \text{weight} \}, \{ \text{weight} \}
\end{align*}
\]
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]

Initialization:

{(Task, r) ↦ {weight}, {weight}}
{(Task, o) ↦ {weight}, {weight}}
{(Task, r) ↦ {weight}, {weight}}
{(Task, o) ↦ {weight}, {weight}}
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]

Allocation: @♯(Task, r) ↦→ @♯(Task, o)

{@♯(Task, r) ↦→ { weight }, { weight }}
{@♯(Task, r) ↦→ { weight }, { weight }}
{@♯(Task, o) ↦→ { weight }, { weight }}
{@♯(Task, o) ↦→ { weight }, { weight }}
{@♯(Task, r) ↦→ ∅, ∅}
{@♯(Task, o) ↦→ { weight }, { weight }}
```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

```latex
\begin{align*}
\mathcal{O}^\#(\text{Task, } r) & \mapsto \{ \text{weight} \}, \{ \text{weight} \} \\
\mathcal{O}^\#(\text{Task, } o) & \mapsto \{ \text{weight} \}, \{ \text{weight} \}
\end{align*}
```

Analysis of $l[0].x = 3$? What about $l[3].x = 3$?
How to perform a numerical analysis?
How to perform a numerical analysis?

Auxiliary variables for attributes: $\text{Addr}^\# \times \text{string} \subseteq \mathcal{V}$
Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation:
How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

$$\{(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2]\}$$
How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation: $@^\#(\text{Task}, r) \sim @^\#(\text{Task}, o)$
How to perform a numerical analysis?

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

```
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

\[
\begin{align*}
\forall x. \big( & \text{Task}, r \big) \cdot \text{weight} \mapsto [2, 2] \\
\forall x. \big( & \text{Task}, o \big) \cdot \text{weight} \mapsto [1, 1] \\
\forall x. \big( & \text{Task}, o \big) \cdot \text{weight} \mapsto [2, 2]
\end{align*}
\]
Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

Auxiliary variables for attributes: \( \text{Addr}^\# \times \text{string} \subseteq V \)

```python
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: \( @^\#(\text{Task}, r) \rightsquigarrow @^\#(\text{Task}, o) \)

\[
\begin{align*}
\{ @^\#(\text{Task}, r) \cdot \text{weight} \mapsto [2, 2] \\
@^\#(\text{Task}, o) \cdot \text{weight} \mapsto [1, 1] \}
\end{align*}
\]

\[
\begin{align*}
\{ @^\#(\text{Task}, r) \} \\
@^\#(\text{Task}, o) \cdot \text{weight} \mapsto [2, 2] \sqcup [1, 1] 
\end{align*}
\]
Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

\[
\begin{align*}
\#(\text{Task}, r) \cdot \text{weight} & \mapsto [2, 2] \\
\#(\text{Task}, o) \cdot \text{weight} & \mapsto [1, 1] \\
\#(\text{Task}, r) \cdot \text{weight} & \mapsto [2, 2] \\
\#(\text{Task}, o) \cdot \text{weight} & \mapsto [1, 2]
\end{align*}
\]
Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Allocation: @^#(Task, r) \leadsto @^#(Task, o)

\[
\begin{align*}
@^#(\text{Task}, r) \cdot \text{weight} & \mapsto [2, 2] \\
@^#(\text{Task}, r) \cdot \text{weight} & \mapsto [1, 1] \\
@^#(\text{Task}, o) \cdot \text{weight} & \mapsto [2, 2] \\
@^#(\text{Task}, r) \cdot \text{weight} & \mapsto [4, 4] \\
@^#(\text{Task}, o) \cdot \text{weight} & \mapsto [1, 2] \\
@^#(\text{Task}, r) \\
@^#(\text{Task}, o) \cdot \text{weight} & \mapsto [1, 2] \sqcup [4, 4]
\end{align*}
\]
Handling memory | Recency abstraction (IV)

How to perform a numerical analysis?

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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

l = [Task(2), Task(1), Task(4), Task(5)]
```

