Formal methods for real-world systems: study of two cases

Raphaël Monat

LIP, ENS de Lyon
28 April 2022
Introduction
Introduction

Curriculum

ENS de Lyon


PhD student, LIP6 ATER, LIP6

Hongseok Yang @ Oxford, 3 months

Eva Darulova @ MPI-SWS, 5 months

Research field: formal methods
⇒ Improve confidence in software.
Means
▶ Theory: formal definition and reasoning over systems
▶ Practice: software development
Introduction

Curriculum

- ENS de Lyon
- PhD student, LIP6
- ATER, LIP6


Hongseok Yang @ Oxford, 3 months
Eva Darulova @ MPI-SWS, 5 months

Research field: formal methods

⇒ Improve confidence in software.
Introduction

Curriculum

<table>
<thead>
<tr>
<th>Year</th>
<th>Institution/Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td>ENS de Lyon</td>
</tr>
<tr>
<td>2015</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>PhD student, LIP6</td>
</tr>
<tr>
<td>2019</td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td>ATER, LIP6</td>
</tr>
<tr>
<td>2022</td>
<td></td>
</tr>
</tbody>
</table>

- Hongseok Yang @ Oxford, 3 months
- Eva Darulova @ MPI-SWS, 5 months

Research field: formal methods

⇒ Improve confidence in software.

Means

- Theory: formal definition and reasoning over systems
- Practice: software development
## Personal methodology

Constant back and forth between theory and practice

1. Find interesting bugs, properties or systems to study (GitHub, ...)
2. Theoretical study and solution
3. Implementation and experimental validation (on 1)

| Studied systems | Implementation of the French tax code | \( \Rightarrow \) | compiler, modernization | • | Denis Merigoux (Inria Prosecco) | • | Jonathan Protzenko (MSR) | • | Abdelraouf Ouadjaout (LIP6) | • | Antoine Miné (LIP6) | ▶ | Python programs using C libraries | \( \Rightarrow \) | static analysis | • | |
Introduction

Personal methodology
Constant back and forth between theory and practice
1. Find interesting bugs, properties or systems to study (GitHub, ...)
2. Theoretical study and solution
3. Implementation and experimental validation (on 1)

Studied systems
- Python programs using C libraries ⇛ static analysis
  - Abdelraouf Ouadjaout (LIP6)
  - Antoine Miné (LIP6)
Introduction

Personal methodology
Constant back and forth between theory and practice
1 Find interesting bugs, properties or systems to study (GitHub, ...)
2 Theoretical study and solution
3 Implementation and experimental validation (on 1)

Studied systems
▶ Python programs using C libraries $\leadsto$ static analysis
  • Abdelraouf Ouadjaout (LIP6)
  • Antoine Miné (LIP6)
▶ Implementation of the French tax code $\leadsto$ compiler, modernization
  • Denis Merigoux (Inria Prosecco)
  • Jonathan Protzenko (MSR)
Software verification

$S[\text{prog}]$

Cheap approach: test \text{prog}. Some bugs may go undetected!

Would there be a way to automatically prove programs correct?
Software verification

Bad states

$S[\text{prog}]$
Software verification

Cheap approach: test $prog$.

Some bugs may go undetected!

Would there be a way to automatically prove programs correct?
Software verification

Cheap approach: test \textit{prog}.

Bad states

$S[\text{prog}]$
Software verification

Cheap approach: test $prog$.

Bad states
Software verification

Cheap approach: test \textit{prog}.

Bad states
Software verification

Cheap approach: test $prog$. Some bugs may go undetected!
Software verification

Cheap approach: test \textit{prog}. Some bugs may go undetected!
Software verification

Cheap approach: test \textit{prog}.
Some bugs may go undetected!

Would there be a way to automatically prove programs correct?
An impossibility theorem

All reported errors are true errors.

Sound: All true errors are reported.
An impossibility theorem

All reported errors are true errors

Complete

Sound

All true errors are reported
An impossibility theorem

Guaranteed Termination

Complete

Sound

All reported errors are true errors

All true errors are reported
An impossibility theorem

Guaranteed Termination

Complete

Sound

∅

Rice’s theorem

All reported errors are true errors

All true errors are reported

4
An impossibility theorem

All reported errors are true errors

Guaranteed Termination
Complete
All true errors are reported

Sound

Rice’s theorem

Abstract Interpretation

∅
Abstract interpretation – the big picture

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Program proved safe

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Abstract interpretation – the big picture

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Abstract interpretation – the big picture

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Abstract interpretation – the big picture

False alarm (Abstraction too coarse)

\[ S[ prog ] \]
\[ \mathcal{D} \text{ (concrete)} \]
\[ S^\#[ prog ] \]
\[ \mathcal{D}^\# \text{ (abstract)} \]

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Abstract interpretation – the big picture

$D$ (concrete)

$\mathcal{S}_[prog]$  

Bad states

$\mathcal{D}^\#$ (abstract)

$\mathcal{S}^\#[prog]$  

Bad states

Unsound analysis (shouldn’t happen)

