Static Analysis by Abstract Interpretation Collecting Types of Python Programs

Raphaël Monat

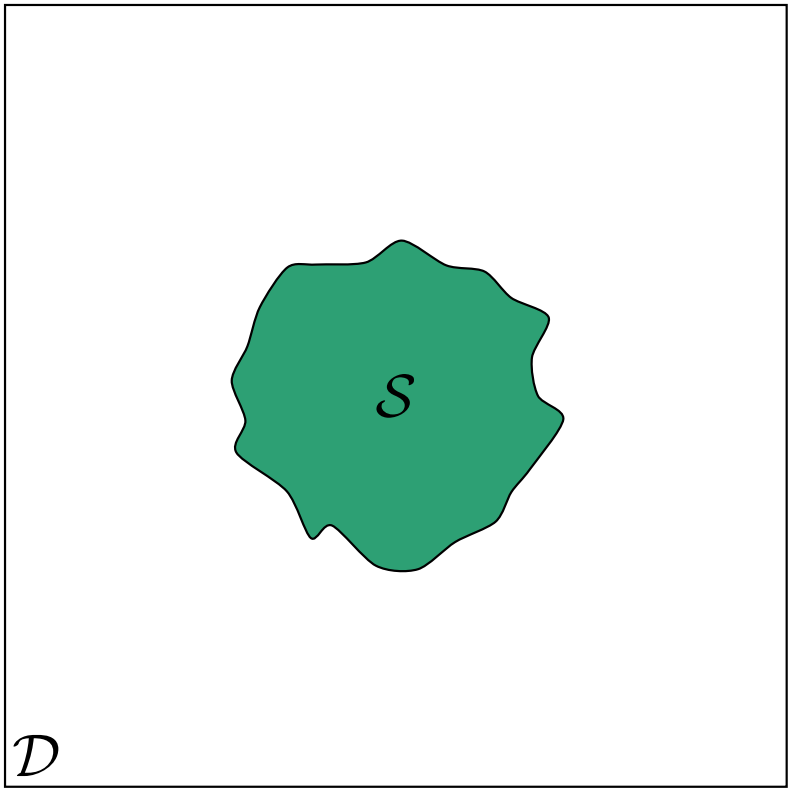
Intern in APR team, LIP6, Sorbonne Université
Under the supervision of Antoine Miné

