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

Raphaël Monat

LIP, ENS de Lyon

28 April 2022

rmonat.fr

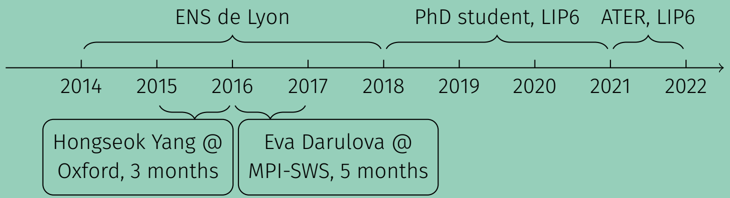
Introduction

Introduction

Curriculum

Introduction

Curriculum

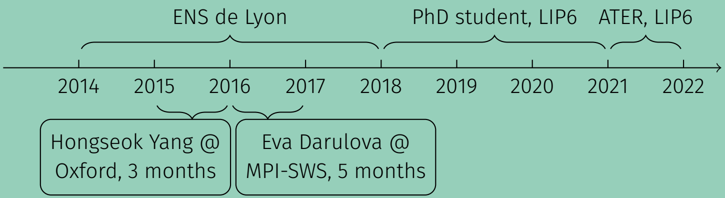


Research field: formal methods

$\Rightarrow$ Improve confidence in software.

Introduction

Curriculum



Research field: formal methods

$\Rightarrow$ Improve confidence in software.

Means

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

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)

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 $\rightsquigarrow$ static analysis
  - Abdelraouf Ouadjaout (LIP6)
  - Antoine Miné (LIP6)

Introduction

Personal methodology

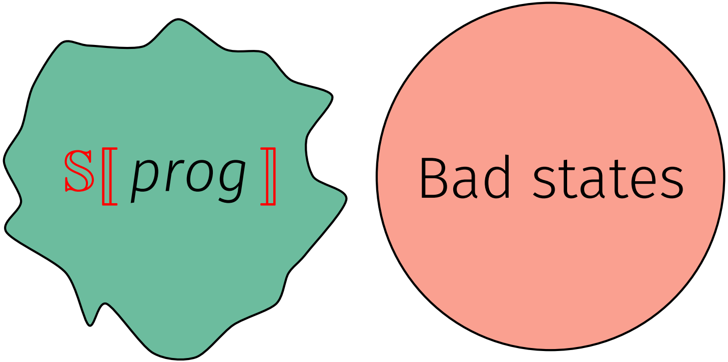
Constant back and forth between theory and practice

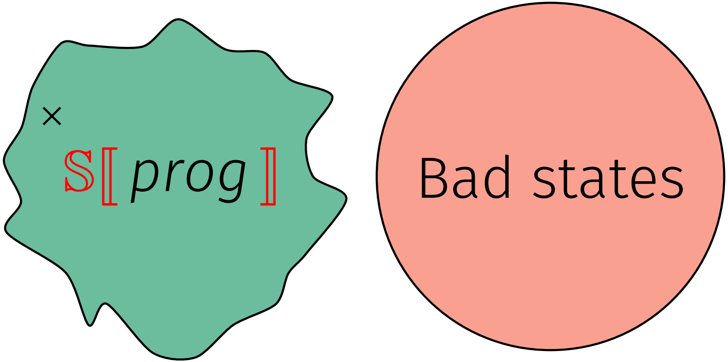
1. Find interesting bugs, properties or systems to study (GitHub, ...)
2. Theoretical study and solution
3. Implementation and experimental validation (on 1)

Studied systems

- Python programs using C libraries $\rightsquigarrow$ static analysis
  - Abdelraouf Ouadjaout (LIP6)
  - Antoine Miné (LIP6)
- Implementation of the French tax code $\rightsquigarrow$ compiler, modernization
  - Denis Merigoux (Inria Prosecco)
  - Jonathan Protzenko (MSR)

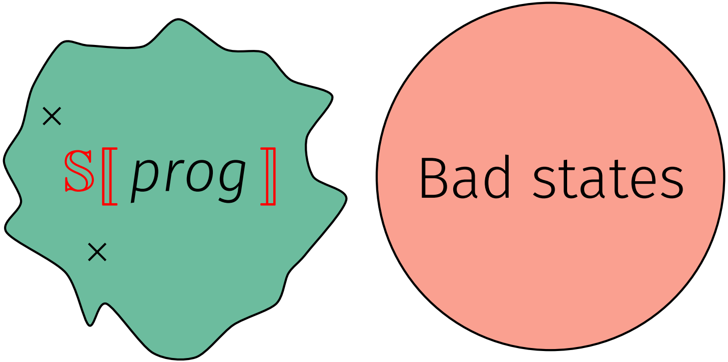
Software verification


$\mathbb{S}[\![prog]\!]$
Bad states

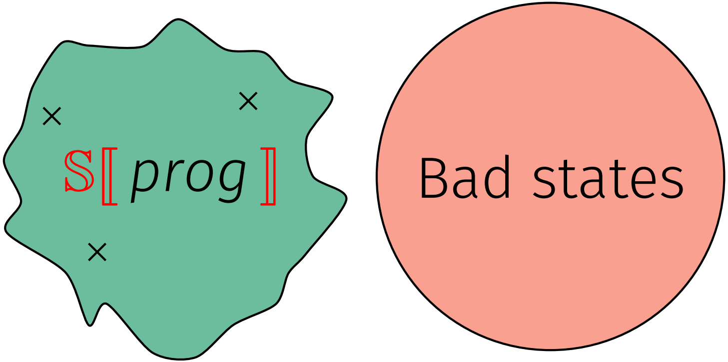


Cheap approach: test *prog*.