P. Cousot and R. Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. POPL 1977
Static analysis of critical software

Successfully applied to critical C software

- Astrée: Airbus A340, A380
- Frama-C: nuclear power plants
### Successfully applied to critical C software

- Astrée: Airbus A340, A380
- Frama-C: nuclear power plants

### How to analyze general software?

- New constructs: dynamic allocation, parallelism, ...
- New languages: Python, ...
- “Generic” static analyzers
## Dynamic programming languages

### Growing popularity

JavaScript #1, Python #2 on GitHub

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

Growing popularity
JavaScript #1, Python #2 on GitHub¹

New features
▶ Object orientation,

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

## Growing popularity

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

## New features

- Object orientation,
- Dynamic typing,

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

Growing popularity

JavaScript #1, Python #2 on GitHub¹

New features

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

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

Growing popularity
JavaScript #1, Python #2 on GitHub

New features
► Object orientation,
► Dynamic typing,
► Dynamic object structure,
► Introspection operators,

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

Growing popularity
JavaScript #1, Python #2 on GitHub¹

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

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

1. Introduction
2. A Taste of Python
3. Analyzing Python Programs
4. Analyzing Python Programs with C Libraries
5. A Modern Compiler for the French Tax Code
6. Conclusion
A Taste of Python
Python’s specificities

No standard

- CPython is the reference
  → manual inspection of the source code and handcrafted tests
Python’s specificities

No standard

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

Operator redefinition

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

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

a = Protected(42)
a._priv # AttributeError raised
```
Python’s specificities (II)

Dual type system

▶ Nominal (classes, MRO)

Exceptions

▶ `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())
```

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())
```

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())
```

Example Semantics – binary operators

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

\[ \text{has_field}(a_1, \text{__add__})? \]

\[ \text{has_field}(a_2, \text{__radd__}) \&\& \text{type}(a_1) \neq \text{type}(a_2)? \]

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

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

\[ \text{Result is } a_3 \]

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

\[ a_3 \]

\[ \text{Type Error} \]
class Infix(object):
    def __init__(self, func):
        self.func = func
    def __or__(self, other):
        return self.func(other)
    def __ror__(self, other):
        return Infix(lambda x: self.func(other, x))

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

@Infix
def padd(x, y):
    print(f"{x} + {y} = {x + y}"")
    return x + y
c = 2 |padd| 3
Analyzing Python Programs
Goal
Detect runtime errors: uncaught raised exceptions
### Analysis | Overview

#### Goal
Detect runtime errors: uncaught raised exceptions

#### Supported constructs

Our analysis supports:

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

**Unsupported constructs**
- Recursive functions
- eval
- Finalizers
**Goal**
Detect runtime errors: uncaught raised exceptions

**Supported constructs**
Our analysis supports:

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

**Unsupported constructs**
- Recursive functions
- `eval`
- Finalizers
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)
```
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))]
```

Searching for a loop invariant (l. 4)

Environment abstraction

\[ m \mapsto \sharp \int \quad i \mapsto \sharp \int \]

Proved safe?

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

```python
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,

Environment abstraction

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

Proved safe?

- \( m \text{ // (i+1)} \)
- \( m + l[i] \)
Averaging numbers

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

Environment abstraction

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

Numeric abstraction (intervals)

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

```python
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**

Searching for a loop invariant (l. 4)

Proved safe?

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

Environment abstraction

\[
\begin{align*}
  m & \mapsto \@\text{int}\# \\
  i & \mapsto \@\text{int}\# \\
  \text{els}(l) & \mapsto \@\text{int}\#
\end{align*}
\]

Numeric abstraction (intervals)

\[
\begin{align*}
  m & \in [0, +\infty) \\
  \text{els}(l) & \in [0, 20] \\
  \text{len}(l) & \in [5, 10] \\
  i & \in [0, 10]
\end{align*}
\]
Analysis | Domains required

Averaging numbers

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

Proved safe?

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

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

Environment abstraction

\[
\begin{align*}
m & \mapsto \mathbb{int}^\# \\
i & \mapsto \mathbb{int}^\# \\
els(l) & \mapsto \mathbb{int}^\#
\end{align*}
\]

Numeric abstraction (polyhedra)

\[
\begin{align*}
m & \in [0, +\infty) \\
els(l) & \in [0, 20] \\
0 & \leq i < \text{len}(l) \\
5 & \leq \text{len}(l) \leq 10
\end{align*}
\]
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
```

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

### Environment abstraction

- $m \mapsto \mathbb{int}^\#$
- $i \mapsto \mathbb{int}^\#$
- $\text{els}(l) \mapsto \mathbb{Task}^\#$

### Numeric abstraction (polyhedra)

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

### Attributes abstraction

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

Proved safe?

- $m // (i+1)$
- $m + l[i].\text{weight}$
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)
```

Proved safe?

