Static Analysis by Abstract Interpretation of Dynamic Programming Languages

Raphaël Monat

23rd October 2018

PhD student (started 1st September 2018), APR team, LIP6, Sorbonne Université
Under the supervision of Antoine Miné

Introduction

Static Analysis

- Detect bugs in programs
- Without having to execute them

Static Analysis

- Detect bugs in programs
- Without having to execute them

. . . by Abstract Interpretation

- Keep possible environments using overapproximations
- Sound analysis: if no bug is detected, no bug will occur
- Automatic analysis: no interaction with expert user needed

Dynamic Programming Languages

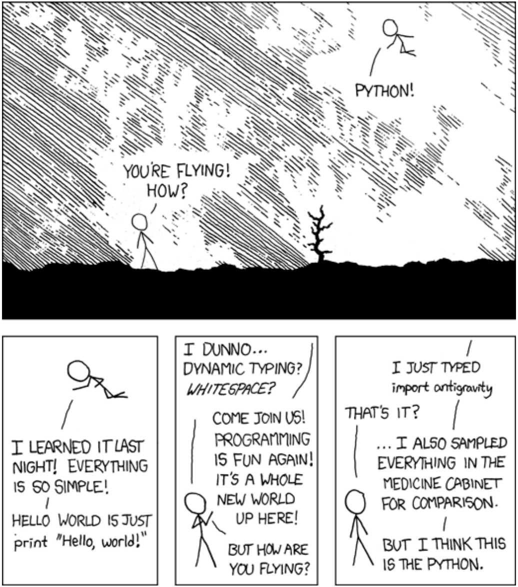
- Simple syntax, high-level features
- Dynamic typing: types are only known at runtime
- Introspection
- Self-modification
- Eval

Examples: JavaScript, Python, . . .

Static analyses are especially helpful – and difficult – on dynamic programming languages.

Python

Why?



Source: xkcd (`https://xkcd.com/353/`)

Semantics of Python

- ✗ No standard (and no formal semantics) defining Python.
- CPython's implementation is the definition.
- We use a slight modification of the semantics of Fromherz, Ouadjaout, and Miné[1].

[1] Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings.

Semantics of +

$$\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=}$$

$\quad$ if $f \neq cur$ then $(f, \epsilon, \sigma, addr_{\text{None}})$ else

$\quad$ letif $(f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma)$ in

$\quad$ letif $(f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1)$ in

$\quad$ if $has\_field(a_1, \_\_add\_\_, \sigma_2)$ then

$\quad\quad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then

$\quad\quad\quad$ if $has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2)$ then

$\quad\quad\quad\quad$ letif $(f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3)$ in

$\quad\quad\quad\quad$ if $\sigma_4(a_4) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_4, \epsilon_4, \sigma_4)$

$\quad\quad\quad\quad$ else $(f_4, \epsilon_4, \sigma_4, a_4)$

$\quad\quad\quad$ else $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $f_3, \epsilon_3, \sigma_3, a_3$

$\quad$ else if $has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2$ then

$\quad\quad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $(f_3, \epsilon_3, \sigma_3, a_3)$

$\quad$ else $\texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

Evaluate e_1

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of $+$

$$
\begin{array}{l}
\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\text{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma) \text{ in} \\
\bullet\ \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1) \text{ in } \bullet \\
\quad \text{if } has\_field(a_1, \_\_add\_\_, \sigma_2) \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
\qquad\quad \text{if } has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2) \text{ then} \\
\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3) \text{ in} \\
\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
\quad \text{else if } has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2 \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

Evaluate e_2

slight modification from Fromherz, Ouadjaout, and Miné, “Static Value Analysis of Python Programs by Abstract Interpretation”

Semantics of +

$$\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=}$$

$\text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\text{None}}) \text{ else}$

$\text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma) \text{ in}$

$\text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1) \text{ in}$

● $\text{if } has\_field(a_1, \_\_add\_\_, \sigma_2) \text{ then}$ ●

$\quad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2) \text{ in}$

$\quad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then}$

$\quad\quad \text{if } has\_field(a_2, \_\_radd\_\_, \sigma_3) \land typeof(a_1) \neq typeof(a_2) \text{ then}$

$\quad\quad\quad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3) \text{ in}$

$\quad\quad\quad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4)$

$\quad\quad\quad \text{else } (f_4, \epsilon_4, \sigma_4, a_4)$

$\quad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad \text{else } f_3, \epsilon_3, \sigma_3, a_3$

