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

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

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

```
argslen.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;
}
```

Specifications of the analyzer

**Infer** run-time errors (or other semantic properties)

**Automatic** no expert knowledge required

**Sound** cover all possible executions

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

No alarm
How does an abstract interpreter work?

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

```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 $\leadsto$ Abstract state $\leadsto$ Semantic property (alarms)
- Combine abstract domains to gain precision

```python
# sum_indexed.py

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 $\leadsto$ Abstract state $\leadsto$ Semantic property (alarms)
- Combine abstract domains to gain precision

```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 $[1, 2, 3]$:

- $\text{len}(l) = 3$
- $0 \leq i < 3$
- $s \geq 0$
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

```python
# sum_indexed.py

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]
len(l) = 3  \{ valid list accesses \}
0 \leq i < 3
s \geq 0

# Call with ['a', 'b', 'c']
l : List[str]
s : int
int + str \{ 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

1. Introduction
2. Multi-language support from the AST
3. Flexible domains
4. Debugging and profiling using hooks
5. Current Results
6. Didn’t you say multi-lingual?
7. Conclusion
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

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, target

Universal.Iterators.Loops

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

Computes fixpoint using widening
Cooperation by delegation

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

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

```
init;
```

```
clean init
```

```
C.iterators.loops
Rewrite and analyze recursively
```

```
Python.Desugar.Loops
```

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

```
```

Cooperation by delegation

for(init; cond; incr) body

Rewrite and analyze recursively

C.iterators.loops

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

for target in iterable: body

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, target

Universal.Iterators.Loops

Matches while(...){...}
Computes fixpoint using widening
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 recursively
  - Optimize for some semantic 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
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 recursively
◦ 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 recursively
◦ Optimize for some semantic 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
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.

\[
\begin{align*}
\mathbb{E}^\#\left[ [e_1, \ldots, e_n]^{loc} \right] \ s \ &= \ \\
\text{let} \ s, @ = \mathbb{E}^\#[\text{alloc(}\text{List}\ loc\text{)}] \ s \ \text{in} \ \\
\text{let} \ \text{contents} = \text{mk\_var} \ @ \ "\text{contents}\" \ \text{in} \ \\
\text{let} \ \text{length} = \text{mk\_var} \ @ \ "\text{length}\" \ \text{in} \ \\
\mathbb{S}^\#[\text{length} = n] \circ \mathbb{S}^\#[\text{contents} \ \text{weak} \ e_n] \circ \ldots \circ \mathbb{S}^\#[\text{contents} \ \text{weak} \ e_1] \ s, @
\end{align*}
\]
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

```ocaml
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

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

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

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

r2 = sum(['a', 'b', 'c'])
r1 = sum([1, 2, 3])
```
Example of hooks: Profiling

Motivation

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

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

Loop Profiler

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

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

Function Profiler

```
p  0.0544s(total) \times 2
```

Current Results
# C Analysis – Results obtained by Abdelraouf and Antoine

## NIST Juliet:

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

- ✔️ Good case safe **and** 1 error in bad case.
- 🔄 Good case unsafe **or** many errors in bad case.

## 19 programs from GNU Coreutils:

![Graph showing time and alarms for GNU Coreutils programs](image)

---

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Type Analysis</th>
<th></th>
<th></th>
<th></th>
<th>Value Analysis</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Mem.</td>
<td>Exceptions detected</td>
<td>Key</td>
<td>Time</td>
<td>Mem.</td>
<td>Exceptions detected</td>
<td>Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Type</td>
<td>Index</td>
<td>Key</td>
<td></td>
<td></td>
<td>Type</td>
</tr>
<tr>
<td>scimark</td>
<td>416</td>
<td>1.4s</td>
<td>12MB</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3.4s</td>
<td>27MB</td>
<td>1</td>
</tr>
<tr>
<td>richards</td>
<td>426</td>
<td>13s</td>
<td>112MB</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>17s</td>
<td>149MB</td>
<td>1</td>
</tr>
<tr>
<td>unpack</td>
<td>458</td>
<td>8.3s</td>
<td>7MB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9.4s</td>
<td>6MB</td>
<td>0</td>
</tr>
<tr>
<td>go</td>
<td>461</td>
<td>27s</td>
<td>345MB</td>
<td>33</td>
<td>20</td>
<td>0</td>
<td>2.0m</td>
<td>1.4GB</td>
<td>33</td>
</tr>
<tr>
<td>hexiom</td>
<td>674</td>
<td>1.1m</td>
<td>525MB</td>
<td>0</td>
<td>46</td>
<td>3</td>
<td>4.7m</td>
<td>3.2GB</td>
<td>0</td>
</tr>
<tr>
<td>regex</td>
<td>1792</td>
<td>23s</td>
<td>18MB</td>
<td>0</td>
<td>2053</td>
<td>0</td>
<td>1.3m</td>
<td>56MB</td>
<td>0</td>
</tr>
<tr>
<td>process</td>
<td>1417</td>
<td>10s</td>
<td>64MB</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td>12s</td>
<td>85MB</td>
<td>7</td>
</tr>
<tr>
<td>choose</td>
<td>2562</td>
<td>1.1m</td>
<td>1.6GB</td>
<td>12</td>
<td>22</td>
<td>7</td>
<td>2.9m</td>
<td>3.7GB</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>9294</td>
<td>4.0m</td>
<td>2.8GB</td>
<td>59</td>
<td>2214</td>
<td>12</td>
<td>13m</td>
<td>9.1GB</td>
<td>59</td>
</tr>
</tbody>
</table>

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

2https://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

²https://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/
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 shared in the memory,
  - but each language has different abstractions

⇒ Share universal domains and synchronize abstractions
Combining C and Python – Configuration View

- CPython
- Py.program
- Py.desugar
- Py.exceptions
- Pylibraries
- Py.objects
- Py.data_model
- Py.Environment
- Py.Attributes
- Py.lists
- Py.tuples
- Py.dicts

- C.program
- C.desugar
- C.goto
- C.stubs
- C.libraries
- C.files
- C.cells
- C.strings
- C.machineNum
- C.pointers

- U.intraproc
- U.loops
- U.interproc
- U.heap
- U.intervals

Sequence
Reduced product
Cartesian product
Universal
C specific
Python specific

18
Combining C and Python – Counter Example

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);
}

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

```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);
}
```

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

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

⇒ Demo!

创造性的例子展示了C和Python的结合。```counter.c```文件包含一个名为Counter的结构体和两个静态方法CounterIncr和CounterGet。```count.py```文件导入了Counter，然后使用Python的random库生成一个随机数，然后对其进行操作，并打印结果。```Demo!```部分显示了不同的结果，根据输入的整数范围和**操作。
Combining C and Python – Counter Example

counter.c

typestruct {
    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

from counter import Counter
from random import randrange

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

▶ power ≤ 30 ⇒ r = 2^{power}
▶ power = 31 ⇒ r = -2^{31}
▶ 32 ≤ power ≤ 62: OverflowError: signed integer is greater than maximum
▶ power ≥ 63: OverflowError: Python int too large to convert to C long
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 PyObject*
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->counter);
}

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()
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 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 – 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;
}
```

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()
```
**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

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

**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

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

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}

Environment
Counter → {@CounterCls:s}
@CounterCls:s.get →
 advisers CounterGet:s
@CounterCls:s.incr →
 advisers 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 PyObject *
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->counter);
}
```

count.py

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

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

C

Pointers

- `{CounterCls,8,ptr} : {PyType_Type}`
- `{CounterCls,232,ptr} : {Counter_methods}`

C

Heap (Recency)

- `{CounterCls} : s
  `{CounterIncr} : s
  `{CounterGet} : `I{CounterCls} : s

Intervals

Python

Attributes

- `{CounterCls} : s → {get, incr}`

Environment

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

count.py

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

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

```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 – State

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);
}

from counter import Counter
from random import randint

c = Counter()
power = randint(128)
c.incr(2**power-1)
c.incr()
r = c.get()
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 PyObject *
CounterGet(Counter *self)
{
    return Py_BuildValue("i", self->counter);
}

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()
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);
}

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

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()
## Combining C and Python – Current analyses

<table>
<thead>
<tr>
<th>Project</th>
<th>C</th>
<th>Py</th>
<th>Tests</th>
<th>Clock</th>
<th>Exclamation-Triangle</th>
<th>Exclamation-Triangle</th>
<th>Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>python-llist</td>
<td>2800</td>
<td>1600</td>
<td>171/194</td>
<td>6.5m</td>
<td>3275/3332</td>
<td>228</td>
<td>235/695</td>
</tr>
<tr>
<td>pyahocorasick</td>
<td>3895</td>
<td>1287</td>
<td>66/97</td>
<td>2.0m</td>
<td>1580/1670</td>
<td>34</td>
<td>41/89</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

Mopsa

- Compositional, flexible architecture
Conclusion

Mopsa

- Compositional, flexible architecture
- Supports different languages

Future work

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

gitlab.com/mopsa
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: $e_1 + e_2$

$\text{a}_1 = \text{eval } e_1; \text{a}_2 = \text{eval } e_2$

- **has_field($\text{a}_1$, __add__)?**
  - **Yes**
  - **has_field($\text{a}_2$, __radd__) && type($\text{a}_1$) < type($\text{a}_2$)?**
    - **Yes**
      - $\text{a}_3 = \text{call } \text{a}_1$’s __add__ on $\text{a}_1, \text{a}_2$
    - **No**
      - $\text{a}_3 = \text{call } \text{a}_1$’s __add__ on $\text{a}_1, \text{a}_2$
    - **a}_3 == \text{NotImplemented?**
      - **Yes**
      - **Type Error**
      - **No**
        - Result is $\text{a}_1$
Python’s Semantics – Example: $e_1 + e_2$

$$E[e_1 + e_2] (f, \epsilon, \sigma) \equiv$$

if $f \neq \text{cur}$ then $(f, \epsilon, \sigma)$ else

let if $(f, \epsilon, \sigma, a_1) = E[e_1] (f, \epsilon, \sigma)$ in

let if $(f, \epsilon, \sigma, a_2) = E[e_2] (f, \epsilon, \sigma)$ in

if hasattr($\sigma(a_1)$, '__add__') then

if hasattr($\sigma(a_2)$, '__radd__') and type($a_1$) < type($a_2$) then

let if $(f, \epsilon, \sigma, a_r) = E[a_2.__radd__(a_1)]$ in

if $\sigma(a_r) = \text{NotImpl}$ then empty_addr $\circ S[\text{raise TypeError}](f, \epsilon, \sigma)$

else $(f, \epsilon, \sigma, a_r)$

else let if $(f, \epsilon, \sigma, a_r) = E[a_1.__add__(a_2)]$ in

if $\sigma(a_r) = \text{NotImpl}$ then empty_addr $\circ S[\text{raise TypeError}](f, \epsilon, \sigma)$

else $(f, \epsilon, \sigma, a_r)$

else $(f, \epsilon, \sigma)$$

else if hasattr($\sigma(a_2)$, '__radd__') and type($a_1$) \neq type($a_2$) then

let if $(f, \epsilon, \sigma, a_r) = E[a_2.__radd__(a_1)]$ in

if $\sigma(a_r) = \text{NotImpl}$ then empty_addr $\circ S[\text{raise TypeError}](f, \epsilon, \sigma)$

else $(f, \epsilon, \sigma, a_r)$

else $(f, \epsilon, \sigma)$

else empty_addr $\circ S[\text{raise TypeError}](f, \epsilon, \sigma)$