Initialization:

\[
\begin{align*}
\mathcal{A}^\#(\text{Task}, r) \cdot \text{weight} & \mapsto [2, 2] \\
\mathcal{A}^\#(\text{Task}, o) \cdot \text{weight} & \mapsto [1, 1] \\
\mathcal{A}^\#(\text{Task}, r) \cdot \text{weight} & \mapsto [2, 2] \\
\mathcal{A}^\#(\text{Task}, o) \cdot \text{weight} & \mapsto [4, 4] \\
\mathcal{A}^\#(\text{Task}, r) \cdot \text{weight} & \mapsto [1, 2] \\
\mathcal{A}^\#(\text{Task}, o) \cdot \text{weight} & \mapsto [5, 5] \\
\mathcal{A}^\#(\text{Task}, r) \cdot \text{weight} & \mapsto [1, 4]
\end{align*}
\]
## Container abstractions

### List abstraction

- Summarization of the content (auxiliary variable)
- Auxiliary length variable
Container abstractions

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.

\[
E_s \llbracket [e_1, \ldots, e_n]^{\text{loc}} \rrbracket \overset{\text{def}}{=} \\
\text{let } s, @ = E_s \llbracket \text{alloc(List loc)} \rrbracket s \text{ in} \\
\text{let } \text{contents} = \text{mk_var } @ \text{ ”contents” } \text{ in} \\
S_s \llbracket \text{contents}^{\text{weak}} = e_n \rrbracket \circ \ldots \circ S_s \llbracket \text{contents} = e_1 \rrbracket s, @
\]
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.

\[
\mathcal{E}\llbracket [e_1, \ldots, e_n]^{\text{loc}} \rrbracket s \overset{\text{def}}{=} \\
\text{let } s, \emptyset = \mathcal{E}\llbracket \text{alloc(List loc)} \rrbracket s \text{ in} \\
\text{let } \text{contents} = \text{mk_var} \emptyset "\text{contents}" \text{ in} \\
\mathcal{S}\llbracket \text{contents}^{\text{weak}} = e_n \rrbracket \circ \ldots \circ \mathcal{S}\llbracket \text{contents} = e_1 \rrbracket s, \emptyset
\]

Demo!
Types | Analysis

- Dynamicity:
  - Type inference first
- Flow-sensitive
- Context-sensitive
Types | Analysis

- Py.program
- Py.desugar
- Py.flow
- U.intraproc
- U.loops
- U.interproc
- Py.libraries
- Py.data_model
- Py.objects

- Py.environment
- Py.attributes
  - Py.lists
  - Py.tuples
  - Py.dicts
  - U.recency
  - U.strings

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

- Dynamicity:
  - type inference first
- Flow-sensitive
- Context-sensitive
Similar in essence to TAJS.\textsuperscript{6}

Dataflow analysis by Fritz and Hage.\textsuperscript{7}

Typpete: SMT-based type inference.\textsuperscript{8}

Pytype, type inference tool used by Google.\textsuperscript{9}

RPython: efficient compilation of a static subset of Python.\textsuperscript{10}

Value analysis by Fromherz et al.\textsuperscript{11}


\textsuperscript{7}Fritz and Hage. “Cost versus precision for approximate typing for Python”. PEPM 2017.

\textsuperscript{8}Hassan, Urban, Eilers, and Müller. “MaxSMT-Based Type Inference for Python 3”. CAV 2018.

\textsuperscript{9}Kramm et al. Pytype. 2019.