$\text{else if } has\_field(a_2, \_\_radd\_\_, \sigma_2) \land typeof\, a_1 \neq typeof\, a_2 \text{ then}$

$\quad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2) \text{ in}$

$\quad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad \text{else } (f_3, \epsilon_3, \sigma_3, a_3)$

$\text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

If e_1 has method $\_\_add\_\_$

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of +

$\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=}$

and if $\_\_add\_\_$ does not return `NotImpl`

$$
\begin{array}{l}
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\texttt{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1) \text{ in} \\
\quad \text{if } has\_field(a_1, \_\_add\_\_, \sigma_2) \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \bullet \text{ if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \bullet \\
\qquad\quad \text{if } has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2) \text{ then} \\
\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3) \text{ in} \\
\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
\quad \text{else if } has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2 \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of +

return the result of the $\_\_add\_\_$ call

$$
\begin{array}{l}
\mathbb{E}[\![\, e_1 + e_2 \,]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\texttt{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![\, e_1 \,]\!](f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![\, e_2 \,]\!](f_1, \epsilon_1, \sigma_1) \text{ in} \\
\quad \text{if } has\_field(a_1, \_\_add\_\_, \sigma_2) \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_1.\_\_add\_\_(a_2) \,]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
\qquad\quad \text{if } has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2) \text{ then} \\
\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!](f_3, \epsilon_3, \sigma_3) \text{ in} \\
\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \bullet\ \text{else } f_3, \epsilon_3, \sigma_3, a_3 \bullet \\
\quad \text{else if } has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2 \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of $+$

otherwise, if e_1 and e_2 have different types

$$
\begin{array}{l}
\mathbb{E}[\![\, e_1 + e_2 \,]\!] (f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\text{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![\, e_1 \,]\!] (f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![\, e_2 \,]\!] (f_1, \epsilon_1, \sigma_1) \text{ in} \\
\quad \text{if } has\_field(a_1, \_\_add\_\_, \sigma_2) \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_1.\_\_add\_\_(a_2) \,]\!] (f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
\qquad\quad \bullet \text{ if } has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2) \text{ then } \bullet \\
\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!] (f_3, \epsilon_3, \sigma_3) \text{ in} \\
\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
\quad \bullet \text{ else if } has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2 \text{ then } \bullet \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!] (f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of +

$\mathbb{E}[\![\, e_1 + e_2 \,]\!]\,(f, \epsilon, \sigma) \stackrel{\text{def}}{=}$

$\quad$ if $f \neq cur$ then $(f, \epsilon, \sigma, addr_{\text{None}})$ else

$\quad$ letif $(f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![\, e_1 \,]\!]\,(f, \epsilon, \sigma)$ in

$\quad$ letif $(f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![\, e_2 \,]\!]\,(f_1, \epsilon_1, \sigma_1)$ in

$\quad$ if $has\_field(a_1, \_\_add\_\_, \sigma_2)$ then

$\quad\quad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_1.\_\_add\_\_(a_2) \,]\!]\,(f_2, \epsilon_2, \sigma_2)$ in

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then

$\quad\quad\quad$ if $has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2)$ then

$\quad\quad\quad\quad$ ● letif $(f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!]\,(f_3, \epsilon_3, \sigma_3)$ in ●

$\quad\quad\quad\quad$ if $\sigma_4(a_4) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_4, \epsilon_4, \sigma_4)$

$\quad\quad\quad\quad$ else $(f_4, \epsilon_4, \sigma_4, a_4)$

$\quad\quad\quad$ else $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $f_3, \epsilon_3, \sigma_3, a_3$

$\quad$ else if $has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2$ then

$\quad\quad$ ● letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, a_2.\_\_radd\_\_(a_1) \,]\!]\,(f_2, \epsilon_2, \sigma_2)$ in ●

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $(f_3, \epsilon_3, \sigma_3, a_3)$

$\quad$ else $\texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

call the reflected method __*radd*__

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Semantics of +

$\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=}$

$\quad$ if $f \neq cur$ then $(f, \epsilon, \sigma, addr_{\text{None}})$ else

$\quad$ letif $(f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma)$ in

$\quad$ letif $(f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1)$ in

$\quad$ if $has\_field(a_1, \_\_add\_\_, \sigma_2)$ then

$\qquad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\qquad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then

$\qquad\quad$ if $has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2)$ then

$\qquad\qquad$ letif $(f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3)$ in

$\qquad\qquad$ if $\sigma_4(a_4) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_4, \epsilon_4, \sigma_4)$

$\qquad\qquad$ else $(f_4, \epsilon_4, \sigma_4, a_4)$

$\qquad\quad$ else $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\qquad$ else $f_3, \epsilon_3, \sigma_3, a_3$

$\quad$ else if $has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2$ then

$\qquad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\qquad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\qquad$ else $(f_3, \epsilon_3, \sigma_3, a_3)$

$\quad$ else $\texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

raise a type error if nothing works

slight modification from Fromherz, Ouadjaout, and Miné, "Static Value Analysis of Python Programs by Abstract Interpretation"

Typing

fspath returns "the file system representation of the path."

