Mopsa, a Multi-Lingual Static Analysis Platform

Antoine Miné, Abdelraouf Ouadjaout, Matthieu Journault, Raphaël Monat

12th March 2021

Introduction

Tool’s target: static program analysis

sum.py

```python
def sum(l):
  s = 0
  for x in l:
    s += x
  return s

r1 = sum([1, 2, 3])
r2 = sum(['a', 'b', 'c'])
```

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

argslen.c

```c
int main(int argc, char *argv[]) {
  int i = 0;
  for (char **p = argv; *p; p++) {
    strlen(*p); // valid string
    i++; // no overflow
  }
  return 0;
}
```

No alarm

Specifications of the analyzer

Infer	run-time errors (or other semantic properties)
Automatic	no expert knowledge required
Sound	cover all possible executions

How does an abstract interpreter work?

- Execution in approximate, computable domains
- Program $\rightsquigarrow$ Abstract state $\rightsquigarrow$ Semantic property (alarms)
- Combine abstract domains to gain precision

sum_indexed.py

```python
def sum(l):
  s = 0
  for i in range(len(l)):
    s += l[i]
  return s

r1 = sum([1, 2, 3])
r2 = sum(['a', 'b', 'c'])
```

How does an abstract interpreter work?

- Execution in approximate, computable domains
- Program $\rightsquigarrow$ Abstract state $\rightsquigarrow$ Semantic property (alarms)
- Combine abstract domains to gain precision

sum_indexed.py

```python
def sum(l):
  s = 0
  for i in range(len(l)):
    s += l[i]
  return s

r1 = sum([1, 2, 3])
r2 = sum(['a', 'b', 'c'])
```

Static Analysis by Abstract Interpretation

How does an abstract interpreter work?

- Execution in approximate, computable domains
- Program $\rightsquigarrow$ Abstract state $\rightsquigarrow$ Semantic property (alarms)
- Combine abstract domains to gain precision

sum_indexed.py

```python
def sum(l):
  s = 0
  for i in range(len(l)):
    s += l[i]
  return s

r1 = sum([1, 2, 3])
r2 = sum(['a', 'b', 'c'])
```

- Call with `[1, 2, 3]`

$$\left.\begin{array}{l}\texttt{len(l)} = 3 \\ 0 \leq \texttt{i} < 3\end{array}\right\}\text{ valid list accesses}$$

$$\texttt{s} \geq 0$$

How does an abstract interpreter work?

- Execution in approximate, computable domains
- Program $\rightsquigarrow$ Abstract state $\rightsquigarrow$ Semantic property (alarms)
- Combine abstract domains to gain precision

sum_indexed.py
```python
def sum(l):
  s = 0
  for i in range(len(l)):
    s += l[i]
  return s

r1 = sum([1, 2, 3])
r2 = sum(['a', 'b', 'c'])
```

- Call with `[1, 2, 3]`

$$\left.\begin{array}{l}\texttt{len(l)} = 3 \\ 0 \leq \texttt{i} < 3\end{array}\right\}\text{valid list accesses}$$

$$\texttt{s} \geq 0$$

- Call with `['a', 'b', 'c']`

$$\left.\begin{array}{l}\texttt{l} : \texttt{List[str]} \\ \texttt{s} : \texttt{int}\end{array}\right\}\texttt{int} + \texttt{str}\text{ invalid}$$

Project's target: experiment new development ideas

Modular Open Platform for Static Analysis

- Multi-language support (C and Python)

	Expressiveness	Keep the original AST of the program.
	Reusability	Reuse abstractions among languages.

- Flexible architecture

	Loose coupling	Divided into interchangeable components.
	Composition	Create complex components from simpler ones.
	Cooperation	Components can communicate and delegate tasks.
	Observability	Pluggable hooks observe the analysis.