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

Environment abstraction

\[ m \rightarrow \text{Task} \]
\[ i \rightarrow \text{int} \]
\[ l \rightarrow \text{list} \]
\[ \sum_{i=0}^{\text{len}(l)-1} \text{Task} \cdot \text{weight} \leq 20 \]

Attributes abstraction

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

Conclusion

- Different domains depending on the precision
- Use of auxiliary variables (underlined)
Modular Open Platform for Static Analysis

Modular Open Platform for Static Analysis

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

Mopsa | Overview

Modular Open Platform for Static Analysis²

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

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

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

---

Mopsa | 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

---

Mopsa | Dynamic, semantic iterators with delegation

<table>
<thead>
<tr>
<th>Universal.Iterators.Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matches <code>while(...) {...}</code></td>
</tr>
<tr>
<td>Computes fixpoint using widening</td>
</tr>
</tbody>
</table>
for(init; cond; incr) body

Universal.Iterators.Loops
Matches while(...) {...}
Computes fixpoint using widening
Mopsa | Dynamic, semantic iterators with delegation

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

```
C.iterators.loops
```

- Rewrite and analyze recursively

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

- Universal.Iterators.Loops
  - Matches `while(...) {...}`
  - Computes fixpoint using widening
Mopsa | Dynamic, semantic iterators with delegation

for(init; cond; incr) body

C.iterators.loops
Rewrite and analyze recursively

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

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

Matches while(...) {...}
Computes fixpoint using widening
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
Definition of the Value 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.intervals

U.strings

Universal

Python specific

Sequence

Cartesian product

Composition
Comparison of the type and value 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)**

R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value analyses

R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value 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)

R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value analyses

<table>
<thead>
<tr>
<th>Averaging tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 class Task:</td>
</tr>
<tr>
<td>2 def <strong>init</strong>(self, weight):</td>
</tr>
<tr>
<td>3 if weight &lt; 0: raise ValueError</td>
</tr>
<tr>
<td>4 self.weight = weight</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6 def average(l):</td>
</tr>
<tr>
<td>7 m = 0</td>
</tr>
<tr>
<td>8 for i in range(len(l)):</td>
</tr>
<tr>
<td>9 m = m + l[i].weight</td>
</tr>
<tr>
<td>10 m = m // (i + 1)</td>
</tr>
<tr>
<td>11 return m</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13 l = []</td>
</tr>
<tr>
<td>14 for i in range(randint(5, 10)):</td>
</tr>
<tr>
<td>15 l.append(Task(randint(0, 20)))</td>
</tr>
<tr>
<td>16 m = average(l)</td>
</tr>
</tbody>
</table>

Type analysis

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

R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value 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)**

R. Monat, A. Ouadjajout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value analyses

```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!

R. Monat, A. Ouadjaout, and A. Miné. “Value and allocation sensitivity in static Python analyses”. SOAP@PLDI 2020
Comparison of the type and value 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></td>
<td></td>
<td>Type</td>
<td>Index</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>
Comparison of the type and value 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></td>
<td></td>
<td></td>
<td></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>3.7GB</td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>1.1m</td>
<td>1.6GB</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 $\approx 3 \times$ costlier
Comparison of the type and value analyses (II)

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Type Analysis</th>
<th></th>
<th>Non-relational Value Analysis</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>Mem.</td>
<td>Exceptions detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Type</td>
<td>Index</td>
<td>Key</td>
</tr>
<tr>
<td>nbody.py</td>
<td>157</td>
<td>1.5s</td>
<td>3MB</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>scimark.py</td>
<td>416</td>
<td>1.4s</td>
<td>12MB</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>richards.py</td>
<td>426</td>
<td>13s</td>
<td>112MB</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td>8.3s</td>
<td>7MB</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>go.py</td>
<td>461</td>
<td>27s</td>
<td>345MB</td>
<td>33</td>
<td>20</td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td>1.1m</td>
<td>525MB</td>
<td>0</td>
<td>46</td>
</tr>
<tr>
<td>regex_v8.py</td>
<td>1792</td>
<td>23s</td>
<td>18MB</td>
<td>0</td>
<td>2053</td>
</tr>
<tr>
<td>processInput.py</td>
<td>1417</td>
<td>10s</td>
<td>64MB</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>choose.py</td>
<td>2562</td>
<td>1.1m</td>
<td>1.6GB</td>
<td>12</td>
<td>22</td>
</tr>
<tr>
<td>Total</td>
<td>9294</td>
<td>4.0m</td>
<td>2.8GB</td>
<td>59</td>
<td>228</td>
</tr>
</tbody>
</table>

**Conclusion**

The non-relational value analysis
- does not remove false type alarms
- significantly reduces index errors
- is \( \approx 3 \times \) costlier

**Heuristic packing and relational analyses**

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

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

- Different runtime-errors (exceptions in Python)

- Garbage collection
Combining C and Python – motivation

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

- To bring better performance (numpy)
- To provide library bindings (pygit2)
Combining C and Python – motivation

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

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

Pitfalls
Combining C and Python – motivation

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

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

Pitfalls

- Different values (arbitrary-precision integers in Python, bounded in C)
Combining C and Python – motivation

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

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

Pitfalls

► Different values (arbitrary-precision integers in Python, bounded in C)
► Different runtime-errors (exceptions in Python)
Combining C and Python – motivation

<table>
<thead>
<tr>
<th>One in five of the top 200 Python libraries contains C code</th>
</tr>
</thead>
<tbody>
<tr>
<td>► To bring better performance (numpy)</td>
</tr>
<tr>
<td>► To provide library bindings (pygit2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pitfalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>► Different values (arbitrary-precision integers in Python, bounded in C)</td>
</tr>
<tr>
<td>► Different runtime-errors (exceptions in Python)</td>
</tr>
<tr>
<td>► Garbage collection</td>
</tr>
</tbody>
</table>
### Combining C and Python – example

<table>
<thead>
<tr>
<th>counter.c</th>
<th>count.py</th>
</tr>
</thead>
</table>
| 1 typedef struct {
  2     PyObject_HEAD;
  3     int count;
  4 } Counter;
  5 |
| 6 static PyObject*
  7     CounterIncr(Counter *self, PyObject *args)
  8 {
  9     int i = 1;
 10    if(!PyArg_ParseTuple(args, "|i", &i))
 11       return NULL;
 12    self->count += i;
 13    Py_RETURN_NONE;
 14 }
| 1  from counter import Counter
| 1  from random import randrange
| 2  |
| 3  c = Counter()
| 4  power = randrange(128)
| 5  c.incr(2**power-1)
| 6  c.incr()
| 7  r = c.get() |
### 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()
```

- \( \text{power} \leq 30 \Rightarrow r = 2^\text{power} \)
- \( \text{power} = 31 \Rightarrow r = -2^{31} \)
- \( 32 \leq \text{power} \leq 64: \text{OverflowError: signed integer is greater than maximum} \)
- \( \text{power} \geq 64: \text{OverflowError: Python int too large to convert to C long} \)
Combining C and Python – example

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

- **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

```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()
```

- **power ≤ 30 → r = 2^{\text{power}}**
- **power = 31 → r = -2^{31}**
- **32 ≤ power ≤ 64**: OverflowError: signed integer is greater than maximum
- **power ≥ 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

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
### Type annotations

```python
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

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

▶ No integer wrap-around in Python
▶ Some effects can’t be written in pure Python (e.g., read-only attributes)
## How to analyze multilanguage programs?

<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>

### 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
## How to analyze multilanguage programs?

### Type annotations

### Rewrite into Python code

### Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion

### 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?

<table>
<thead>
<tr>
<th>Type annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewrite into Python code</td>
</tr>
<tr>
<td><strong>Drawbacks of the current approaches</strong></td>
</tr>
<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>
### 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>
</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><strong>Drawbacks of the current approaches</strong></td>
</tr>
<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>
<tr>
<td><strong>Our approach</strong></td>
</tr>
<tr>
<td>► Analyze both the C and Python sources</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>
</tbody>
</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>
</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>
<tr>
<td>- Not automatic: manual conversion</td>
</tr>
<tr>
<td>- Not sound: some effects are not taken into account</td>
</tr>
<tr>
<td>Our approach</td>
</tr>
<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>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Drawbacks of the current approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<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>--------------</td>
</tr>
<tr>
<td></td>
</tr>
<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
#define 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, 
{ 
    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); 
}

counter.c

from counter import Counter
from random import randrange

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

Difficulty: shared memory

► Each language may change the memory state, and has a different view of it
► Synchronization? We could perform a full state translation, but
  • the cost would be high in the analysis
  • some abstractions can be shared between Python and C
# High-level idea

## Difficulty: shared memory

- Each language may change the memory state, and has a different view of it
- Synchronization? We could perform a full state translation, but
  - the cost would be high in the analysis
  - some abstractions can be shared between Python and C

## State separation $\rightsquigarrow$ reduced synchronization

- Observation: structures are directly dereferenceable by one language only
- Switch to other language otherwise ($c$.incr() $\rightsquigarrow$ `self->count += 1`)

  Additional hypothesis: C accesses to Python objects through the API

- Synchronization: only when objects change language for the first time
- Mopsa supports shared abstractions
## Multilanguage semantics

### Concrete definition

- Builds upon the Python and C semantics
Multilanguage semantics

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
- Boundary functions handling the reduced synchronization

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
- Boundary functions handling the reduced synchronization

## 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
- **Boundary functions** handling the reduced synchronization

### Limitations

- Garbage collection not handled
Multilanguage semantics

Concrete definition

- Builds upon the Python and C semantics
- Defines the API: calls between languages, value conversions
- Boundary functions handling the reduced synchronization

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
- **Boundary functions** handling the reduced synchronization

### 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...
From distinct Python and C analyses... to a multilanguage analysis!
From distinct Python and C analyses... 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>
Benchmarks

Corpus selection

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

<table>
<thead>
<tr>
<th>Library</th>