Excerpt (and simplification) from Python's stdlib (os.py:1022).

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

Typing

fspath returns "the file system representation of the path."

Excerpt (and simplification) from Python's stdlib (`os.py`:1022).

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

Two kinds of typing:

- nominal typing (`isinstance`)
- duck typing (`hasattr`)

Typing

fspath returns "the file system representation of the path."

Excerpt (and simplification) from Python's stdlib (`os.py`:1022).

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

Two kinds of typing:

- nominal typing (`isinstance`)
- duck typing (`hasattr`)

Signature?

Typing

fspath returns "the file system representation of the path."

Excerpt (and simplification) from Python's stdlib (`os.py`:1022).

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

Two kinds of typing:

- nominal typing (`isinstance`)
- duck typing (`hasattr`)

Signature?

$\alpha \to \alpha,\ \alpha \in \{\, \textit{str}, \textit{bytes} \,\}$

Typing

fspath returns "the file system representation of the path."

Excerpt (and simplification) from Python's stdlib (`os.py`:1022).

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

Two kinds of typing:

- nominal typing (`isinstance`)
- duck typing (`hasattr`)

Signature?

$\alpha \to \alpha,\ \alpha \in \{\, \mathit{str}, \mathit{bytes} \,\}$

Objects having method *__fspath__*, itself returning type α

Typing +

$$
\begin{aligned}
&\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
&\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\texttt{None}}) \text{ else} \\
&\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma) \text{ in} \\
&\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1) \text{ in} \\
&\quad \text{if } \bullet has\_field(a_1, \_\_add\_\_, \sigma_2)\bullet \text{ then} \\
&\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
&\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
&\qquad\quad \text{if } \bullet has\_field(a_2, \_\_radd\_\_, \sigma_3)\bullet \land typeof(a_1) \neq typeof(a_2) \text{ then} \\
&\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3) \text{ in} \\
&\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
&\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
&\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
&\qquad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
&\quad \text{else if } \bullet has\_field(a_2, \_\_radd\_\_, \sigma_2)\bullet \land typeof\, a_1 \neq typeof\, a_2 \text{ then} \\
&\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2) \text{ in} \\
&\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
&\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
&\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{aligned}
$$

Need to know:

- attributes of e_1, e_2

Typing +

$\mathbb{E}[\![e_1 + e_2]\!](f, \epsilon, \sigma) \stackrel{\text{def}}{=}$

$\quad$ if $f \neq cur$ then $(f, \epsilon, \sigma, addr_{\texttt{None}})$ else

$\quad$ letif $(f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![e_1]\!](f, \epsilon, \sigma)$ in

$\quad$ letif $(f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![e_2]\!](f_1, \epsilon_1, \sigma_1)$ in

$\quad$ if $has\_field(a_1, \_\_add\_\_, \sigma_2)$ then

$\quad\quad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_1.\_\_add\_\_(a_2)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then

$\quad\quad\quad$ if $has\_field(a_2, \_\_radd\_\_, \sigma_3) \wedge typeof(a_1) \neq typeof(a_2)$ then

$\quad\quad\quad\quad$ letif $(f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_3, \epsilon_3, \sigma_3)$ in

$\quad\quad\quad\quad$ if $\sigma_4(a_4) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_4, \epsilon_4, \sigma_4)$

$\quad\quad\quad\quad$ else $(f_4, \epsilon_4, \sigma_4, a_4)$

$\quad\quad\quad$ else $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $f_3, \epsilon_3, \sigma_3, a_3$

$\quad$ else if $has\_field(a_2, \_\_radd\_\_, \sigma_2) \wedge typeof\, a_1 \neq typeof\, a_2$ then

$\quad\quad$ letif $(f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![a_2.\_\_radd\_\_(a_1)]\!](f_2, \epsilon_2, \sigma_2)$ in

$\quad\quad$ if $\sigma_3(a_3) = (\_, \texttt{NotImpl})$ then $\texttt{TypeError}(f_3, \epsilon_3, \sigma_3)$