\textsuperscript{11}Fromherz, Ouadjaout, and Miné. “Static Value Analysis of Python Programs by Abstract Interpretation”. NFM 2018.
## Types | Experimental evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Mopsa</th>
<th>Fritz &amp; Hage</th>
<th>Pytype</th>
<th>Typpete</th>
<th>Fromherz et al.</th>
<th>RPython</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="bellman_ford.py" /></td>
<td>61</td>
<td>0.24s</td>
<td>1.4s</td>
<td>0.99s</td>
<td>1.4s</td>
<td>2.4m</td>
<td>7.1s</td>
</tr>
<tr>
<td><img src="image" alt="float.py" /></td>
<td>63</td>
<td>82ms</td>
<td>1.7s</td>
<td>0.92s</td>
<td>1.3s</td>
<td>0.84s</td>
<td>5.6s</td>
</tr>
<tr>
<td><img src="image" alt="coop_concat.py" /></td>
<td>64</td>
<td>43ms</td>
<td>1.8s</td>
<td>0.81s</td>
<td>1.3s</td>
<td>20ms</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="crafting.py" /></td>
<td>132</td>
<td>0.41s</td>
<td>1.6s</td>
<td>0.97</td>
<td>1.7s</td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="nbody.py" /></td>
<td>157</td>
<td>0.80s</td>
<td>1.7s</td>
<td>1.3s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="chaos.py" /></td>
<td>324</td>
<td>2.3s</td>
<td>13s</td>
<td>11s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="scimark.py" /></td>
<td>416</td>
<td>0.55s</td>
<td>8.5s</td>
<td>4.4s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="richards.py" /></td>
<td>426</td>
<td>5.0s</td>
<td>38s</td>
<td>2.4s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="unpack_seq.py" /></td>
<td>458</td>
<td>4.2s</td>
<td>1.1s</td>
<td>7.4s</td>
<td>2.7s</td>
<td>14s</td>
<td>7.8s</td>
</tr>
<tr>
<td><img src="image" alt="go.py" /></td>
<td>461</td>
<td>15s</td>
<td>8.5s</td>
<td>3.4s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="hexiom.py" /></td>
<td>674</td>
<td>22s</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="regex_v8.py" /></td>
<td>1792</td>
<td>15s</td>
<td>4.9s</td>
<td></td>
<td>1.7m</td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="processInput.py" /></td>
<td>1417</td>
<td>4.8s</td>
<td>2.4s</td>
<td>11s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="choose.py" /></td>
<td>2562</td>
<td>46s</td>
<td>1.7s</td>
<td>15s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ![Bug](image) unsupported by the analyzer (crash)
- ![CLOCK](image) timeout (after 1h)

Smashed exceptions: KeyError, IndexError, ValueError
## Experimental evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Mopsa</th>
<th>Fritz &amp; Hage</th>
<th>Pytype</th>
<th>Typpete</th>
<th>Fromherz et al.</th>
<th>RPython</th>
</tr>
</thead>
<tbody>
<tr>
<td>bellman_ford.py</td>
<td>61</td>
<td>0.24s</td>
<td>1.4s</td>
<td>0.99s</td>
<td>1.4s</td>
<td>2.4m</td>
<td>7.1s</td>
</tr>
<tr>
<td>float.py</td>
<td>63</td>
<td>82ms</td>
<td>1.7s</td>
<td>0.92s</td>
<td>1.3s</td>
<td>0.84s</td>
<td>5.6s</td>
</tr>
<tr>
<td>coop_concat.py</td>
<td>64</td>
<td>43ms</td>
<td>1.8s</td>
<td>0.81s</td>
<td>1.3s</td>
<td>20ms</td>
<td></td>
</tr>
<tr>
<td>crafting.py</td>
<td>132</td>
<td>0.41s</td>
<td>1.6s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nbody.py</td>
<td>157</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chaos.py</td>
<td>324</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scimark.py</td>
<td>416</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>richards.py</td>
<td>426</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7.8s</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14s</td>
</tr>
<tr>
<td>go.py</td>
<td>461</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td></td>
<td></td>
<td>14.2s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regex_v8.py</td>
<td>1792</td>
<td>15s</td>
<td>4.9s</td>
<td>1.2s</td>
<td></td>
<td>1.7m</td>
<td></td>
</tr>
<tr>
<td>processInput.py</td>
<td>1417</td>
<td>4.8s</td>
<td>7.0s</td>
<td>2.4s</td>
<td>11s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>46s</td>
<td>17.0s</td>
<td>1.7s</td>
<td>15s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”.

35
Thanks to Mopsa, switching from types to values is straightforward!

Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”.
Thanks to Mopsa, switching from types to values is straightforward!