Outline

Multi-language support from the AST

An extensible AST as input

One AST to analyze them all

- No static translation to an intermediate language
- The original AST of the program is kept
- Coexistence of all languages in the same AST
- Dynamic translation (analysis' results can be used as guide)

Cooperation by delegation

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

Cooperation by delegation

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

C.iterators.loops

Rewrite and analyze recursively

Cooperation by delegation

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

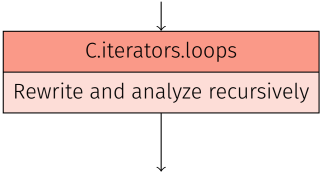
C.iterators.loops

Rewrite and analyze recursively

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

Cooperation by delegation

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




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

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

Cooperation by 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 recrusively
- Optimize for some semantic cases

Cooperation by 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 recrusively
- Optimize for some semantic cases

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

Cooperation by 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 recrusively
- Optimize for some <u>semantic</u> cases

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

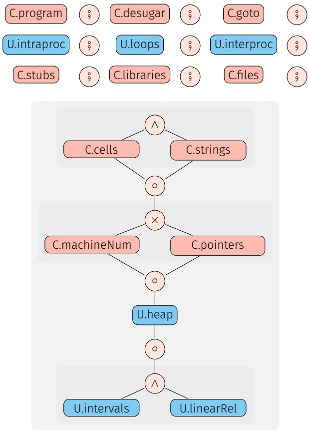
Universal.Iterators.Loops

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

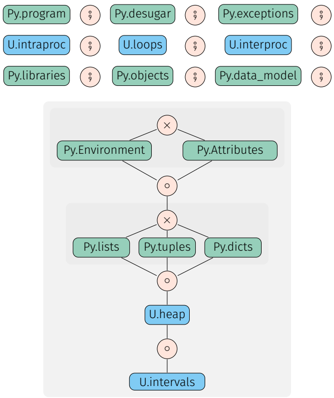
Flexible domains

Flexible domains

Sequence
Reduced product
Cartesian product
Universal
C specific
Python specific



C configuration



Python configuration

Sharing by delegation – Smashing Lists

Python List Abstraction

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

$$\mathbb{E}^\sharp[\![\, [e_1, \dots, e_n]^{loc} \,]\!]\, s \stackrel{\text{def}}{=}$$

$$\quad \text{let } \mathrm{s}, @ = \mathbb{E}^\sharp[\![\, \mathrm{alloc}(\mathbf{List}\ loc) \,]\!]\, s \text{ in}$$

$$\quad \text{let contents} = \text{mk\_var } @ \text{ "contents" in}$$

$$\quad \text{let length} = \text{mk\_var } @ \text{ "length" in}$$

$$\quad \mathbb{S}^\sharp[\![\, \text{length} = n \,]\!] \circ \mathbb{S}^\sharp[\![\, \text{contents} \stackrel{\text{weak}}{=} e_n \,]\!] \circ \dots \circ \mathbb{S}^\sharp[\![\, \text{contents} \stackrel{\text{weak}}{=} e_1 \,]\!] \mathrm{s}, @$$

Sharing by delegation

Demo time with `l = [1, 2, 3]`

Debugging and profiling using hooks

Hooks as observers of the analysis

Idea

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

hook signature

```
module type STATELESS_HOOK =
sig
  val name : string
  val init : 'a ctx -> unit

  val on_before_exec : stmt -> ('a,'a) man  -> 'a flow -> unit
  val on_after_exec :  stmt -> ('a,'a) man -> 'a flow -> ('a, unit) cases -> unit

  val on_before_eval : expr -> ('a,'a) man -> 'a flow -> unit
  val on_after_eval : expr -> ('a,'a) man -> 'a flow -> ('a, expr) cases -> unit

  val on_finish : ('a,'a) man -> 'a flow -> unit
end
```

Example of hooks: Logs

Logs

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

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

Example of hooks: Coverage

Coverage

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

```
def sum(l):
  s = 0
  for x in l:
    s += x
  return s

r2 = sum(['a', 'b', 'c'])
r1 = sum([1, 2, 3])
```

Example of hooks: Profiling

Motivation

- `perf`, `memtrace` too low-level and global
- Higher-level view by profiling at the analyzed program's level
- Inlining and nested loops $\implies$ analysis time $\not\propto$ program size

```python
def p(l1, l2):
    r = []
    for x in l1:
        for y in l2:
            r.append((x, y))
    return r

r1 = p([1,2,3], [4,5,6])
r2 = p(['a', 'b'], ['c', 'd'])
```

Loop Profiler

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

Function Profiler

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

Current Results

C Analysis – Results obtained by Abdelraouf and Antoine[1]

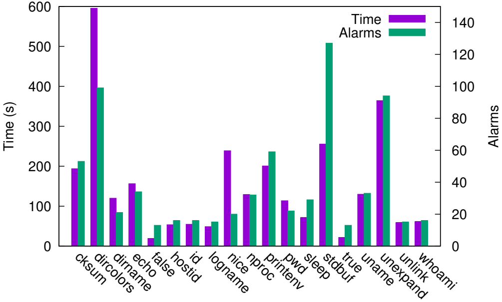
NIST Juliet:

CWE	Lines	Time (h:m:s)	✅	⚠️
Stack-based Buffer Overflow	234k	00:59:12	89%	11%
Heap-based Buffer Overflow	174k	00:37:12	86%	14%
Buffer Underwrite	93k	00:18:28	86%	14%
Buffer Over-read	75k	00:14:45	85%	15%
Buffer Under-read	89k	00:18:26	87%	13%
Integer Overflow	440k	01:24:47	52%	48%
Integer Underflow	340k	01:02:27	55%	45%
Divide By Zero	109k	00:13:17	55%	45%
Double Free	17k	00:04:21	100%	0%
Use After Free	14k	00:02:40	100%	0%
Illegal Pointer Subtraction	1k	00:00:24	100%	0%
NULL Pointer Dereference	21k	00:04:53	100%	0%

✅ Good case safe **and** 1 error in bad case.

⚠️ Good case unsafe **or** many errors in bad case.

19 programs from GNU Coreutils:



[1] Ouadjaout and Miné. "A Library Modeling Language for the Static Analysis of C Programs". SAS 2020.

Python Analysis

Name	LOC	Type Analysis					Value Analysis				
		Time	Mem.	Exceptions detected			Time	Mem.	Exceptions detected		
				Type	Index	Key			Type	Index	Key
scimark	416	1.4s	12MB	1	1	0	3.4s	27MB	1	**0**	0
richards	426	13s	112MB	1	4	0	17s	149MB	1	**2**	0
unpack	458	8.3s	7MB	0	0	0	9.4s	6MB	0	0	0
go	461	27s	345MB	33	20	0	2.0m	1.4GB	33	20	0
hexiom	674	1.1m	525MB	0	46	3	4.7m	3.2GB	0	**21**	3
regex	1792	23s	18MB	0	2053	0	1.3m	56MB	0	**145**	0
process	1417	10s	64MB	7	7	1	12s	85MB	7	**4**	1
choose	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

Monat, Ouadjaout, and Miné. "Static Type Analysis by Abstract Interpretation of Python Programs". ECOOP 2020
Monat, Ouadjaout, and Miné. "Value and allocation sensitivity in static Python analyses". SOAP@PLDI 2020

Didn’t you say *multi-lingual*?

Combining C and Python – Motivation

Early work

Motivation

- Some Python libraries = C code + Python wrapper
- How to analyze programs using those libraries?
  - Ignore calls

[2] `https://github.com/python/typeshed/`

Combining C and Python – Motivation

Early work

Motivation

- Some Python libraries = C code + Python wrapper
- How to analyze programs using those libraries?
  - Ignore calls
  - Use stubs

[2] https://github.com/python/typeshed/

Combining C and Python – Motivation

Early work

Motivation

- Some Python libraries = C code + Python wrapper
- How to analyze programs using those libraries?
  - Ignore calls
  - Use stubs
    - Type annotations[2]

[2] https://github.com/python/typeshed/

Combining C and Python – Motivation

Early work

Motivation

- Some Python libraries = C code + Python wrapper
- How to analyze programs using those libraries?
  - Ignore calls
  - Use stubs
    - Type annotations[2]
    - Manual work