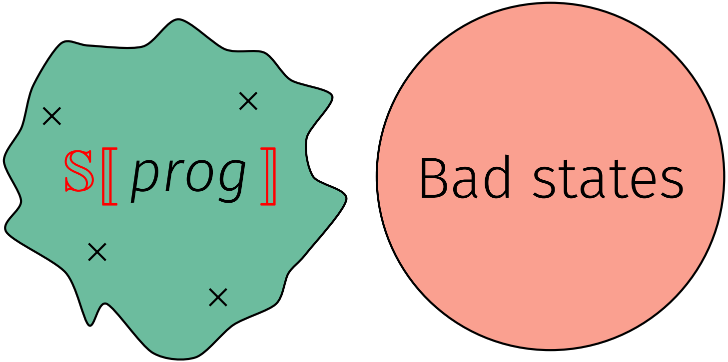
Software verification



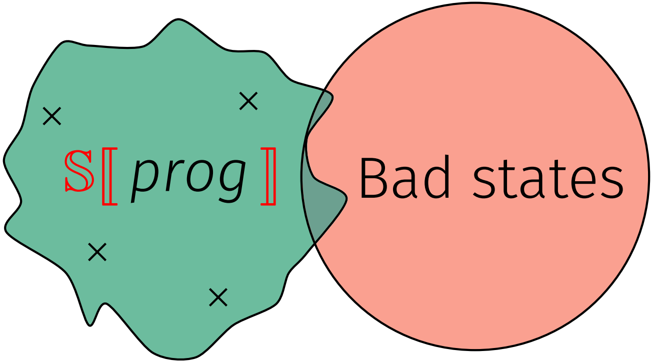
Cheap approach: test *prog*.



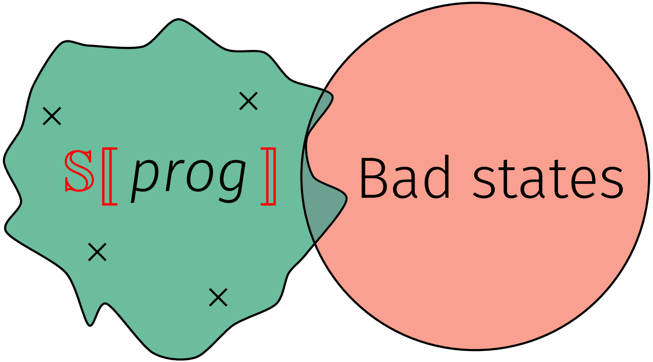
Cheap approach: test *prog*.



Cheap approach: test *prog*.



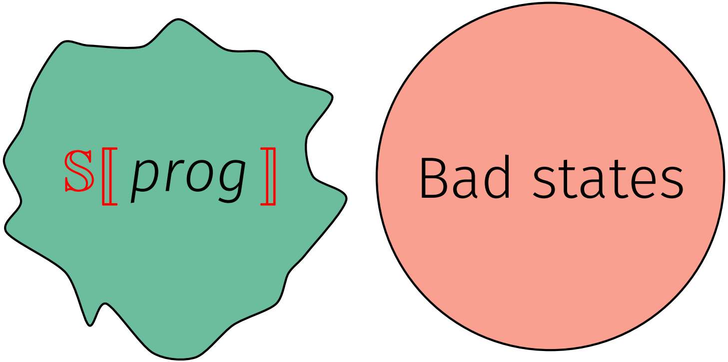
Cheap approach: test *prog*.



Cheap approach: test *prog*.

Some bugs may go undetected!

Software verification

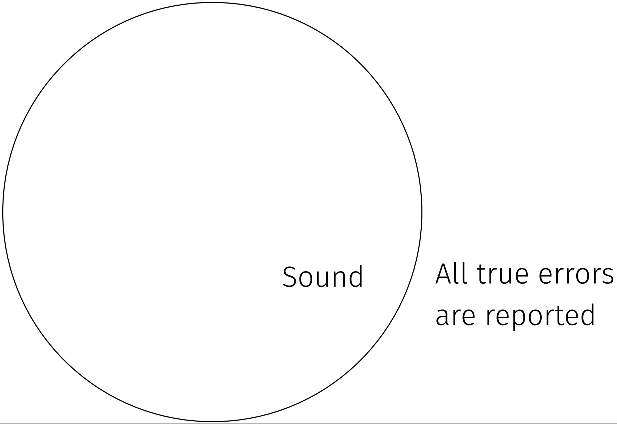


Cheap approach: test *prog*.

Some bugs may go undetected!

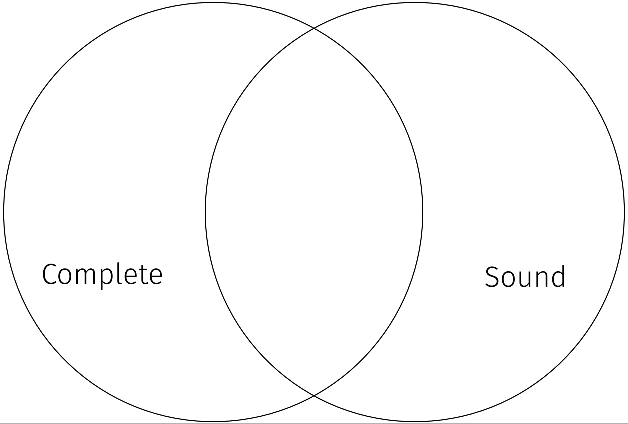
Would there be a way to automatically prove programs correct?