Monat, Ouadjaout, and Miné. “Value and allocation sensitivity in static Python analyses”. 
Types \( \rightsquigarrow \) values | Comparing the analyses

Averaging tasks

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

l = []
for i in range(randint(5, 10)):
    l.append(Task(randint(0, 20)))
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

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

    def average(l):
        m = 0
        for i in range(len(l)):
            m = m + l[i].weight
        m = m // (i + 1)
        return m

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

**Type analysis**

- **ValueError** (l. 3)
- **IndexError** (l. 9)
Types \( \leadsto \) values | Comparing the analyses

Averaging tasks

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

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

Type analysis

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

class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

l = []
for i in range(randint(5, 10)):
    l.append(Task(randint(0, 20)))

m = average(l)

Type analysis

- **ValueError** (l. 3)
- **IndexError** (l. 9)
- **ZeroDivisionError** (l. 10)
- **NameError** (l. 10)
Types ⇝ values | Comparing the analyses

### Averaging tasks

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

l = []
for i in range(randint(5, 10)):
    l.append(Task(randint(0, 20)))
    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 \(\sim\) values | Comparing the analyses

### Averaging tasks

```python
class Task:
    def __init__(self, weight):
        if weight < 0: raise ValueError
        self.weight = weight

def average(l):
    m = 0
    for i in range(len(l)):
        m = m + l[i].weight
    m = m // (i + 1)
    return m

l = []
for i in range(randint(5, 10)):
    l.append(Task(randint(0, 20)))
    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 ⇝ values | Comparing the analyses (II)

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Type Analysis</th>
<th>Non-relational Value Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Mem.</td>
</tr>
<tr>
<td>nbody.py</td>
<td>157</td>
<td>1.5s</td>
<td>3MB</td>
</tr>
<tr>
<td>scimark.py</td>
<td>416</td>
<td>1.4s</td>
<td>12MB</td>
</tr>
<tr>
<td>richards.py</td>
<td>426</td>
<td>13s</td>
<td>112MB</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td>8.3s</td>
<td>7MB</td>
</tr>
<tr>
<td>go.py</td>
<td>461</td>
<td>27s</td>
<td>345MB</td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td>1.1m</td>
<td>525MB</td>
</tr>
<tr>
<td>regex_v8.py</td>
<td>1792</td>
<td>23s</td>
<td>18MB</td>
</tr>
<tr>
<td>processInput.py</td>
<td>1417</td>
<td>10s</td>
<td>64MB</td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>1.1m</td>
<td>1.6GB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>9294</td>
<td>4.0m</td>
<td>2.8GB</td>
</tr>
</tbody>
</table>
## Types $\rightsquigarrow$ values | Comparing the analyses (II)

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Type Analysis</th>
<th>Non-relational Value Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbody.py</td>
<td>157</td>
<td>1.5s</td>
<td>1.5s</td>
</tr>
<tr>
<td>scimark.py</td>
<td>416</td>
<td>1.4s</td>
<td>1.4s</td>
</tr>
<tr>
<td>richards.py</td>
<td>426</td>
<td>13s</td>
<td>22</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td>8.3s</td>
<td>22</td>
</tr>
<tr>
<td>go.py</td>
<td>461</td>
<td>27s</td>
<td>22</td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td>1.1m</td>
<td>22</td>
</tr>
<tr>
<td>regex_v8.py</td>
<td>1792</td>
<td>23s</td>
<td>2053</td>
</tr>
<tr>
<td>processInput.py</td>
<td>1417</td>
<td>10s</td>
<td>7</td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>1.1m</td>
<td>2.9m</td>
</tr>
<tr>
<td>Total</td>
<td>9294</td>
<td>4.0m</td>
<td>2.8GB</td>
</tr>
</tbody>
</table>

### Conclusion

The non-relational value analysis

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

The non-relational value analysis

- does not remove false type alarms
- significantly reduces index errors
- is $\simeq 3 \times$ costlier
## Types ⟷ values | Comparing the analyses (II)

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Time</th>
<th>Mem.</th>
<th>Exceptions detected</th>
<th>Type Index Key</th>
<th>Non-relational Value Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Mem.</td>
<td>Exceptions detected</td>
<td>Type Index Key</td>
<td>Time</td>
</tr>
<tr>
<td>nbody.py</td>
<td>157</td>
<td>1.5s</td>
<td>3MB</td>
<td>0</td>
<td>1</td>
<td>1.5s</td>
</tr>
<tr>
<td>scimark.py</td>
<td>416</td>
<td>1.4s</td>
<td>12MB</td>
<td>1</td>
<td>0</td>
<td>1.4s</td>
</tr>
<tr>
<td>richards.py</td>
<td>426</td>
<td>13s</td>
<td>112MB</td>
<td>0</td>
<td>1</td>
<td>13s</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td>8.3s</td>
<td>7MB</td>
<td>0</td>
<td>0</td>
<td>8.3s</td>
</tr>
<tr>
<td>go.py</td>
<td>461</td>
<td>27s</td>
<td>6MB</td>
<td>0</td>
<td>0</td>
<td>27s</td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td>1.1m</td>
<td>525MB</td>
<td>0</td>
<td>3</td>
<td>1.1m</td>
</tr>
<tr>
<td>regex_v8.py</td>
<td>1792</td>
<td>23s</td>