$\quad\quad$ else $(f_3, \epsilon_3, \sigma_3, a_3)$

$\quad$ else $\texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

Need to know:

- attributes of e_1, e_2
- type of e_1, e_2

Typing +

$$
\begin{array}{l}
\mathbb{E}[\![\, e_1 + e_2 \,]\!]\,(f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\text{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![\, e_1 \,]\!]\,(f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![\, e_2 \,]\!]\,(f_1, \epsilon_1, \sigma_1) \text{ in} \\
\quad \text{if } \; has\_field(a_1, \_\_add\_\_, \sigma_2) \; \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, \bullet a_1.\_\_add\_\_(a_2) \bullet \,]\!]\,(f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
\qquad\quad \text{if } \; has\_field(a_2, \_\_radd\_\_, \sigma_3) \; \wedge \; typeof(a_1) \neq typeof(a_2) \; \text{ then} \\
\qquad\qquad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![\, \bullet a_2.\_\_radd\_\_(a_1) \bullet \,]\!]\,(f_3, \epsilon_3, \sigma_3) \text{ in} \\
\qquad\qquad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\qquad\qquad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\qquad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
\quad \text{else if } \; has\_field(a_2, \_\_radd\_\_, \sigma_2) \; \wedge \; typeof\, a_1 \neq typeof\, a_2 \; \text{ then} \\
\qquad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\, \bullet a_2.\_\_radd\_\_(a_1) \bullet \,]\!]\,(f_2, \epsilon_2, \sigma_2) \text{ in} \\
\qquad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\qquad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

Need to know:

- attributes of e_1, e_2
- type of e_1, e_2
- return value for called function

Typing +

$$
\begin{array}{l}
\mathbb{E}[\![\, e_1 + e_2 \,]\!]\,(f, \epsilon, \sigma) \stackrel{\text{def}}{=} \\
\quad \text{if } f \neq cur \text{ then } (f, \epsilon, \sigma, addr_{\texttt{None}}) \text{ else} \\
\quad \text{letif } (f_1, \epsilon_1, \sigma_1, a_1) = \mathbb{E}[\![\, e_1 \,]\!]\,(f, \epsilon, \sigma) \text{ in} \\
\quad \text{letif } (f_2, \epsilon_2, \sigma_2, a_2) = \mathbb{E}[\![\, e_2 \,]\!]\,(f_1, \epsilon_1, \sigma_1) \text{ in} \\
\quad \text{if } \; has\_field(a_1, \_\_add\_\_, \sigma_2) \; \text{ then} \\
\quad\quad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\; a_1.\_\_add\_\_(a_2) \;]\!]\,(f_2, \epsilon_2, \sigma_2) \text{ in} \\
\quad\quad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then} \\
\quad\quad\quad \text{if } \; has\_field(a_2, \_\_radd\_\_, \sigma_3) \; \wedge \; typeof(a_1) \neq typeof(a_2) \; \text{ then} \\
\quad\quad\quad\quad \text{letif } (f_4, \epsilon_4, \sigma_4, a_4) = \mathbb{E}[\![\; a_2.\_\_radd\_\_(a_1) \;]\!]\,(f_3, \epsilon_3, \sigma_3) \text{ in} \\
\quad\quad\quad\quad \text{if } \sigma_4(a_4) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_4, \epsilon_4, \sigma_4) \\
\quad\quad\quad\quad \text{else } (f_4, \epsilon_4, \sigma_4, a_4) \\
\quad\quad\quad \text{else } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\quad\quad \text{else } f_3, \epsilon_3, \sigma_3, a_3 \\
\quad \text{else if } \; has\_field(a_2, \_\_radd\_\_, \sigma_2) \; \wedge \; typeof\, a_1 \neq typeof\, a_2 \; \text{ then} \\
\quad\quad \text{letif } (f_3, \epsilon_3, \sigma_3, a_3) = \mathbb{E}[\![\; a_2.\_\_radd\_\_(a_1) \;]\!]\,(f_2, \epsilon_2, \sigma_2) \text{ in} \\
\quad\quad \text{if } \sigma_3(a_3) = (\_, \texttt{NotImpl}) \text{ then } \texttt{TypeError}(f_3, \epsilon_3, \sigma_3) \\
\quad\quad \text{else } (f_3, \epsilon_3, \sigma_3, a_3) \\
\quad \text{else } \texttt{TypeError}(f_2, \epsilon_2, \sigma_2)
\end{array}
$$