<th>C + Py. Loc</th>
<th>Tests</th>
<th>#/test</th>
<th># proved checks</th>
<th>%</th>
<th># checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>noise</td>
<td>1397</td>
<td>15/15</td>
<td>1.2s</td>
<td>99.7%</td>
<td>(6690)</td>
<td></td>
</tr>
<tr>
<td>cdistance</td>
<td>2345</td>
<td>28/28</td>
<td>4.1s</td>
<td>98.0%</td>
<td>(13716)</td>
<td></td>
</tr>
<tr>
<td>llist</td>
<td>4515</td>
<td>167/194</td>
<td>1.5s</td>
<td>98.8%</td>
<td>(36255)</td>
<td></td>
</tr>
<tr>
<td>ahocorasick</td>
<td>4877</td>
<td>46/92</td>
<td>1.2s</td>
<td>96.7%</td>
<td>(6722)</td>
<td></td>
</tr>
<tr>
<td>levenshtein</td>
<td>5798</td>
<td>17/17</td>
<td>5.3s</td>
<td>84.6%</td>
<td>(4825)</td>
<td></td>
</tr>
<tr>
<td>bitarray</td>
<td>5841</td>
<td>159/216</td>
<td>1.6s</td>
<td>94.9%</td>
<td>(25566)</td>
<td></td>
</tr>
</tbody>
</table>
Contributions around the static analysis of Python programs

<table>
<thead>
<tr>
<th>Python’s semantics</th>
<th>$\sim S_{py} \left[ \cdot \right]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Reverse-engineering CPython (160kLoc C)</td>
<td></td>
</tr>
<tr>
<td>▶ Backlinks to source code (auditability)</td>
<td></td>
</tr>
<tr>
<td>▶ On-paper formalization ($\approx 44$ pages)</td>
<td></td>
</tr>
</tbody>
</table>
Contributions around the static analysis of Python programs

<table>
<thead>
<tr>
<th>Python’s semantics</th>
<th>$\text{Spy} \Rightarrow S_{\text{py}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Reverse-engineering CPython (160kLoc C)</td>
<td></td>
</tr>
<tr>
<td>- Backlinks to source code (auditability)</td>
<td></td>
</tr>
<tr>
<td>- On-paper formalization ($\approx 44$ pages)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Python’s analyses</th>
<th>$S_{\text{py}} \Rightarrow S_{# \text{py}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Type and value analyses</td>
<td></td>
</tr>
<tr>
<td>- Combining numerous abstractions</td>
<td></td>
</tr>
<tr>
<td>- Analysis of real programs</td>
<td></td>
</tr>
</tbody>
</table>
Contributions around the static analysis of Python programs

### Python’s semantics

- Reverse-engineering CPython (160kLoc C)
- Backlinks to source code (auditability)
- On-paper formalization (≈ 44 pages)

### Python’s analyses

- Type and value analyses
- Combining numerous abstractions
- Analysis of real programs

### Multilanguage, Python/C analysis

- First real multilanguage analysis
- Reuses off-the-shelf Python and C analyses
- Analysis of real-world libraries
 Contributions around the static analysis of Python programs

<table>
<thead>
<tr>
<th>Python’s semantics</th>
<th>Python’s analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Reverse-engineering CPython (160kLoc C)</td>
<td>▶ Type and value analyses</td>
</tr>
<tr>
<td>▶ Backlinks to source code (auditability)</td>
<td>▶ Combining numerous abstractions</td>
</tr>
<tr>
<td>▶ On-paper formalization (≈ 44 pages)</td>
<td>▶ Analysis of real programs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multilanguage, Python/C analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ First real multilanguage analysis</td>
</tr>
<tr>
<td>▶ Reuses off-the-shelf Python and C analyses</td>
</tr>
<tr>
<td>▶ Analysis of real-world libraries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation into Mopsa</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Open-source analyzer for C and Python</td>
</tr>
<tr>
<td>▶ Factors abstractions between languages</td>
</tr>
<tr>
<td>▶ Sharing between abstractions</td>
</tr>
</tbody>
</table>
A Modern Compiler for the French Tax Code
Research field: formal methods

⇒ Improve confidence in software.
Research field: formal methods
⇒ Improve confidence in software.

Personal methodology
Constant back and forth between theory and practice
1. Find interesting bugs, properties or systems to study (GitHub, ...)
2. Theoretical study and solution
3. Implementation and experimental validation (on 1)
### French income tax

- 38M households, 75Md€ of income
- Made public in April 2016: \(\approx 92\text{kLoc M, custom language}

⚠️ Computation not reproducible in 2019
### French income tax

- 38M households, 75Md€ of income
- Made public in April 2016: ≃ 92kLoc M, custom language
- Computation not reproducible in 2019

### Trusting the computation?

- Reproducibility of the computation?
- Accurate simulation of tax reforms?
- Compliance with the law, acting as specification?
Variable declaration

IRNETBIS : calculee primrest = 0 : "IRNET avant bidouille du 8ZI" ;
8ZI : "Impot net apres depart a l'etranger (non residents)" ;
Example M code

Variable declaration
IRNETBIS : calculee primrest = 0 : "IRNET avant bidouille du 8ZI" ;
8ZI : "Impot net apres depart a l'etranger (non residents)" ;

Computation rule
rule 221220:
application : iliad ;
IRNETBIS = max(0, IRNETTER -
    PIR * positif(SEUIL_12 - IRNETTER + PIR)
    * positif(SEUIL_12 - PIR)
    * positif_ou_nul(IRNETTER - SEUIL_12));
The core of M: arithmetic expressions assigned to variables.
The core of M: arithmetic expressions assigned to variables.

M quirks