<td>18MB</td>
<td>2053</td>
<td>0</td>
<td>23s</td>
</tr>
<tr>
<td>processInput.py</td>
<td>1417</td>
<td>10s</td>
<td>64MB</td>
<td>7</td>
<td>7</td>
<td>10s</td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>1.1m</td>
<td>1.6GB</td>
<td>12</td>
<td>12</td>
<td>1.1m</td>
</tr>
<tr>
<td>Total</td>
<td>9294</td>
<td>4.0m</td>
<td>2.8GB</td>
<td>59</td>
<td>2214</td>
<td>12m</td>
</tr>
</tbody>
</table>

### Conclusion

The non-relational value analysis

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

### Heurisitic packing and relational analyses

- Static packing, using function’s scope
- Rules out all 145 alarms of `regex_v8.py` (1792 LOC) at 2.5× cost
Selectivity of the non-relational value analysis

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

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

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
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)

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 – 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
A combined static analysis of C/Python

- Targeting C extensions using the CPython API

---

### A combined static analysis of C/Python

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

---

A combined static analysis of C/Python

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

---

A combined static analysis of C/Python

- 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

---

Combining C and Python – first look

A combined static analysis of C/Python\textsuperscript{12}

\begin{itemize}
\item Targeting C extensions using the CPython API
\item To detect runtime errors (in C, Python, and the “glue”)
\item Observations
  \begin{itemize}
  \item allocated objects are \underline{shared} in the memory
  \item but each language has \underline{different} abstractions
  \end{itemize}
\end{itemize}

\textsuperscript{12}Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.
Combining C and Python – first look

A combined static analysis of C/Python\textsuperscript{12}

- 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

\textsuperscript{12}Monat, Ouadjaout, and Miné. “A Multilanguage Static Analysis of Python Programs with Native C Extensions”. SAS 2021.
### Combining C and Python – example

#### counter.c

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

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

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

#### count.py

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

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```
Combining C and Python – example

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

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

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

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

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

► $\text{power} \leq 30 \Rightarrow r = 2^\text{power}$
Combining C and Python – example

**counter.c**

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

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

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

**count.py**

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

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

- **power ≤ 30 ⇒ r = 2^power**

- **32 ≤ power ≤ 64:** OverflowError: signed integer is greater than maximum

- **power ≥ 64:** OverflowError: Python int too large to convert to C long
Combining C and Python – example

### counter.c

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

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

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

### count.py

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

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

- ▶ \( \text{power} \leq 30 \Rightarrow r = 2^{\text{power}} \)
- ▶ \( \text{power} = 31 \Rightarrow r = -2^{31} \)
- ▶ \( 32 \leq \text{power} \leq 64 \): 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 ⇒ 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 ➞ missed errors
- Only types
How to analyze multilanguage programs?

Type annotations

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

- No raised exceptions $\rightarrow$ 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 → 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

```python
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)
<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewrite into Python code</td>
</tr>
<tr>
<td>Drawbacks of the current approaches</td>
</tr>
</tbody>
</table>
How to analyze multilanguage programs?

<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewrite into Python code</td>
</tr>
<tr>
<td>Drawbacks of the current approaches</td>
</tr>
<tr>
<td>▶ Not the real code</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewrite into Python code</td>
</tr>
</tbody>
</table>

#### 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
- 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
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?