An impossibility theorem



An impossibility theorem

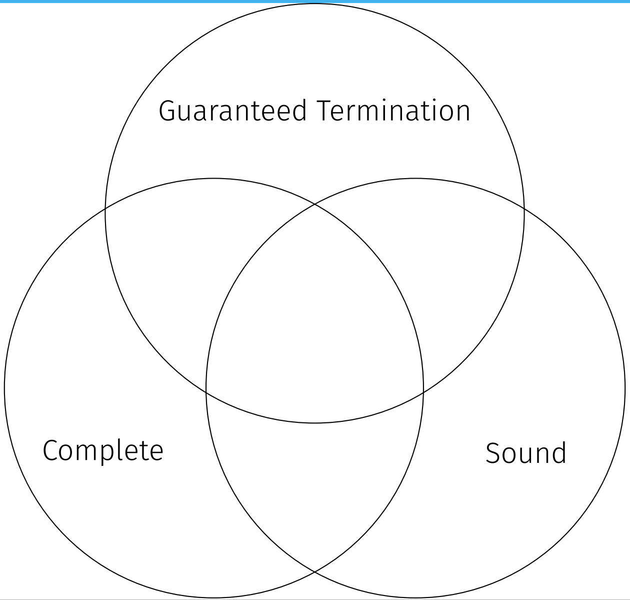
All reported errors are true errors



All true errors are reported

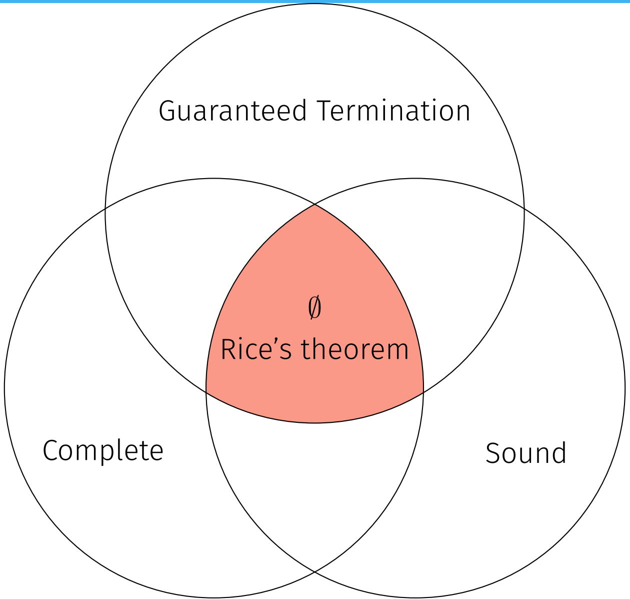
An impossibility theorem

All reported errors are true errors



All true errors are reported

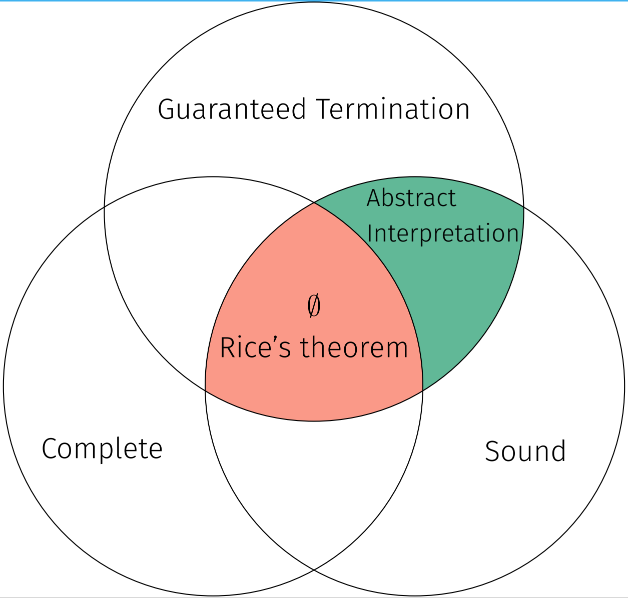
An impossibility theorem



All reported errors are true errors

All true errors are reported

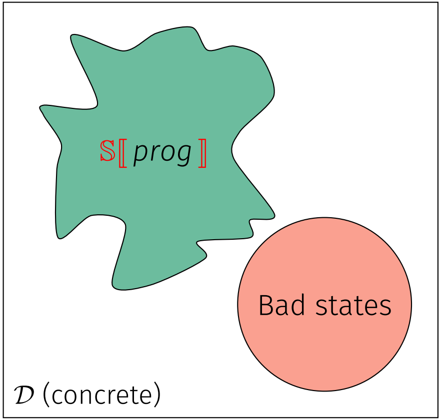
An impossibility theorem



All reported errors are true errors

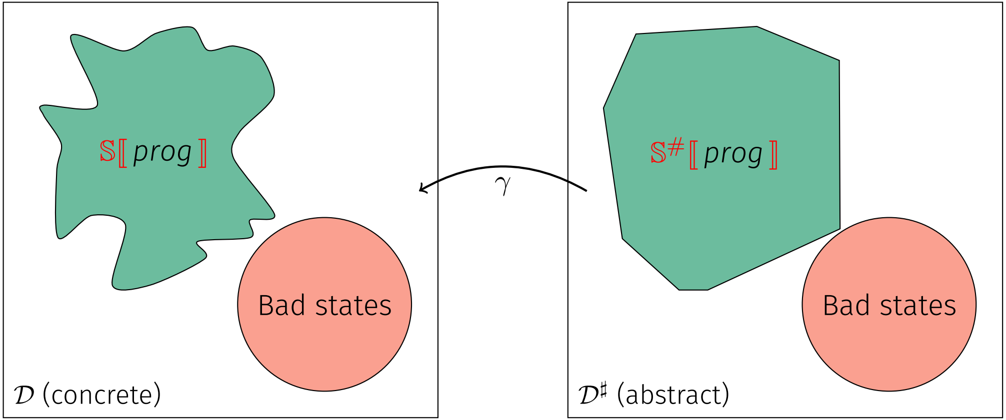
All true errors are reported

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

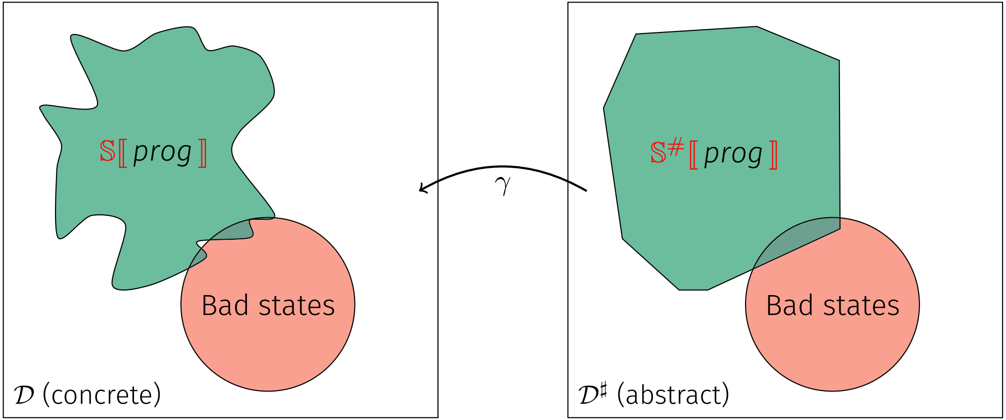
Abstract interpretation – the big picture