- Static-size arrays (size defined at declaration)
M, briefly

The core of M: **arithmetic expressions** assigned to variables.

M quirks

- Static-size arrays (size defined at declaration)
- Small, unrollable loops
M, briefly

The core of M: arithmetic expressions assigned to variables.

**M quirks**

- Static-size arrays (size defined at declaration)
- Small, unrollable loops
- Use of floating-point numbers, booleans are zero and one
The core of M: **arithmetic expressions** assigned to variables.

**M quirks**

- Static-size arrays (size defined at declaration)
- Small, unrollable loops
- Use of floating-point numbers, booleans are zero and one
- `undef` value
A formal semantics for M

We reverse-engineered the semantics:

- At first, using the online simulator\(^3\)
- Later, using the private tests DGFiP sent us (August 7, 2019)

Fun facts:
- \(f + \text{undef} = f\), \(f ÷ 0 = 0\), \(x \mid x \mid + 1 = \text{undef}\), \(x \mid -1 = 0\)

\(^3\)https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm
We reverse-engineered the semantics:

- At first, using the online simulator\(^3\)
- Later, using the private tests DGFiP sent us (August 7, 2019)

⇒ a \(\mu M\) kernel, its semantics formalized in the Coq proof assistant.

\(^3\)https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm
A formal semantics for M

We reverse-engineered the semantics:

- At first, using the online simulator
- Later, using the private tests DGFiP sent us (August 7, 2019)

⇒ a $\mu$M kernel, its semantics formalized in the Coq proof assistant.

The `undef` value

- Used for: default inputs, runtime errors & missing cases in inline conditionals
- Fun facts: $f + \text{undef} = f, f \div 0 = 0, x[|x| + 1] = \text{undef}, x[-1] = 0...$

DGFiP’s legacy architecture

After 9 months of negotiations, we’re in!

“rules”
M files → “rules”
C files

Shared state

“inter”
C files

“calculette”
Shared library

DGFiP’s internal compiler

GCC

M files

C files

35kLoc of C to bypass M’s lack of functions.

Security concerns ⇝ no publication

How to extract the logic of the code?

DSLs to the rescue! Introducing M++

▶ High-level, no mutable state under the hood

▶ Tailored for the needs of the “inter” files and DGFiP devs

⇒ 6,000 lines of “inter” C code

100 lines of M++
DGFiP’s legacy architecture

After 9 months of negotiations, we’re in!

“rules” M files

“rules” C files

Shared state

“calculette” Shared library

“inter” C files

DGFiP’s internal compiler

GCC

“How to extract the logic of the code?

DSLs to the rescue! Introducing M ++

▶ High-level, no mutable state under the hood
▶ Tailored for the needs of the “inter” files and DGFiP devs

⇒ 35kLoc of C to bypass M’s lack of functions.

“inter” files
DGFiP’s legacy architecture

After 9 months of negotiations, we’re in!

“rules” M files

“rules” C files

Shared state

“inter” C files

“calculette” Shared library

DGFiP’s internal compiler

GCC

“inter” files

35kLoc of C to bypass M’s lack of functions.

Security concerns ⇞ no publication

How to extract the logic of the code?

DSLs to the rescue! Introducing M++

▶ High-level, no mutable state under the hood
▶ Tailored for the needs of the “inter” files and DGFiP devs

⇒ 6,000 lines of “inter” C code
⇒ 100 lines of M++
DGFiP’s legacy architecture

After 9 months of negotiations, we’re in!

“rules” M files

“rules” C files

Shared state

“inter” C files

“calculette” Shared library

DGFiP’s internal compiler

GCC

“inter” files

35kLoc of C to bypass M’s lack of functions.

Security concerns ➞ no publication

How to extract the logic of the code?

DSLs to the rescue! Introducing M++

► High-level, no mutable state under the hood
► Tailored for the needs of the “inter” files and DGFiP devs
► 6,000 lines of “inter” C code ➞ 100 lines of M++
MLANG’s architecture

MLANG: written in OCaml, 10k lines of code
https://github.com/MLanguage/mlang
MLANG’s architecture

MLANG: written in OCaml, 10k lines of code
https://github.com/MLanguage/mlang

sources.m → M AST → M IR → BIR → Python
source.mpp → M++ AST → M++ IR → OIR → C

Parsing → Desugaring → Inlining → Optimization Transpiling
MLANG’s correctness

How to check that MLANG is correct?

- 476 tests from DGFiP
- Generation of our own tests
- Quality measure: value coverage

It works (precise down to the euro)!
All backends validated, on all tests

<table>
<thead>
<tr>
<th>Number of distinct values assigned</th>
<th>Percentage of assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>2 or more</td>
<td>40%</td>
</tr>
<tr>
<td>3</td>
<td>60%</td>
</tr>
<tr>
<td>4 or more</td>
<td>80%</td>
</tr>
</tbody>
</table>

DGFiP Private (476 tests)
Randomized (1267 tests)
Fuzzer-generated (275 tests)
### How to check that MLANG is correct?

- 476 tests from DGFiP
- Generation of our own tests
- Quality measure: value coverage

### It works (precise down to the euro)!

All backends validated, on all tests
MLANG’s correctness

How to check that MLANG is correct?

- 476 tests from DGFiP
- Generation of our own tests
- Quality measure: value coverage

It works (precise down to the euro)!
All backends validated, on all tests