<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
</table>

#### Rewrite into Python code

<table>
<thead>
<tr>
<th>Drawbacks of the current approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Not the real code</td>
</tr>
<tr>
<td>▶ Not automatic: manual conversion</td>
</tr>
<tr>
<td>▶ Not sound: some effects are not taken into account</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Analyze both the C and Python sources</td>
</tr>
<tr>
<td>▶ Switch from one language to the other just as the program does</td>
</tr>
<tr>
<td>▶ Reuse previous analyses of C and Python</td>
</tr>
</tbody>
</table>
How to analyze multilanguage programs?

<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Rewrite into Python code</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Drawbacks of the current approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Not the real code</td>
</tr>
<tr>
<td>▶ Not automatic: manual conversion</td>
</tr>
<tr>
<td>▶ Not sound: some effects are not taken into account</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Analyze both the C and Python sources</td>
</tr>
<tr>
<td>▶ Switch from one language to the other just as the program does</td>
</tr>
<tr>
<td>▶ Reuse previous analyses of C and Python</td>
</tr>
<tr>
<td>▶ Detect runtime errors in Python, in C, and at the boundary</td>
</tr>
</tbody>
</table>
```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

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

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

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

c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

static PyObject* CounterIncr(Counter *self, PyObject *args)
{
    int i = 1;
    if(!PyArg_ParseTuple(args, '|i', &i))
        return NULL;
    self->count += i;
    Py_RETURN_NONE;
}

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

from counter import Counter
from random import randrange

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

<table>
<thead>
<tr>
<th>Concrete definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Builds upon the Python and C semantics</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Concrete definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>► Builds upon the Python and C semantics</td>
</tr>
<tr>
<td>► Defines the API: calls between languages, value conversions</td>
</tr>
<tr>
<td>► Shared heap, with disjoint, complementary <strong>views</strong></td>
</tr>
<tr>
<td>► <strong>Boundary functions</strong> when objects switch views for the first time</td>
</tr>
</tbody>
</table>
### 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)
- Manual modelization from CPython’s source code
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)
### 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)
- 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!

### Implementation LOC

<table>
<thead>
<tr>
<th>Part</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>13200</td>
</tr>
<tr>
<td>Universal</td>
<td>5600</td>
</tr>
<tr>
<td>C</td>
<td>11700</td>
</tr>
<tr>
<td>Python</td>
<td>12600</td>
</tr>
</tbody>
</table>
... to a multilanguage analysis!

Implementation LOC

<table>
<thead>
<tr>
<th>Part</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>13200</td>
</tr>
<tr>
<td>Universal</td>
<td>5600</td>
</tr>
<tr>
<td>C</td>
<td>11700</td>
</tr>
<tr>
<td>Python</td>
<td>12600</td>
</tr>
<tr>
<td>Multilanguage</td>
<td>2500</td>
</tr>
</tbody>
</table>
Analysis of the multilanguage example

### counter.c
```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

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

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

### count.py
```python
from counter import Counter
from random import randrange

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

---

**Universal**

- **Heap (Recency)**
  - @CounterCls ↦→ {CounterGet, incr}
- **Universal Pointers**
  - ⟨CounterCls, 8, ptr⟩ : {PyType_Type}
  - ⟨CounterCls, 232, ptr⟩ : {Counter_methods}
- **Intervals**
  - ⟨@CounterCls, 8, ptr⟩ : {CounterCls}
  - ⟨CounterCls.get⟩ ↦→ {c function CounterGet}
  - ⟨CounterCls.incr⟩ ↦→ {c function CounterIncr}

---

**Python**

- **Attributes**
  - @CounterCls ↦→ {get, incr}
- **Environment**
  - Counter ↦→ {@CounterCls}
  - @CounterCls.get ↦→
    - {c function CounterGet}
  - @CounterCls.incr ↦→
    - {c function CounterIncr}
### Analysis of the multilanguage example

<table>
<thead>
<tr>
<th>counter.c</th>
<th>count.py</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef struct {</code></td>
<td><code>from counter import Counter</code></td>
</tr>
<tr>
<td><code>  PyObject_HEAD;</code></td>
<td><code>from random import randrange</code></td>
</tr>
<tr>
<td><code>  int count;</code></td>
<td><code>c = Counter()</code></td>
</tr>
<tr>
<td><code>} Counter;</code></td>
<td><code>power = randrange(128)</code></td>
</tr>
<tr>
<td><code>static PyObject*</code></td>
<td><code>c.incr(2**power-1)</code></td>
</tr>
<tr>
<td><code>CounterIncr(Counter *self, PyObject *args)</code></td>
<td><code>c.incr()</code></td>
</tr>
<tr>
<td><code>{</code></td>
<td><code>r = c.get()</code></td>
</tr>
<tr>
<td><code>  int i = 1;</code></td>
<td></td>
</tr>
<tr>
<td>`  if(!PyArg_ParseTuple(args, &quot;</td>
<td>i&quot;, &amp;i))`</td>
</tr>
<tr>
<td><code>    return NULL;</code></td>
<td></td>
</tr>
<tr>
<td><code>  self-&gt;counter += i;</code></td>
<td></td>
</tr>
<tr>
<td><code>  Py_RETURN_NONE;</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td><code>static PyObject*</code></td>
<td></td>
</tr>
<tr>
<td><code>CounterGet(Counter *self)</code></td>
<td></td>
</tr>
<tr>
<td><code>{</code></td>
<td></td>
</tr>
<tr>
<td><code>  return Py_BuildValue(&quot;i&quot;, self-&gt;counter);</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
</tbody>
</table>

### Python

**Attributes**
- `@CounterCls` $\mapsto \{\text{get, incr}\}$

**Environment**
- `Counter` $\mapsto \{@\text{CounterCls}\}$
- `@\text{CounterCls}.get` $\mapsto \{c \text{ function CounterGet}\}$
- `@\text{CounterCls}.incr` $\mapsto \{c \text{ function CounterIncr}\}$
## Analysis of the multilanguage example

### counter.c

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

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

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

### count.py

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

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

### Python

- **Attributes**
  - @CounterCls \( \mapsto \{\text{get}, \text{incr}\} \)
  
- **Environment**
  - Counter \( \mapsto \{\text{@CounterCls}\} \)
  - @CounterCls.get \( \mapsto \{\text{c function CounterGet}\} \)
  - @CounterCls.incr \( \mapsto \{\text{c function CounterIncr}\} \)

### Universal

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

- **Intervals**
  - \( \langle \text{@CounterCls}, 8, \text{ptr} \rangle : \{\text{PyType_Type}\} \)
  - \( \langle \text{CounterCls}, 232, \text{ptr} \rangle : \{\text{Counter_methods}\} \)

### Pointers

- \( \langle \text{CounterCls}, 8, \text{ptr} \rangle : \{\text{PyType_Type}\} \)

---

\( E^\#_{\text{Ccall}}[\text{PyType_GenericNew(CounterCls, NULL, NULL)}] \sigma^# \)
Analysis of the multilanguage example

counter.c

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

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

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

count.py

```python
from counter import Counter
from random import randrange
c = Counter()
power = randrange(128)
c.incr(2**power-1)
c.incr()
r = c.get()
```

**C**

- **Points**: <CounterCls,8,ptr> : {PyType_Type}
  <CounterCls,232,ptr> : {Counter_methods}
  <@I{CounterCls},8,ptr> : {CounterCls}

**Python**

- **Attributes**: @CounterCls \(\rightarrow\) {get, incr}
- **Environment**: @CounterCls \(\leftrightarrow\) {CounterCls}
  @CounterCls.get \(\leftrightarrow\) {c function CounterGet}
  @CounterCls.incr \(\leftrightarrow\) {c function CounterIncr}

**Universal**

- **Heap**: (Recency)
  - @CounterCls @CounterIncr

- **Intervals**
### Analysis of the multilanguage example

**counter.c**

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

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

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

**count.py**

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

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

**Diagram**

- **Python**
  - Attributes: `{get, incr}`
  - Environment: `{CounterCls}`
  - Functions: `{@CounterCls::get, incr}`

- **Universal**
  - Heap (Recency):
    - `@CounterCls::CounterIncr`
    - `@CounterCls::CounterGet`
  - Intervals: `{@CounterCls,16,s32} \rightarrow [0, 0]`

- **C**
  - Pointers:
    - `{CounterCls,8,ptr}`: `{PyType_Type}
    - `{CounterCls,232,ptr}`: `{Counter_methods}
    - `{@CounterCls,8,ptr}`: `{CounterCls}`
Analysis of the multilanguage example

counter.c

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

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

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

count.py

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

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

Python

Attributes
- `@CounterCls` $\mapsto$ `{get, incr}
- `@I{CounterCls}` $\mapsto$ $\emptyset$

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

Universal

Heap (Recency)
- `@CounterCls` $\mapsto$ `@CounterIncr`
- `@CounterGet` $\mapsto$ `@I{CounterCls}

Intervals
- `{@I{CounterCls},16,s32}` $\mapsto$ `[0, 0]`

Pointers
- `{CounterCls,8,ptr}` : `{PyType_Type}
- `{CounterCls,232,ptr}` : `{Counter_methods}
- `{@I{CounterCls},8,ptr}` : `{CounterCls}`
### Analysis of the multilanguage example

**counter.c**

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

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

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

**count.py**

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

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

**Universal Pointer Table**

| CounterCls,8,ptr | {PyType_Type} |
| CounterCls,232,ptr | {Counter_methods} |
| @I{CounterCls} | {get, incr} |
| @CounterCls | ∅ |

**Heap Table (Recency)**

- @CounterCls ↦→ @CounterIncr
- @CounterCls ↦→ @CounterGet

**Intervals**

- ⟨@I{CounterCls},16,s32⟩ ↦→ [0, 0]

**Python Environment**

- Counter ↦→ {@CounterCls}
- @CounterCls.get ↦→ {@c function CounterGet}
- @CounterCls.incr ↦→ {@c function CounterIncr}
- c ↦→ {@I{CounterCls}}
Analysis of the multilanguage example

### counter.c
```c
typedef struct {
    PyObject_HEAD;
    int count;
} Counter;

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

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

### count.py
```python
from counter import Counter
from random import randrange

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

**Python**
- **Attributes**
  - `@CounterCls` ↦→ `{get, incr}
  - `@I{CounterCls}` ↦→ `∅`
- **Environment**
  - `Counter` ↦→ `{@CounterCls}
  - `@CounterCls` ↦→ `{@c function CounterGet}
  - `@CounterCls`.incr ↦→ `{@c function CounterIncr}
  - `c` ↦→ `{@I{CounterCls}}
  - `power` ↦→ `{@I{int}}

**Universal**
- **Pointers**
  - `(CounterCls,8,ptr) : {PyType_Type}
  - `(CounterCls,232,ptr) : {Counter_methods}
  - `(I{CounterCls},8,ptr) : {CounterCls}
- **Heap (Recency)**
  - `@CounterCls` `@CounterIncr`  `@CounterGet` `@I{CounterCls}`
- **Intervals**
  - `(I{CounterCls},16,s32) ↦→ [0, 0]
  - `power` ↦→ `[0, 127]`
**Benchmarks**

**Corpus selection**

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

| Library     | |C| |Py| Tests | safe C checks | total C checks | total C checks | 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        |
Related work

Theoretical frameworks

- Matthews and Findler\textsuperscript{13} boundary functions as value conversions between two languages.
- Buro, Crole, and Mastroeni\textsuperscript{14} generic framework for combining analyses of different languages.

\textsuperscript{13}Matthews and Findler. “Operational semantics for multi-language programs”. 2009.
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\(^{15}\)
- Type inference of Java objects in C code\(^{16}\)
- Extraction of C callbacks to Java\(^{17}\)

- Modular analyses
- No numeric information
- Missing C runtime errors


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

Previous works

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

Our results

- Type analysis
- Numeric value analysis & new sensitivities for the recency abstraction
- Relational value analysis with packing (manuscript)
- Scale to small, real-world benchmarks

18 Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 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

---

18 Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 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\(^{18}\),
- **Numeric value analysis** & new sensitivities for the recency abstraction\(^{19}\)
- **Relational value analysis** with packing (manuscript)
- Scale to small, real-world benchmarks

---

\(^{18}\) Monat, Ouadjaout, and Miné. “Static Type Analysis by Abstract Interpretation of Python Programs”. ECOOP 2020.

\(^{19}\) 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