[2] https://github.com/python/typeshed/

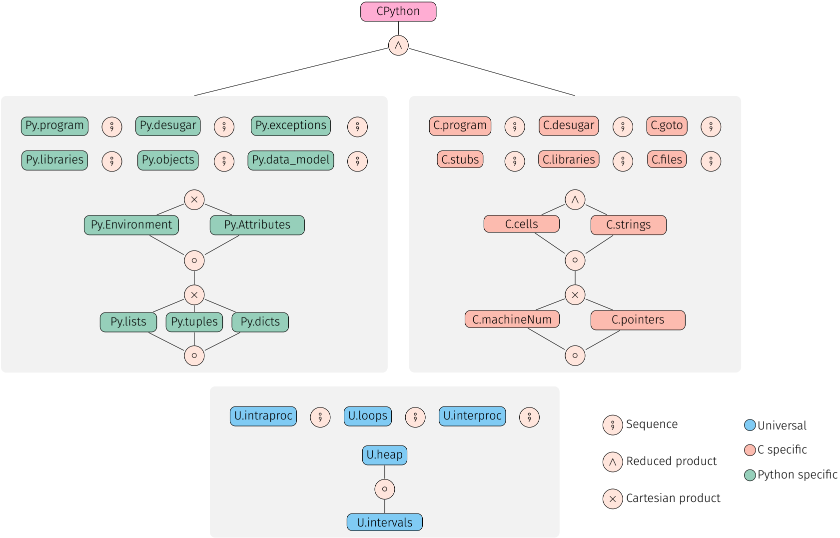
Combining C and Python – First Look

A combined static analysis of C/Python

- Target: C extensions using the CPython API
- Goal: detect runtime errors (in C, Python, and the "glue")
- Observations
  - allocated objects are <u>shared</u> in the memory,
  - but each language has <u>different abstractions</u>

$\Rightarrow$ Share universal domains and synchronize abstractions

Combining C and Python – Configuration View



Combining C and Python – Counter Example

counter.c

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

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

    self->counter += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

Combining C and Python – Counter Example

counter.c

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

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

    self->counter += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

$\Rightarrow$ Demo!

Combining C and Python – Counter Example

counter.c

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

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

    self->counter += i;
    Py_RETURN_NONE;
}

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

count.py

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

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

- $\texttt{power} \leq 30 \Rightarrow \texttt{r} = 2^{\texttt{power}}$
- $\texttt{power} = 31 \Rightarrow \texttt{r} = -2^{31}$
- $32 \leq \texttt{power} \leq 62$: OverflowError: signed integer is greater than maximum
- $\texttt{power} \geq 63$: OverflowError: Python int too large to convert to C long

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P      t*
```

count.py

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

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

C

Pointers

Universal

Heap (Recency)

Intervals

Python

Attributes

Environment

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P t*
```

count.py

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

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

C

Pointers

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s
Intervals

Python

Attributes

Environment

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s
Intervals

Python

Attributes

Environment

Combining C and Python – Counter Example – State

counter.c

```
typedef struct {
  PyObject_HEAD;
  int counter;
} Counter;

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P   t*
```

count.py

```
from counter import Counter
from random import randrange

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s
Intervals

Python

Attributes
@CounterCls:s ⇀ {get, incr}

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P      t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s
Intervals

Python

Attributes
@CounterCls:s ⇀ {get, incr}

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P          t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s **@I{CounterCls}:s**
Intervals

Python

Attributes
@CounterCls:s ⇀ {get, incr}

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P   t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}
⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s @I{CounterCls}:s
Intervals
⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]

Python

Attributes
@CounterCls:s ⇀ {get, incr}

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P      t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}
⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s @I{CounterCls}:s
Intervals
⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]

Python

Attributes
@CounterCls:s ⇀ {get, incr}
@I{CounterCls}:s ⇀ ∅

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}
c ⇀ **{@I{CounterCls}:s}**

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P t*
```

count.py

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

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

C

```
Pointers
 ⟨CounterCls,8,ptr⟩ : {PyType_Type}
 ⟨CounterCls,232,ptr⟩ : {Counter_methods}
 ⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}
```

Universal

```
Heap (Recency)
 @CounterCls:s @CounterIncr:s
 @CounterGet:s @I{CounterCls}:s
Intervals
 ⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]
```

Python

```
Attributes
 @CounterCls:s ⇀ {get, incr}
 @I{CounterCls}:s ⇀ ∅

Environment
 Counter ⇀ {@CounterCls:s}
 @CounterCls:s·get ⇀
 {@c function CounterGet:s}
 @CounterCls:s·incr ⇀
 {@c function CounterIncr:s}
 c ⇀ {@I{CounterCls}:s}
```

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}
⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s @I{CounterCls}:s **@I{int}:w**
Intervals
⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]
power ⇀ [0,127]

Python

Attributes
@CounterCls:s ⇀ {get, incr}
@I{CounterCls}:s ⇀ ∅

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}
c ⇀ {@I{CounterCls}:s}
power ⇀ {@I{int}:w}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P        t*
```

count.py

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

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

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}
⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s
@CounterGet:s @I{CounterCls}:s @I{int}:w
Intervals
⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]
power ⇀ [0,127]

Python

Attributes
@CounterCls:s ⇀ {get, incr}
@I{CounterCls}:s ⇀ ∅

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}
c ⇀ {@I{CounterCls}:s}
power ⇀ {@I{int}:w}

Combining C and Python – Counter Example – State

counter.c

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

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

  self->counter += i;
  Py_RETURN_NONE;
}

static P        t*
```

count.py

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

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

Python

Attributes
@CounterCls:s ⇀ {get, incr}
@I{CounterCls}:s ⇀ ∅

Environment
Counter ⇀ {@CounterCls:s}
@CounterCls:s·get ⇀
{@c function CounterGet:s}
@CounterCls:s·incr ⇀
{@c function CounterIncr:s}
c ⇀ {@I{CounterCls}:s}
power ⇀ {@I{int}:w}
@tuple[1]:s·[0] ⇀ {@I{int}:w}

C

Pointers
⟨CounterCls,8,ptr⟩ : {PyType_Type}
⟨CounterCls,232,ptr⟩ : {Counter_methods}
⟨@I{CounterCls}:s,8,ptr⟩ : {CounterCls}
args : {@tuple[1]:s}
self : {@I{CounterCls}:s}

Universal

Heap (Recency)
@CounterCls:s @CounterIncr:s **@tuple[1]:s**
@CounterGet:s @I{CounterCls}:s @I{int}:w
Intervals
⟨@I{CounterCls}:s,16,s32}⟩ ⇀ [0,0]
power ⇀ [0,127]
@tuple[1]:s·[0] ⇀ [0,$2^{127}-1$]

Combining C and Python – Current analyses

Project	|C|	|Py|	|Tests|				Assertions
python-llist	2800	1600	171/194	6.5m	3275/3332	228	235/695
pyahocorasick	3895	1287	66/97	2.0m	1580/1670	34	41/89

Conclusion

Conclusion

Mopsa

- Compositional, flexible architecture

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Future work

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Future work

- C/Python analysis

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Future work

- C/Python analysis
- Sequence of analyses

Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Future work

- C/Python analysis
- Sequence of analyses
- Backward analysis

Conclusion

Mopsa

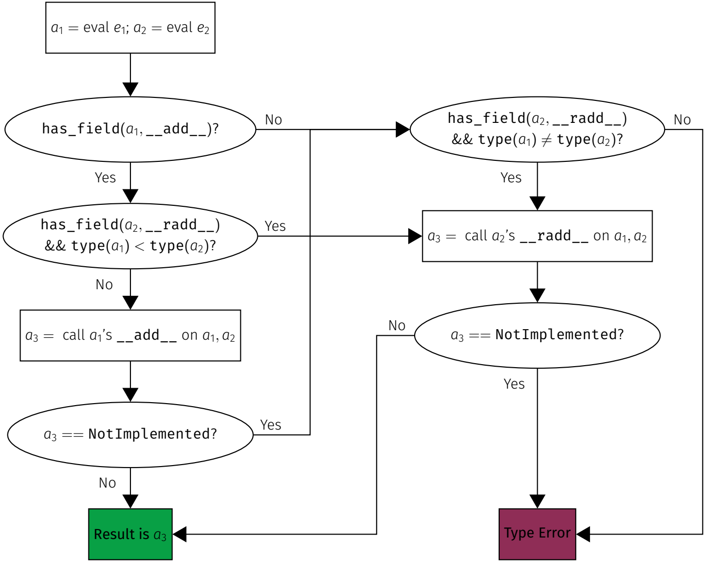
- Compositional, flexible architecture
- Supports different languages
  - C analysis: Coreutils, Juliet suite
  - Python analysis: pyperformance-benchmarks, PathPicker

Future work

- C/Python analysis
- Sequence of analyses
- Backward analysis

`gitlab.com/mopsa`

Python's Semantics – Example: $e_1 + e_2$



Python's Semantics – Example: $e_1 + e_2$

$$
\begin{array}{l}
\mathbb{E}[\![\, e_1 + e_2 \,]\!]\,(f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq \mathit{cur} \text{ then } (f, \epsilon, \sigma) \text{ else} \\
\quad \text{letif } (f, \epsilon, \sigma, a_1) = \mathbb{E}[\![\, e_1 \,]\!]\,(f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f, \epsilon, \sigma, a_2) = \mathbb{E}[\![\, e_2 \,]\!]\,(f, \epsilon, \sigma) \text{ in} \\
\quad \text{if } \mathit{hasattr}(\sigma(a_1), \_\_\mathit{add}\_\_) \text{ then} \\
\quad\quad \text{if } \mathit{hasattr}(\sigma(a_2), \_\_\mathit{radd}\_\_) \wedge \mathit{type}(a_1) < \mathit{type}(a_2) \text{ then} \\
\quad\quad\quad \text{letif } (f, \epsilon, \sigma, a_r) = \mathbb{E}[\![\, a_2.\_\_\mathit{radd}\_\_(a_1) \,]\!] \text{ in} \\
\quad\quad\quad \text{if } \sigma(a_r) = \texttt{NotImpl} \text{ then } \mathit{empty\_addr} \circ \mathbb{S}[\![\, \texttt{raise TypeError} \,]\!](f, \epsilon, \sigma) \\
\quad\quad\quad \text{else } (f, \epsilon, \sigma, a_r) \\
\quad\quad \text{else letif } (f, \epsilon, \sigma, a_r) = \mathbb{E}[\![\, a_1.\_\_\mathit{add}\_\_(a_2) \,]\!] \text{ in} \\
\quad\quad\quad \text{if } \sigma(a_r) = \texttt{NotImpl} \text{ then} \\
\quad\quad\quad\quad \text{if } \mathit{hasattr}(\sigma(a_2), \_\_\mathit{radd}\_\_) \wedge \mathit{type}(a_1) \neq \mathit{type}(a_2) \text{ then} \\
\quad\quad\quad\quad\quad \text{letif } (f, \epsilon, \sigma, a_r) = \mathbb{E}[\![\, a_2.\_\_\mathit{radd}\_\_(a_1) \,]\!] \text{ in} \\
\quad\quad\quad\quad\quad \text{if } \sigma(a_r) = \texttt{NotImpl} \text{ then } \mathit{empty\_addr} \circ \mathbb{S}[\![\, \texttt{raise TypeError} \,]\!](f, \epsilon, \sigma) \\
\quad\quad\quad\quad\quad \text{else } (f, \epsilon, \sigma, a_r) \\
\quad\quad\quad \text{else } (f, \epsilon, \sigma, a_r) \\
\quad \text{else if } \mathit{hasattr}(\sigma(a_2), \_\_\mathit{radd}\_\_) \wedge \mathit{type}(a_1) \neq \mathit{type}(a_2) \text{ then} \\
\quad\quad \text{letif } (f, \epsilon, \sigma, a_r) = \mathbb{E}[\![\, a_2.\_\_\mathit{radd}\_\_(a_1) \,]\!] \text{ in} \\
\quad\quad \text{if } \sigma(a_r) = \texttt{NotImpl} \text{ then } \mathit{empty\_addr} \circ \mathbb{S}[\![\, \texttt{raise TypeError} \,]\!](f, \epsilon, \sigma) \\
\quad\quad \text{else } (f, \epsilon, \sigma, a_r) \\
\quad \text{else } \mathit{empty\_addr} \circ \mathbb{S}[\![\, \texttt{raise TypeError} \,]\!](f, \epsilon, \sigma)
\end{array}
$$