Code optimization

Compiler optimizations

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
**Code optimization**

**Compiler optimizations**

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
# Code optimization

## Compiler optimizations

- Global value numbering
- Dead code elimination
- Partial evaluation
- Dataflow defined-ness analysis

<table>
<thead>
<tr>
<th>Spec. name</th>
<th># inputs</th>
<th># outputs</th>
<th># instructions</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>2,459</td>
<td>10,411</td>
<td>129,683</td>
<td>80.2%</td>
</tr>
<tr>
<td>Selected outs</td>
<td>2,459</td>
<td>11</td>
<td>99,922</td>
<td>84.8%</td>
</tr>
<tr>
<td>Tests</td>
<td>1,635</td>
<td>646</td>
<td>111,839</td>
<td>83.0%</td>
</tr>
<tr>
<td>Simplified</td>
<td>228</td>
<td>11</td>
<td>4,172</td>
<td>99.4%</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td>1</td>
<td>553</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

# of instructions with optimizations disabled (2018 code): 656,020.
## Contributions around the French tax code

<table>
<thead>
<tr>
<th>Component</th>
<th>Published by DGFiP?</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>yes</td>
<td>Reverse-engineering $M \rightsquigarrow \mu M$ in Coq</td>
</tr>
<tr>
<td>“inter” code (C)</td>
<td>no</td>
<td><strong>DSL</strong> $M++$ (6kLoc C $\rightsquigarrow$ 100 $M++$)</td>
</tr>
<tr>
<td>M compiler</td>
<td>no</td>
<td><strong>MLANG</strong> (10kLoc OCaml) with optimizations</td>
</tr>
<tr>
<td>Tests</td>
<td>no</td>
<td>Fuzzing-generated tests (better coverage)</td>
</tr>
</tbody>
</table>

⇒ Now reproducible outside DGFiP!
## Contributions around the French tax code

<table>
<thead>
<tr>
<th>Component</th>
<th>Published by DGFiP?</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>yes</td>
<td>Reverse-engineering $M \mapsto \mu M$ in Coq</td>
</tr>
<tr>
<td>“inter” code (C)</td>
<td>no</td>
<td><strong>DSL</strong> $M^{++}$ ($6k\text{Loc C} \mapsto 100 M^{++}$)</td>
</tr>
<tr>
<td>M compiler</td>
<td>no</td>
<td><strong>MLANG</strong> ($10k\text{Loc OCaml}$) with optimizations</td>
</tr>
<tr>
<td>Tests</td>
<td>no</td>
<td>Fuzzing-generated tests (better coverage)</td>
</tr>
</tbody>
</table>

$\Rightarrow$ Now reproducible outside DGFiP!

## Contributions around the French tax code

<table>
<thead>
<tr>
<th>Component</th>
<th>Published by DGFiP?</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>yes</td>
<td>Reverse-engineering $M \rightsquigarrow \mu M$ in Coq</td>
</tr>
<tr>
<td>&quot;inter&quot; code (C)</td>
<td>no</td>
<td>DSL $M++$ (6kLoc C $\rightsquigarrow$ 100 $M++$)</td>
</tr>
<tr>
<td>$M$ compiler</td>
<td>no</td>
<td>MLANG (10kLoc OCaml) with optimizations</td>
</tr>
<tr>
<td>Tests</td>
<td>no</td>
<td>Fuzzing-generated tests (better coverage)</td>
</tr>
</tbody>
</table>

$\implies$ Now reproducible outside DGFiP!

### Interacting with DGFiP

- Long term work: 9 months to access the missing C code
- Pedagogy required: very legal environment

Contributions around the French tax code

<table>
<thead>
<tr>
<th>Component</th>
<th>Published by DGFiP?</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>yes</td>
<td>Reverse-engineering $\mu M$ in Coq</td>
</tr>
<tr>
<td>“inter” code (C)</td>
<td>no</td>
<td>DSL M++ (6kLoc C $\Rightarrow$ 100 M++)</td>
</tr>
<tr>
<td>M compiler</td>
<td>no</td>
<td>Compiler (10kLoc OCaml) with optimizations</td>
</tr>
<tr>
<td>Tests</td>
<td>yes</td>
<td>Fuzzing-generated tests (better coverage)</td>
</tr>
</tbody>
</table>

Now reproducible outside DGFiP!

Interacting with DGFiP:

- Long term work: 9 months to access the missing C code
- Pedagogy required: very legal environment

Transfer to DGFiP!

Ongoing work to bring Mlang at DGFiP.

- 30-day mission between January and August
- Supervision of 3 developers (2 OCamlPro, 1 DGFiP)

Does the implementation comply with the law?

Law (specification) \(\xrightarrow{\text{GF-1A}}\) Technical description \(\xrightarrow{\text{BSI4}}\) Implementation
### Future DSLs

**Does the implementation comply with the law?**

<table>
<thead>
<tr>
<th>Law (specification)</th>
<th>GF-1A</th>
<th>Technical description</th>
<th>BSI4</th>
<th>Implementation</th>
</tr>
</thead>
</table>

- No structural correspondance
- 2019～2020: 30% of 90kLoc M changed

---


---

"GF-1A" and "BSI4" seem to be specific identifiers or versions, possibly referring to standards or tools related to the law and technical description processes.
Does the implementation comply with the law?

Catala, another DSL to the rescue

Article D521-1 du code de la sécurité sociale

I - Pour l’application de l’article L. 521-1, le montant des allocations familiales et de la majoration pour âge prévue à l’article L. 521-3 est défini selon le barème suivant :

1° Lorsque le ménage ou la personne a disposé d’un montant de ressources inférieur ou égal au plafond défini au I de l’article D. 521-3, les taux servant au calcul des allocations familiales sont fixés, en pourcentage de la base mensuelle prévue à l’article L. 551-1, à :

a) 32 % pour le deuxième enfant à charge ;

```catala
champ d'application AllocationsFamiliales :
définition montant_initial_base_deuxième_enfant sous condition
  ressources_ménage ≤ € plafond_I_d521_3
conséquence égal à
  si nombre de enfants_à_charge_droit_ouvert_prestation_familiale ≥ 2
    alors prestations_familiales.base_mensuelle × € 32 %
  sinon 0 €
```

Does the implementation comply with the law?

Catala, another DSL to the rescue

Article D521-1 du code de la sécurité sociale

I - Pour l’application de l’article L. 521-1, le montant des allocations familiales et de la majoration pour âge prévue à l’article L. 521-3 est défini selon le barème suivant :

1° Lorsque le ménage ou la personne a disposé d’un montant de ressources inférieur ou égal au plafond défini au I de l’article D. 521-3, les taux servant au calcul des allocations familiales sont fixés, en pourcentage de la base mensuelle prévue à l’article L. 551-1, à :

a) 32 % pour le deuxième enfant à charge ;

```catala
champ d'application AllocationsFamiliales :
  définition montant_initial_base_deuxième_enfant sous condition
    ressources_ménage ≤ € plafond_I_d521_3
  conséquence égal à
    si nombre de enfants_à_charge_droit_ouvert_prestation_familiale ≥ 2
      alors prestations_familiales.base_mensuelle ×€ 32 %
    sinon 0 €
```

▶ Literal programming
▶ Default logic
▶ Participation to ongoing development

Conclusion
### Summary of the contributions

<table>
<thead>
<tr>
<th>Static analysis of Python programs using C libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Ouadjaout and Miné (LIP6, Sorbonne Université)</td>
</tr>
<tr>
<td>▶ SAS’21, ECOOP’20, VSTTE’19 (invited), SOAP@PLDI’20 (award), JFLA’21 (fr, tool)</td>
</tr>
<tr>
<td>▶ Mopsa (LGPL v3, 60kLoc OCaml), main contributor since September 2018</td>
</tr>
</tbody>
</table>
Summary of the contributions

<table>
<thead>
<tr>
<th>Static analysis of Python programs using C libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Ouadjaout and Miné (LIP6, Sorbonne Université)</td>
</tr>
<tr>
<td>▶ SAS’21, ECOOP’20, VSTTE’19 (invited), SOAP@PLDI’20 (award), JFLA’21 (fr, tool)</td>
</tr>
<tr>
<td>▶ Mopsa (LGPL v3, 60kLoc OCaml), main contributor since September 2018</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A modern compiler for the French tax code</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Merigoux (Prosecco, Inria Paris) et Protzenko (Microsoft Research)</td>
</tr>
<tr>
<td>▶ CC’21, JFLA’20 (fr), JFLA’21 (fr, tool)</td>
</tr>
<tr>
<td>▶ Mlang (GPL v3, 10kLoc OCaml), main contributor since May 2019</td>
</tr>
<tr>
<td>▶ Ongoing transfer work</td>
</tr>
<tr>
<td>• 30 days mission between January and August</td>
</tr>
<tr>
<td>• Supervision of 3 developers (2 OCamlPro, 1 DGFiP)</td>
</tr>
</tbody>
</table>
Future research directions

Current state of static analysis tools

- Astrée: sound analysis of specific critical systems
- Infer: bug detector for general programs

⇒ Unify approaches to ensure a massive adoption by developers
Future research directions

Current state of static analysis tools

- Astrée: sound analysis of specific critical systems
- Infer: bug detector for general programs

⇒ Unify approaches to ensure a massive adoption by developers

Precise, sound and efficient static analyses for general programs

1. Formalization and analysis of concrete semantics $\cong S_{py}[\cdot] \cong S_{py}[\cdot]$  
2. Make static analysis more usable
3. Formal methods for legal code
Formal methods for real-world systems: study of two cases

Questions

Raphaël Monat