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

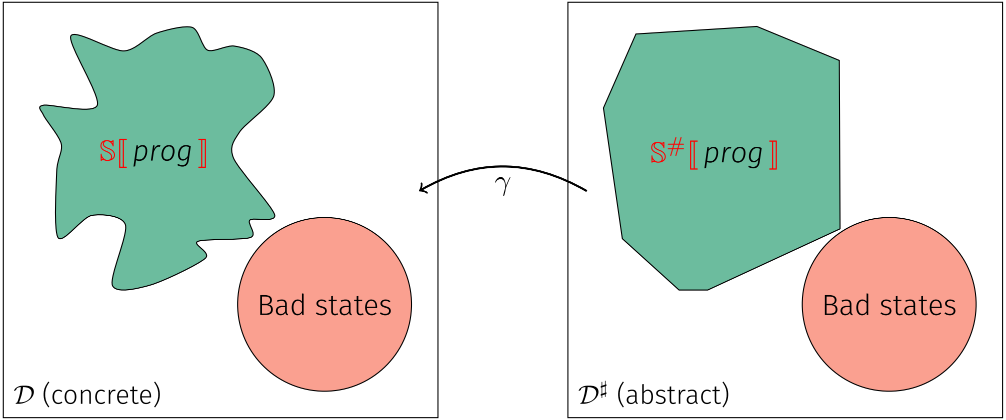
Abstract interpretation – the big picture



True alarm

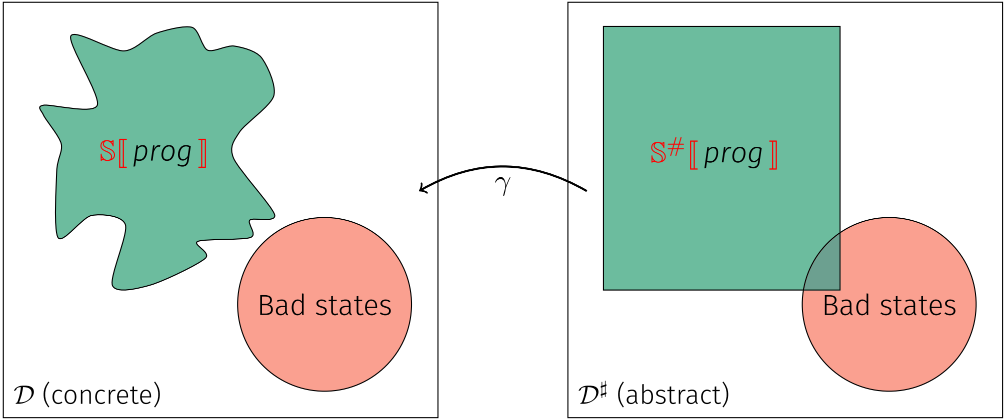
P. Cousot and R. Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". POPL 1977

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

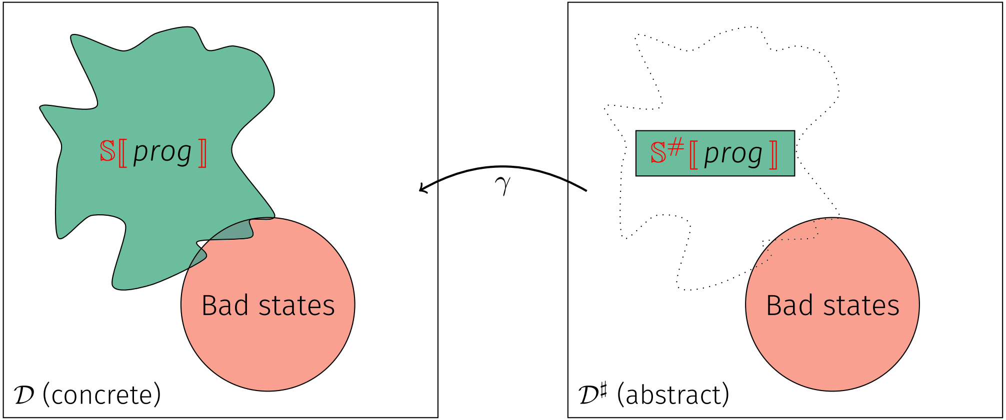
Abstract interpretation – the big picture



False alarm (Abstraction too coarse)

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



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

Static analysis of critical software

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]

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

Dynamic programming languages

Growing popularity

JavaScript #1, Python #2 on GitHub[1]

New features

- Object orientation,

[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,

[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,

[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`

Outline

A Taste of Python

Python's specificities

No standard

- CPython is the reference

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

Python's specificities

No standard

- CPython is the reference

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

Operator redefinition

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

Protected attributes

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

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

Python's specificities (II)

Dual type system

- Nominal (classes, MRO)

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())
```

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

Python’s specificities (II)

Dual type system

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

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())
```

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

Python’s specificities (II)

Dual type system

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

Exceptions

Exceptions rather than specific values

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

Fspath (from standard library)

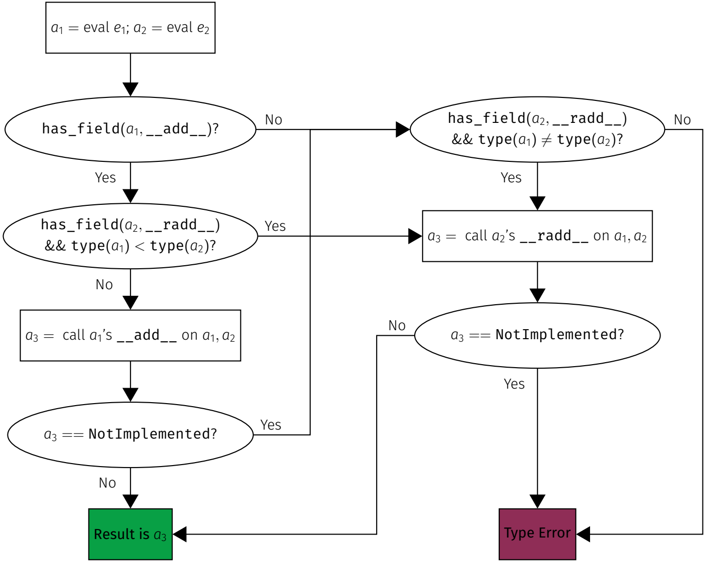
```python
class Path:
  def __fspath__(self): return 42

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())
```

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

Example Semantics – binary operators



Crazy Python

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(f"{x} + {y} = {x + y}")
  return x + y
c = 2 |padd| 3
```

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

Analyzing Python Programs

Goal

Detect runtime errors: uncaught raised exceptions

Goal

Detect runtime errors: uncaught raised exceptions

Supported constructs

Our analysis supports:

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

Goal

Detect runtime errors: uncaught raised exceptions

Supported constructs

Our analysis supports:

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

Unsupported constructs

- Recursive functions
- `eval`
- Finalizers

Analysis | Domains required

Averaging numbers