Need to know:

- attributes of e_1, e_2
- type of e_1, e_2
- return value for called function

$\Longrightarrow$ difficult to type + modularly

Type-based analysis of Python

Current state of the type analysis

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

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

```
r1 = fspath('a')
r2 = fspath(b'path')
r3 = fspath(FSPath())
```

- `r1:` `str`
- `r2:` `bytes`
- r3: TypeError raised

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly
- Bottom-up analysis (as in OCaml)

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly
- Bottom-up analysis (as in OCaml): best performance, extremely difficult here (typing of +)

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly
- Bottom-up analysis (as in OCaml): best performance, extremely difficult here (typing of +)
- Summary-based, top-down analysis

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly
- Bottom-up analysis (as in OCaml): best performance, extremely difficult here (typing of +)
- **Summary-based, top-down analysis**

In the works: a partially modular interprocedural analysis

What kind of interprocedural analysis?

- Inlining: most precise, but costly
- Bottom-up analysis (as in OCaml): best performance, extremely difficult here (typing of $+$)
- **Summary-based, top-down analysis**

Summaries keep the results of the previous analyses (a bit similar to memoization).

Digression: implementation into MOPSA

Modular **O**pen **P**latform for **S**tatic **A**nalysis.

- Modular abstract domains for: abstract values, control-flow, ...
- Statements flow through abstract domains until one answers
- User selects the combination of abstract domains
- Currently, subsets of C and Python are supported

Future work

Future work

- Other modular interprocedural analyses

Future work

- Other modular interprocedural analyses
- Library analysis

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation
  - Python has a massive standard library

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation
  - Python has a massive standard library
  - Many different libraries exist for Python

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation
  - Python has a massive standard library
  - Many different libraries exist for Python
- Multilingual analysis (Python and C)

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation
  - Python has a massive standard library
  - Many different libraries exist for Python
- Multilingual analysis (Python and C)
  - Analyze both Python code and CPython calls

Future work

- Other modular interprocedural analyses
- Library analysis
  - Infer possible orders of function calls
    (example: first open, then read and finally close a file)
  - Notion of soundness?
- Automatic stub generation
  - Python has a massive standard library
  - Many different libraries exist for Python
- Multilingual analysis (Python and C)
  - Analyze both Python code and CPython calls
  - Similarly, some Python libraries use a lot of C code

Appendix

Other Type Analyses

Program	Fritz and Hage	Pytype	Typpete	MOPSA - types
Analysis method	Dataflow analysis	Unclear	SMT-solver	AI
class_attr_ok	✓	✗	✳	✓
class_pre_store	✓	✓	✓	✓
default_args_class	✓	✓	✓	✓
except_clause	✗	✳	✓	✓
fspath	✗	✗	✳	✓
magic	✓	✓	✓	✳
polyfib	✳	✗	✳	✳
poly_lists	✳	✓	✳	✓
vehicle	✓	✓	✓	✳
widening	✓	✗	✳	✓

✓ sound and precise

✳ sound but false alarm

✗ unsound

Other Type Analyses

- Typpete [2]: uses an SMT solver to perform type inference

[2]Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).

[3]Fritz and Hage. "Cost versus precision for approximate typing for Python". Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2017, Paris, France, January 18-20, 2017.

Other Type Analyses

- Typpete 🐇[2]: uses an SMT solver to perform type inference
- Work of Fritz and Hage[3]: analysis written in terms of data-flow equations

[2] Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).

[3] Fritz and Hage. "Cost versus precision for approximate typing for Python". Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2017, Paris, France, January 18-20, 2017.

Other Type Analyses

- Typpete [2]: uses an SMT solver to perform type inference
- Work of Fritz and Hage[3]: analysis written in terms of data-flow equations
- Pytype: from engineers at Google, no technical reference

[2] Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).

[3] Fritz and Hage. "Cost versus precision for approximate typing for Python". Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2017, Paris, France, January 18-20, 2017.

Value Analysis of Python

Value Analysis[4]:

- incomparable with our analysis (due to the relationality)

[4] Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings.

Value Analysis of Python

Value Analysis[4]:

- incomparable with our analysis (due to the relationality)
- generally more precise information, more costly

[4] Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings.

Value Analysis of Python

Value Analysis[4]:

- incomparable with our analysis (due to the relationality)
- generally more precise information, more costly
- support for large Python library more difficult to implement

[4] Fromherz, Ouadjaout, and Miné. "Static Value Analysis of Python Programs by Abstract Interpretation". NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings.