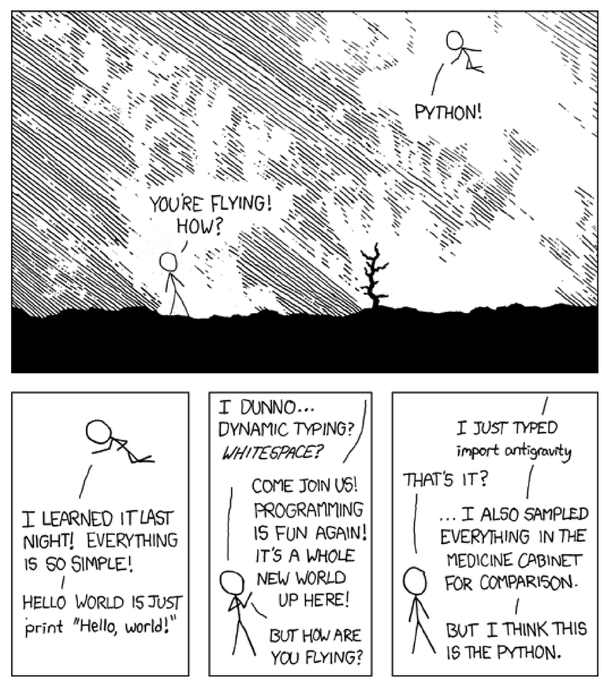
Semantics & Static Analysis of Python Programs★

Raphaël Monat, Abdelraouf Ouadjaout, Antoine Miné

raphael.monat@lip6.fr LIP6, Sorbonne Université



What is Python?

Python is an object-oriented, interpreted, dynamic programming language. It features a powerful and permissive, high-level syntax; and ranks 2nd in popularity on GitHub.

Semantics?

What? A *mathematical* description of the meaning of Python operators.

Why? To relate static analyses with the actual program behavior.

Specificities of Python

Runtime Errors

```
y = int(input())
try:
 x = 1 / y
except DivisionByZeroError:
 print("errors are catchable")
```

Variable Scope

```
a = 2
def f():
 z = a
 a = 1+z
f()
# f raises an UnboundLocalError
```

Dynamic Typing

Variables may have different type at different program locations:

```
if *: z = 3
else: z = "a"
```

Introspection

Combined with dynamic typing, the control-flow can depend on the types:

```
def dint(x):
 if isinstance(x, int):
  return x*2
 else: raise TypeError
z = f('a')
```

Object Mutability

```
class A:
  def __init__(self):
    self.val = 0
  def update(self, x):
    self.val = x

x = A()
c = x.val
y = x
y.update('a')
z = x.val
# z = 'a'
```

Static Type Analysis

Goal

- Detect potential run-time errors without executing programs.
- Automatic analysis: no expert knowledge needed.
- Sound analysis: no error found means no runtime error.

We use the Abstract Interpretation framework[1].

Motivation

- Static analyses are widespread for statically-typed programming languages, and successfully used in critical software certification.
- Dynamic programming languages leave less information in the syntax.
- Thus, semantic static analyses would be most valuable in this setting.

Implementation & Benchmarks

- Implementation into MOPSA[5], whose goal is to provide modular analyses.
- Type analysis: 2500 lines.
- Container abstraction: 2100 lines.
- Python's Semantics: 5500 lines.
- We are able to analyze some offical Python benchmarks[6]!

Future work

- Stable, easily maintainable and checkable concrete semantics.
- Handle libraries through automatic stub generation and multilingual analysis (most libraries are in C).
- Summary-based function analysis, where the summaries can be reused in different contexts.
- Analyze real-world programs and frameworks (Django, SageMath, ...)

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

p = "/dev" if random() else Path() ●

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

r = fspath(p) ●
```

After `p = ...`: $\begin{cases} p \mapsto \{@_{str}, @_{Path}\} \\ @_{Path} \mapsto \{\_\_fspath\_\_ \mapsto @_{int}, \emptyset\} \end{cases}$

After `if isinstance(p, str):`: $p \mapsto \{@_{str}\}$

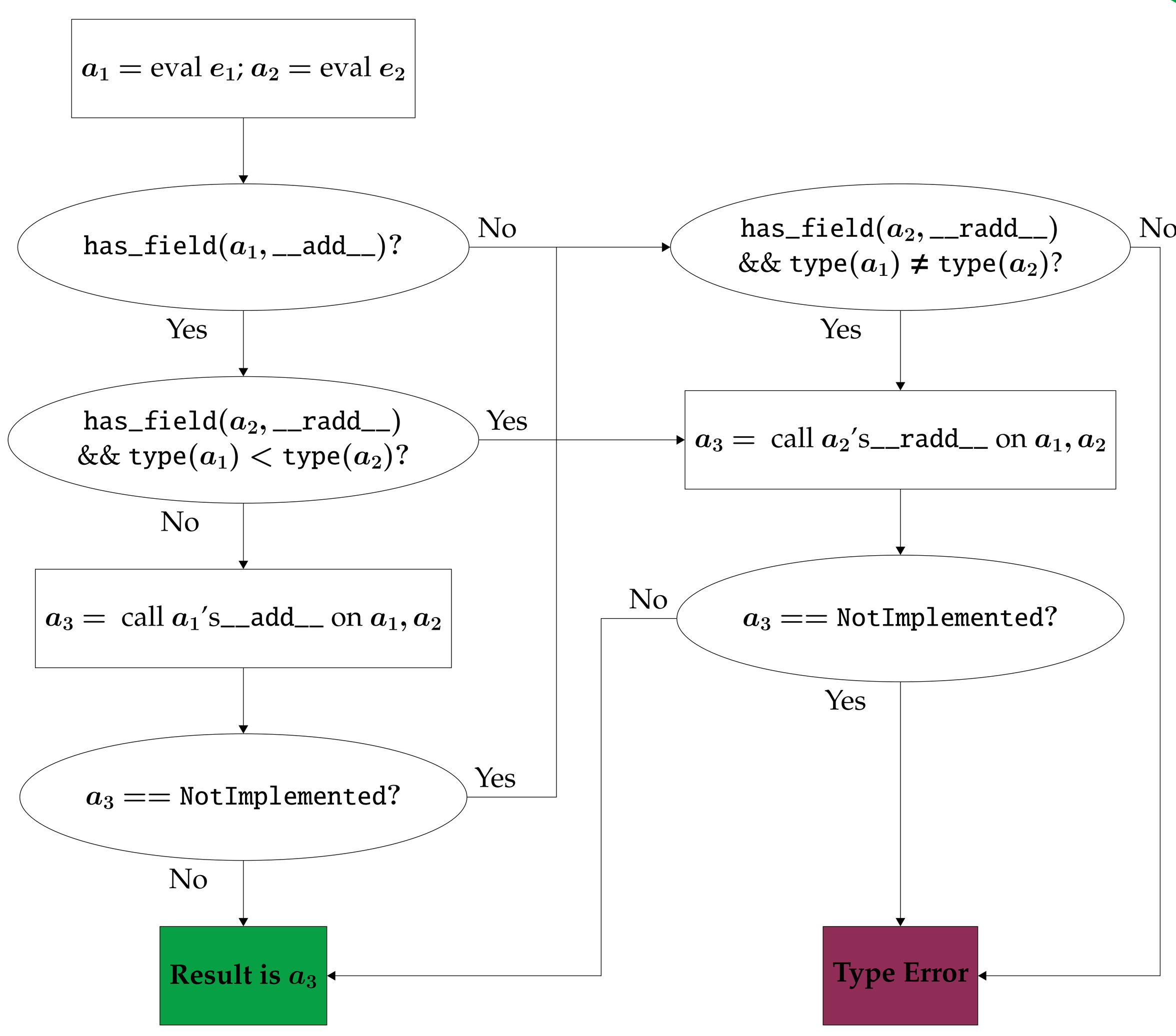
After `r = p.__fspath__()`: $\begin{cases} p \mapsto \{@_{path}\}; r \mapsto \{@_{int}\} \\ @_{Path} \mapsto \{\_\_fspath\_\_ \mapsto @_{int}, \emptyset\} \end{cases}$

After `r = fspath(p)`: $\text{TypeError} \vee r \mapsto \{@_{str}\}$

Name	LOC	Time (inlining)	Time (fun. cache)	# Alarms	# False Alarms
fannkuch.py	59	0.07s	0.07s	0	0
float.py	63	0.10s	0.06s	0	0
spectral_n.py	74	3.9 s	0.33s	0	1
nbody.py	157	2.6s	1.5s	0	1
chaos.py	324	19s	5.9s	**1** [7]	0
unpack_seq.py	458	5.6s	5.4s	0	0
hexiom.py	674	61.7m	2.2m	0	52

Semantics Example: Computing $e_1 + e_2$

Python's efficient and concise syntax entails the semantics to be **as accommodating as possible rather than raise exceptions**, creating many cases for operators as simple the addition. Our current semantics is an **input-output semantics** which is defined on paper, and whose abstract version is implemented in our static analyzer, updated from [2].



Guess the result – My favorite Python game

```
[] and 'a'
127 is 127
128 is 128
"a" is "a"
"a,1" is "a,1"
```

```
l = list(range(10))
for x in l:
 l.remove(x)
print(l)
```

```
d = {0: 'a'}
for i in d:
 d.pop(i)
 d[i+1+len(d)]='a'
 print(i)
```

Semantics Challenges

Uncovering the semantics By reading the documentation and the implementation.

Checking the semantics is correct

- By writing tests and comparing the results with the interpreter;
- By checking that our analysis passes the interpreter's unit tests.

Other approaches Coq[4], K framework[3] are attractive tools (to extract a concrete interpreter, or to be able to write proofs), but their use would be time-consuming.

[1] Cousot and Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *POPL 1977*. 1977. DOI: 10.1145/512950.512973.

[2] Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". In: *NFM 2018 Proceedings*. Vol. 10811. 2018, pp. 185–202. DOI: 10.1007/978.3.319.77935.5.14.

[3] Grigore Roşu and Traian Florin Şerbănuţă. "An Overview of the K Semantic Framework". In: *Journal of Logic and Algebraic Programming* 79.6 (2010), pp. 397–434. DOI: 10.1016/j.jlap.2010.03.012.

[4] The Coq Development Team. *The Coq Proof Assistant, version 8.9.0*. Jan. 2019. DOI: 10.5281/zenodo.2554024.

[5] *MOPSA Project*. http://mopsa.lip6.fr.

[6] *Python Performance Benchmarks*. https://github.com/python/pyperformance/tree/master/pyperformance/benchmarks.

[7] *Python Performance Benchmarks Bug Report*. https://github.com/python/pyperformance/issues/57.

★ This work is supported by the European Research Council under Consolidator Grant Agreement 681393 – MOPSA.

LaTeX TikZposter