```
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)
```

Analysis | Domains required

Averaging numbers

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Environment abstraction

$m \mapsto @^\sharp_{\mathtt{int}^\sharp} \quad i \mapsto @^\sharp_{\mathtt{int}^\sharp}$

Analysis | Domains required

Averaging numbers

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

Searching for a loop invariant (l. 4)

Stateless domains: **list content**,

Environment abstraction

$m \mapsto @^\sharp_{\texttt{int}^\sharp} \quad i \mapsto @^\sharp_{\texttt{int}^\sharp} \quad \underline{\text{els}}(l) \mapsto @^\sharp_{\texttt{int}^\sharp}$

Proved safe?

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

Analysis | Domains required

Averaging numbers

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Stateless domains: list content,

Environment abstraction

$m \mapsto @^\sharp_{\mathtt{int}^\sharp}$ $\quad i \mapsto @^\sharp_{\mathtt{int}^\sharp}$ $\quad \underline{\mathrm{els}}(l) \mapsto @^\sharp_{\mathtt{int}^\sharp}$

Numeric abstraction (intervals)

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

Analysis | Domains required

Averaging numbers

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

Proved safe?

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

Searching for a loop invariant (l. 4)

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

Environment abstraction

$m \mapsto @^{\sharp}_{\mathtt{int}^{\sharp}}$ $\quad i \mapsto @^{\sharp}_{\mathtt{int}^{\sharp}}$ $\quad \underline{\mathrm{els}}(l) \mapsto @^{\sharp}_{\mathtt{int}^{\sharp}}$

Numeric abstraction (intervals)

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

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

Analysis | Domains required

Averaging numbers

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

Environment abstraction

$m \mapsto @^\sharp_{\mathtt{int}^\sharp} \quad i \mapsto @^\sharp_{\mathtt{int}^\sharp} \quad \underline{\mathrm{els}}(l) \mapsto @^\sharp_{\mathtt{int}^\sharp}$

Numeric abstraction (polyhedra)

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

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

Analysis | Domains required

Averaging tasks

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

Environment abstraction

$m \mapsto @^\sharp_{\texttt{int}^\sharp}$ $\quad i \mapsto @^\sharp_{\texttt{int}^\sharp}$ $\quad \underline{\mathsf{els}(l)} \mapsto @^\sharp_{\texttt{Task}}$

$\underline{@^\sharp_{\texttt{Task}} \cdot \texttt{weight}} \mapsto @^\sharp_{\texttt{int}^\sharp}$

Numeric abstraction (polyhedra)

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

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

$0 \leq \underline{@^\sharp_{\texttt{Task}} \cdot \texttt{weight}} \leq 20$

Attributes abstraction

$@^\sharp_{\texttt{Task}} \mapsto (\{\,\texttt{weight}\,\}, \emptyset)$

Analysis | Domains required

Averaging tasks

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

Proved safe?

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

Searching for a loop invariant (l. 4)

Stateless domains: list content, list length

Environment abstraction

$m \mapsto @^\sharp$ ... $@^\sharp_{\texttt{Task}}$

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

$0 \leq \underline{@^\sharp_{\texttt{Task}} \cdot \texttt{weight}} \leq 20$

Attributes abstraction

$@^\sharp_{\texttt{Task}} \mapsto (\{\texttt{weight}\}, \emptyset)$

Conclusion

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

Modular Open Platform for Static Analysis[2]

[2] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019.

Modular Open Platform for Static Analysis[2]

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

[2] Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.

Modular Open Platform for Static Analysis[2]

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability
- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability

[2]Journault, Miné, Monat, and Ouadjaout. “Combinations of reusable abstract domains for a multilingual static analyzer”. VSTTE 2019.

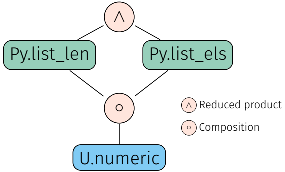
Modular Open Platform for Static Analysis[2]

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability
- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability
- DAG of abstract domains
  - Composition
  - Cooperation

[2] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019.

Modular Open Platform for Static Analysis[2]

- One AST to analyze them all
  - Multilanguage support
  - Expressiveness
  - Reusability
- Unified domain signature
  - Semantic rewriting
  - Loose coupling
  - Observability
- DAG of abstract domains
  - Composition
  - Cooperation



[2] Journault, Miné, Monat, and Ouadjaout. "Combinations of reusable abstract domains for a multilingual static analyzer". VSTTE 2019.

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

Mopsa | Dynamic, semantic iterators with delegation

```
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

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

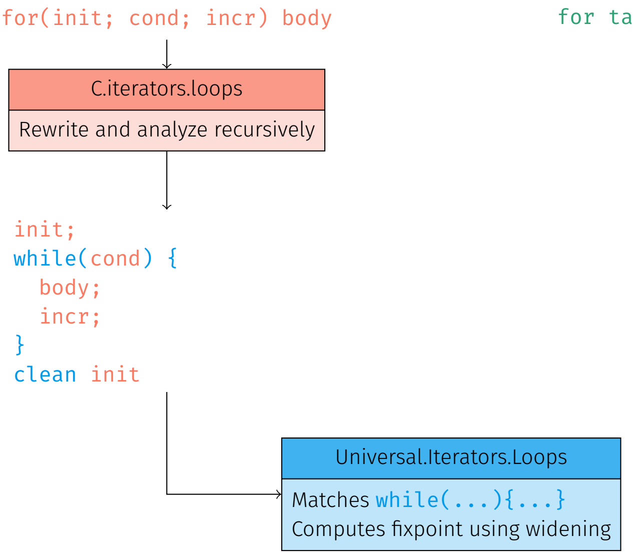
```
init;
while(cond) {
  body;
  incr;
}
clean init
```

Universal.Iterators.Loops

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

Computes fixpoint using widening

Mopsa | Dynamic, semantic iterators with delegation



for target in iterable: body

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

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

Python.Desugar.Loops

- Rewrite and analyze recursively
- Optimize for some <u>semantic</u> cases

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
```

Python.Desugar.Loops

- Rewrite and analyze recursively
- Optimize for some <u>semantic</u> cases

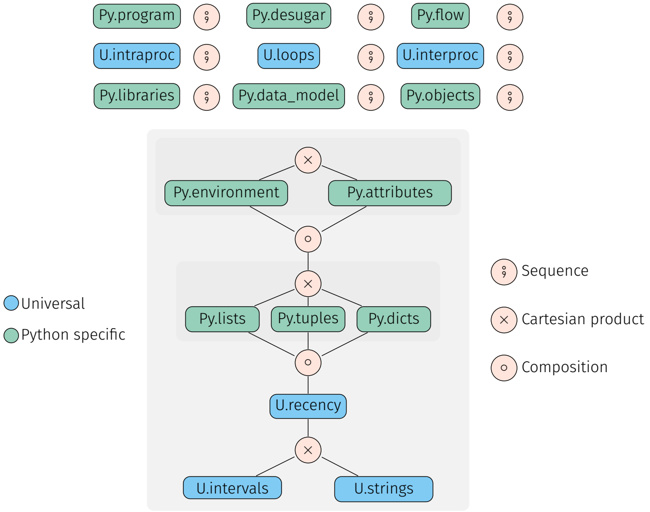
```
it = iter(iterable)
while(1) {
 try: target = next(it)
 except StopIteration: break
 body
}
clean it
```

Universal.Iterators.Loops

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

Computes fixpoint using widening

Definition of the Value Analysis



Comparison of the type and value analyses

Averaging tasks

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

Type analysis

- `ValueError` (l. 3)

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

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

Type analysis

- `ValueError` (l. 3)
- `IndexError` (l. 9)

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

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

Type analysis

- `ValueError` (l. 3)
- `IndexError` (l. 9)
- `ZeroDivisionError` (l. 10)

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

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

Type analysis

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

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

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

Type analysis

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

Non-relational value analysis

`IndexError` (l. 9)

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

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

Type analysis

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

Non-relational value analysis

`IndexError` (l. 9)

Relational value analysis

No alarm!

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)

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

Comparison of the type and value analyses (II)

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

Conclusion

The non-relational value analysis

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

Comparison of the type and value analyses (II)

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

Conclusion

The non-relational value analysis

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

Heuristic packing and relational analyses

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

Combining C and Python – motivation

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

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

Combining C and Python – motivation

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

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

Pitfalls

Combining C and Python – motivation

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

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

Pitfalls

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

Combining C and Python – motivation

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

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

Pitfalls

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

Combining C and Python – motivation

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

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

Pitfalls

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

Combining C and Python – example

counter.c

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

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

    self->count += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

Combining C and Python – example

counter.c

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

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

    self->count += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

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

Combining C and Python – example

counter.c

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

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

    self->count += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

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

Combining C and Python – example

counter.c

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

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

    self->count += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

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

How to analyze multilanguage programs?

Type annotations

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

How to analyze multilanguage programs?

Type annotations

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

- No raised exceptions $\Longrightarrow$ missed errors

How to analyze multilanguage programs?

Type annotations

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

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

How to analyze multilanguage programs?

Type annotations

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

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

How to analyze multilanguage programs?

Type annotations

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

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

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

- No integer wrap-around in Python

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

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

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

- Not the real code
- Not automatic: manual conversion

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

Our approach

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

Our approach

- Analyze both the C and Python sources

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

Our approach

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

Our approach

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

How to analyze multilanguage programs?

Type annotations

Rewrite into Python code

Drawbacks of the current approaches

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

Our approach

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

Analysis result

counter.c

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

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

    self->count += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

Analysis result

counter.c

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

count.py

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

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

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

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

 6: c.incr(2**p-1)
    ^^^^^^^^^^^^^^
 Uncaught Python exception: OverflowError: signed integer is greater than maximum
 Uncaught Python exception: OverflowError: Python int too large to convert to C long
 Callstack:
       from ./counter.c:17.6-38::convert_single[0]: PyParseTuple_int
       from count.py:7.0-14: CounterIncr
 +1 other callstack
```

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
  - <u>some</u> 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

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

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

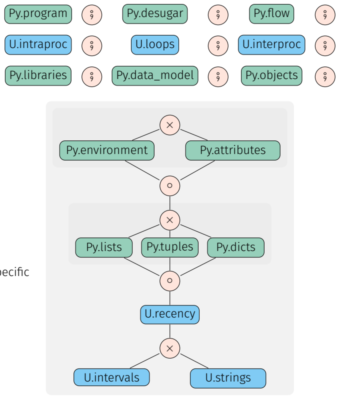
Concrete definition

- 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

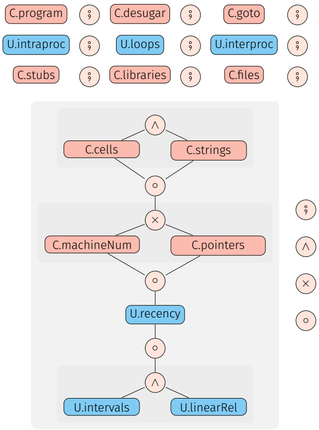
From distinct Python and C analyses...



Universal

C specific

Python specific



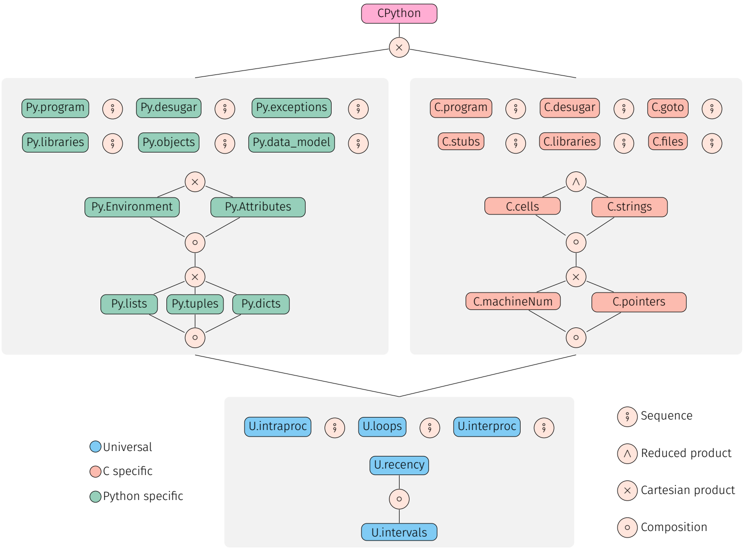
⨾ Sequence

∧ Reduced product

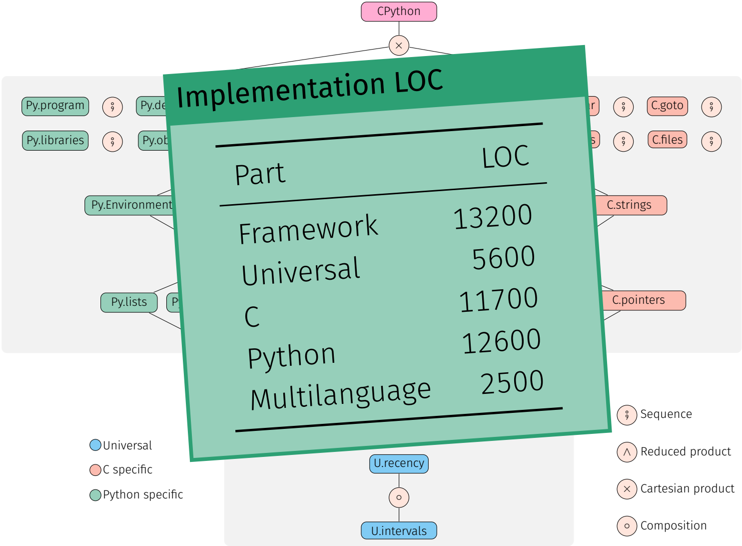
× Cartesian product

∘ Composition

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



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



Implementation LOC

Part	LOC
Framework	13200
Universal	5600
C	11700
Python	12600
Multilanguage	2500

Benchmarks

Corpus selection

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

Library	C + Py. Loc	Tests	⏱/test	$\frac{\text{\# proved checks}}{\text{\# checks}}$ %	# checks
`noise`	1397	15/15	1.2s	99.7%	(6690)
`cdistance`	2345	28/28	4.1s	98.0%	(13716)
`llist`	4515	167/194	1.5s	98.8%	(36255)
`ahocorasick`	4877	46/92	1.2s	96.7%	(6722)
`levenshtein`	5798	17/17	5.3s	84.6%	(4825)
`bitarray`	5841	159/216	1.6s	94.9%	(25566)

Python's semantics $\rightsquigarrow \mathbb{S}_{py}[\![\cdot]\!]$

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

Contributions around the static analysis of Python programs

Python's semantics 🐍 $\rightsquigarrow \mathbb{S}_{py}[\![\cdot]\!]$

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

Python's analyses $\mathbb{S}_{py}[\![\cdot]\!] \rightsquigarrow \mathbb{S}^{\#}_{py}[\![\cdot]\!]$

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

Contributions around the static analysis of Python programs

Python's semantics $\leadsto \mathbb{S}_{py}[\![\cdot]\!]$

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

Python's analyses $\mathbb{S}_{py}[\![\cdot]\!] \leadsto \mathbb{S}^{\#}_{py}[\![\cdot]\!]$

- 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

Python's semantics $\rightsquigarrow \mathbb{S}_{py}[\![\cdot]\!]$

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

Python's analyses $\mathbb{S}_{py}[\![\cdot]\!] \rightsquigarrow \mathbb{S}^{\#}_{py}[\![\cdot]\!]$

- 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

Implementation into Mopsa

- Open-source analyzer for C and Python
- Factors abstractions between languages
- Sharing between abstractions

A Modern Compiler for the French Tax Code

Back to the introduction

Research field: formal methods

$\Rightarrow$ Improve confidence in software.

Back to the introduction

Research field: formal methods

$\Rightarrow$ 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)

Legal implementations

French income tax

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

Legal implementations

French income tax

- 38M households, 75Md€ of income
- Made public in April 2016 : $\simeq$ 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?

Example M code

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));
```

M, briefly

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

M, briefly

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

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
- `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)

[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[3]
- Later, using the private tests DGFiP sent us (August 7, 2019)

$\Rightarrow$ a μM 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[3]
- Later, using the private tests DGFiP sent us (August 7, 2019)

$\Rightarrow$ a μM kernel, its semantics formalized in the Coq proof assistant.

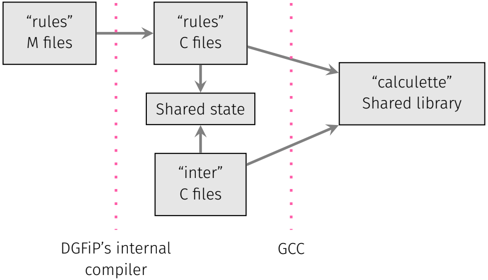
The `undef` value

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

[3] `https://www3.impots.gouv.fr/simulateur/calcul_impot/2020/index.htm`

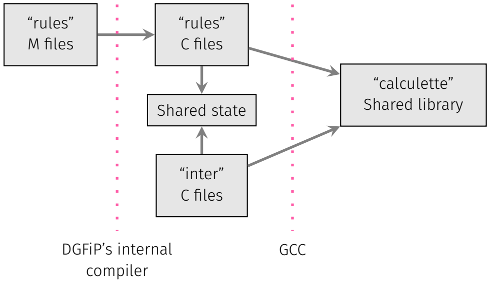
DGFiP's legacy architecture

After 9 months of negotiations, we're in!



DGFiP’s legacy architecture

After 9 months of negotiations, we’re in!

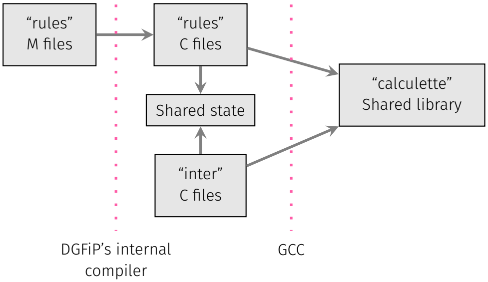


“inter” files

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

DGFiP's legacy architecture

After 9 months of negotiations, we're in!



"inter" files

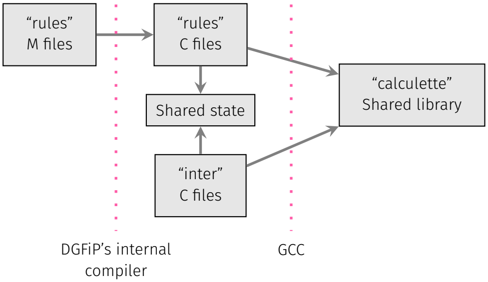
35kLoc of C to bypass M's lack of functions.

Security concerns ⇝ no publication

How to extract the logic of the code?

DGFiP's legacy architecture

After 9 months of negotiations, we're in!



"inter" files

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

Security concerns $\rightsquigarrow$ 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 $\Rightarrow$ 100 lines of M++

MLANG's architecture

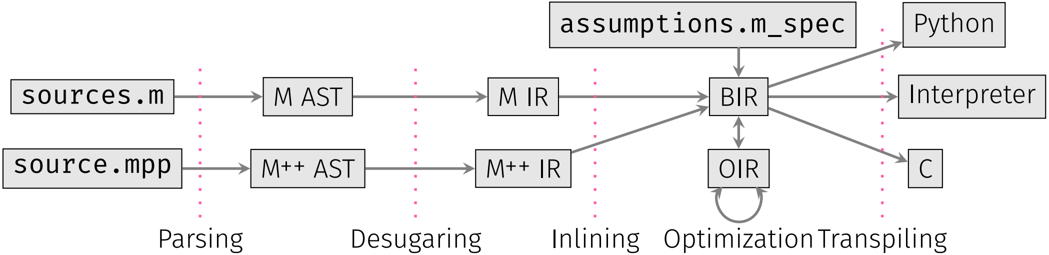
MLANG: written in OCaml, 10k lines of code

`https://github.com/MLanguage/mlang`

MLANG's architecture

MLANG: written in OCaml, 10k lines of code

`https://github.com/MLanguage/mlang`



MLANG's correctness

How to check that MLANG is correct?

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

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

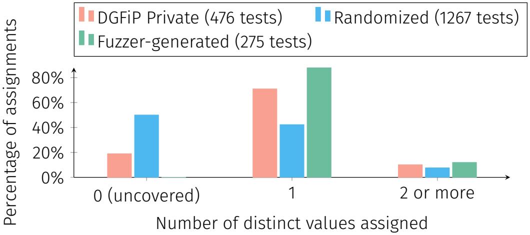
MLANG's correctness

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

Compiler optimizations

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

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

# 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

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

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

Contributions around the French tax code

Component	Published by DGFiP?	Contributions
M	yes	Reverse-engineering M $\leadsto$ μM in Coq
“inter” code (C)	no	**DSL** M++ (6kLoc C $\leadsto$ 100 M++)
M compiler	no	MLANG (10kLoc OCaml) with optimizations
Tests	no	Fuzzing-generated tests (better coverage)

$\Longrightarrow$ Now reproducible outside DGFiP!

Contributions around the French tax code

Component	Published by DGFiP?	Contributions
M	yes	Reverse-engineering M $\rightsquigarrow$ μM in Coq
"inter" code (C)	no	**DSL** M++ (6kLoc C $\rightsquigarrow$ 100 M++)
M compiler	no	MLANG (10kLoc OCaml) with optimizations
Tests	no	Fuzzing-generated tests (better coverage)

$\Longrightarrow$ Now reproducible outside DGFiP!

Merigoux, Raphaël Monat, and Protzenko. "A Modern Compiler for the French Tax Code". Compiler Construction (CC) 2021

Contributions around the French tax code

Component	Published by DGFiP?	Contributions
M	yes	Reverse-engineering M $\rightsquigarrow$ μM in Coq
"inter" code (C)	no	**DSL** M++ (6kLoc C $\rightsquigarrow$ 100 M++)
M compiler	no	MLANG (10kLoc OCaml) with optimizations
Tests	no	Fuzzing-generated tests (better coverage)

$\Longrightarrow$ Now reproducible outside DGFiP!

Interacting with DGFiP

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

Merigoux, Raphaël Monat, and Protzenko. "A Modern Compiler for the French Tax Code". Compiler Construction (CC) 2021

Contributions around the French tax code

Component	Published by DGFiP?	Contributions
M	yes	Reverse-engineering M $\rightsquigarrow$ μM in Coq
“inter” code (C)	no	…)
M comp…		…ptimizations
Tests		…r coverage)

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)

Interactin…

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

Merigoux, Raphaël Monat, and Protzenko. “A Modern Compiler for the French Tax Code”. Compiler Construction (CC) 2021

Future DSLs

Does the implementation comply with the law?

$$\boxed{\text{Law (specification)}} \xrightarrow{\text{GF-1A}} \boxed{\text{Technical description}} \xrightarrow{\text{BSI4}} \boxed{\text{Implementation}}$$

Future DSLs

Does the implementation comply with the law?

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

- No structural correspondance
- 2019$\rightsquigarrow$2020 : 30% of 90kLoc M changed

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 €
```
````

Merigoux, Chataing, and Protzenko. "Catala: a programming language for the law". 2021

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

Merigoux, Chataing, and Protzenko. "Catala: a programming language for the law". 2021

Conclusion

Summary of the contributions

Static analysis of Python programs using C libraries

- Ouadjaout and Miné (LIP6, Sorbonne Université)
- SAS'21, ECOOP'20, VSTTE'19 (invited), *SOAP@PLDI'20* (award), JFLA'21 (fr, tool)
- Mopsa (LGPL v3, 60kLoc OCaml), main contributor since September 2018

Summary of the contributions

Static analysis of Python programs using C libraries

- Ouadjaout and Miné (LIP6, Sorbonne Université)
- SAS'21, ECOOP'20, VSTTE'19 (invited), *SOAP@PLDI'20* (award), JFLA'21 (fr, tool)
- Mopsa (LGPL v3, 60kLoc OCaml), main contributor since September 2018

A modern compiler for the French tax code

- Merigoux (Prosecco, Inria Paris) et Protzenko (Microsoft Research)
- CC'21, JFLA'20 (fr), JFLA'21 (fr, tool)
- Mlang (GPL v3, 10kLoc OCaml), main contibutor since May 2019
- Ongoing transfer work
  - 30 days mission between January and August
  - Supervision of 3 developers (2 OCamlPro, 1 DGFiP)

Future research directions

Current state of static analysis tools

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

$\Rightarrow$ Unify approches 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

$\Rightarrow$ Unify approches to ensure a massive adoption by developers

Precise, sound and efficient static analyses for general programs

1. Formalization and analysis of concrete semantics $\rightsquigarrow \mathbb{S}_{py}[\![\cdot]\!] \rightsquigarrow \mathbb{S}^{\#}_{py}[\![\cdot]\!]$
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

LIP, ENS de Lyon

28 April 2022

rmonat.fr