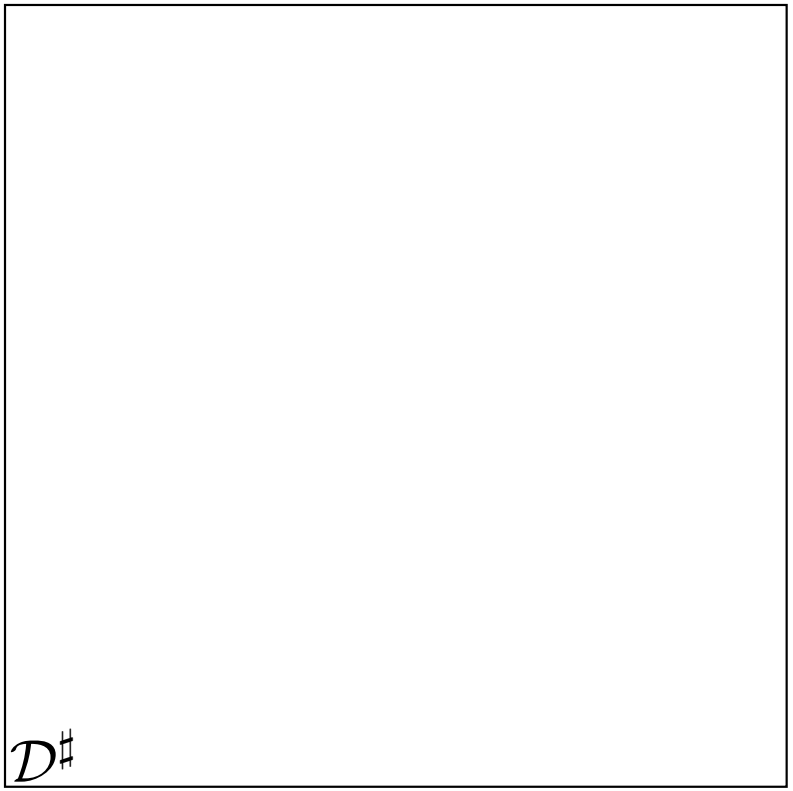
Introduction

Software Bugs

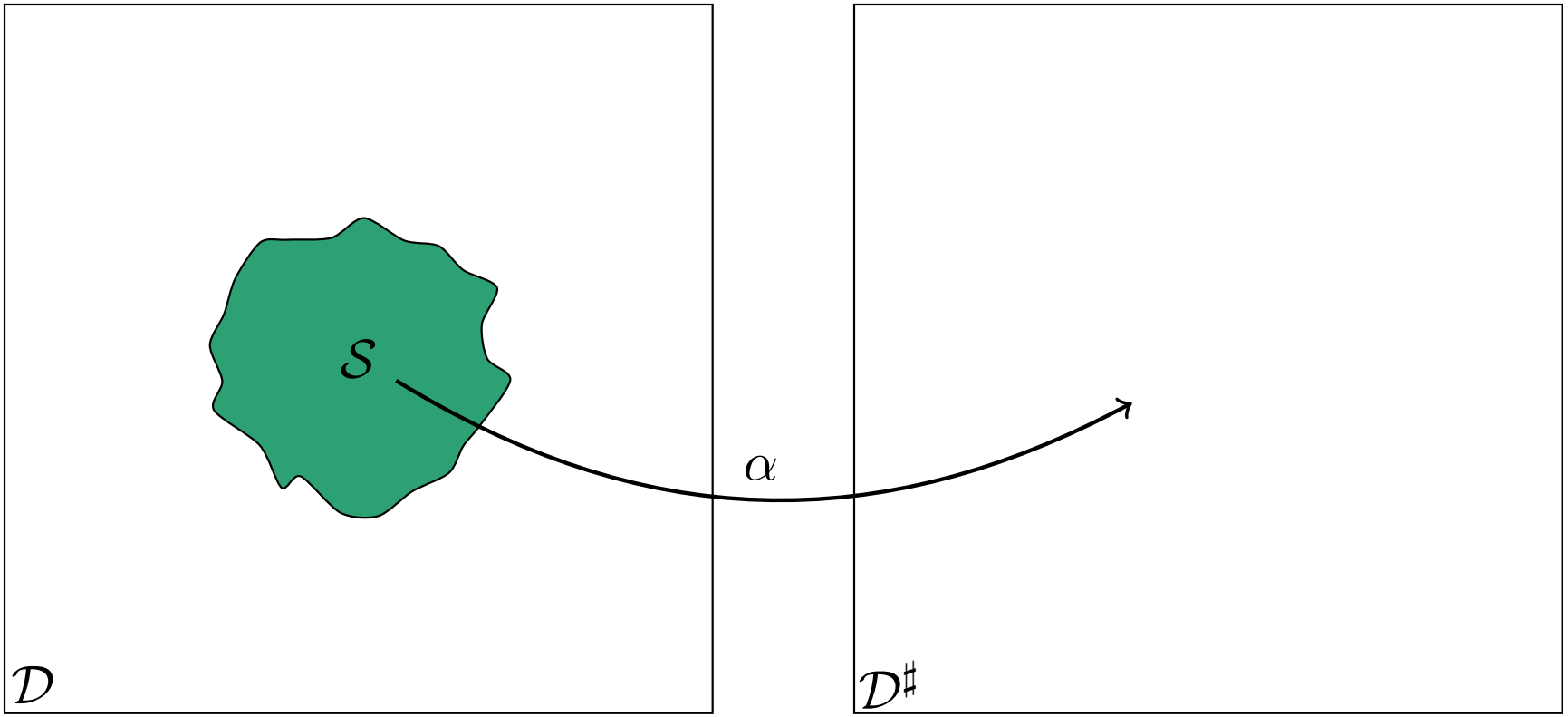
Year 2000
Ariane 5
software bugs
Patriot
HeartBleed
Boeing 787
OpenSSL
Nasa Mariner 1
Mars Global Surveyor
Amazon
Bank of Queensland
Soviet Gas
Deutsche Telekom
Kerberos
PayPal
Pentium
Zenit 3SL
Delta-3
AT&T
A400M
Chinook
Hamburg trains
Nasa Spirit
Mars Climate Orbiter
Internet Explorer
Northeast Blackout
Hartford Coliseum
Skyshield 35
Titan IV
Therac-25
F-16
F-22
Osprey
Medtronic
Toyota
Hanover trains
Mars Pathfinder
Mars Polar Lander
USS Yorktown
Phobos 1
Year 2000
Hitomi
CryoSat-1
Nasa Mariner 1
Ariane 5
Boeing 787
Soviet Gas
Chinook
Northeast Blackout
software bugs
PayPal
Amazon
Bank of Queensland
Internet Explorer
Mars Global Surveyor
Nasa Mariner 1
OpenSSL
Kerberos
Deutsche Telekom
Hamburg trains
Zenit 3SL
Delta-3
Pentium
Patriot
HeartBleed
Boeing 787

Abstract Interpretation

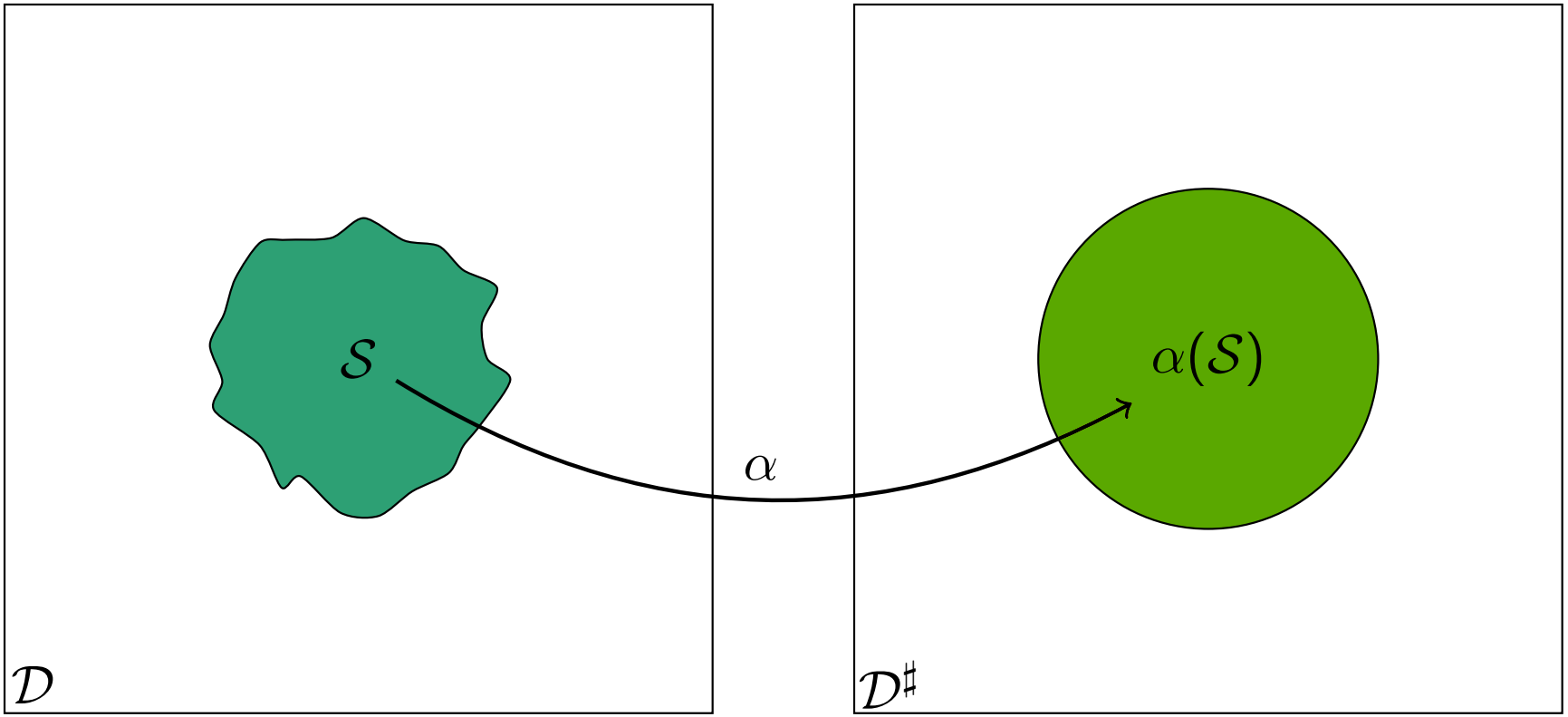




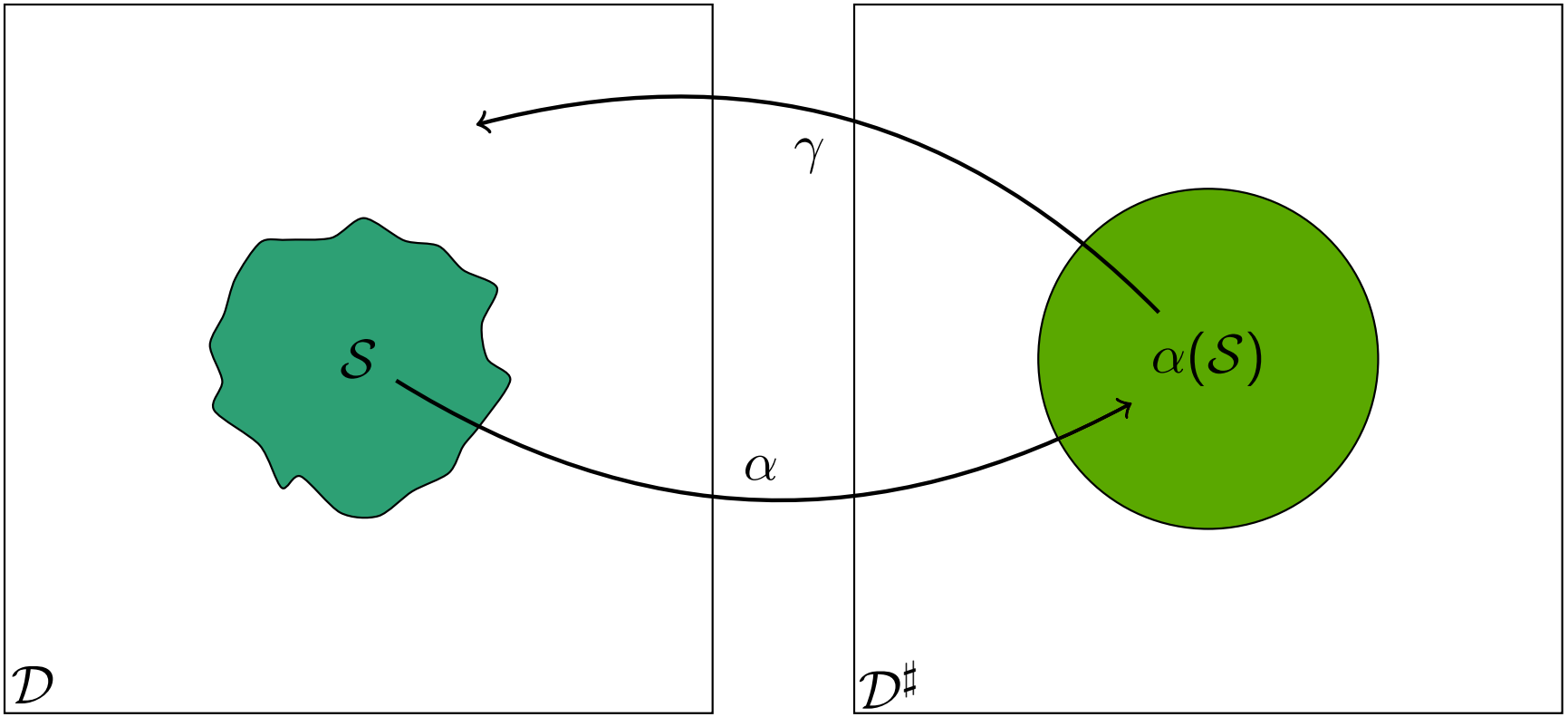
Abstract Interpretation



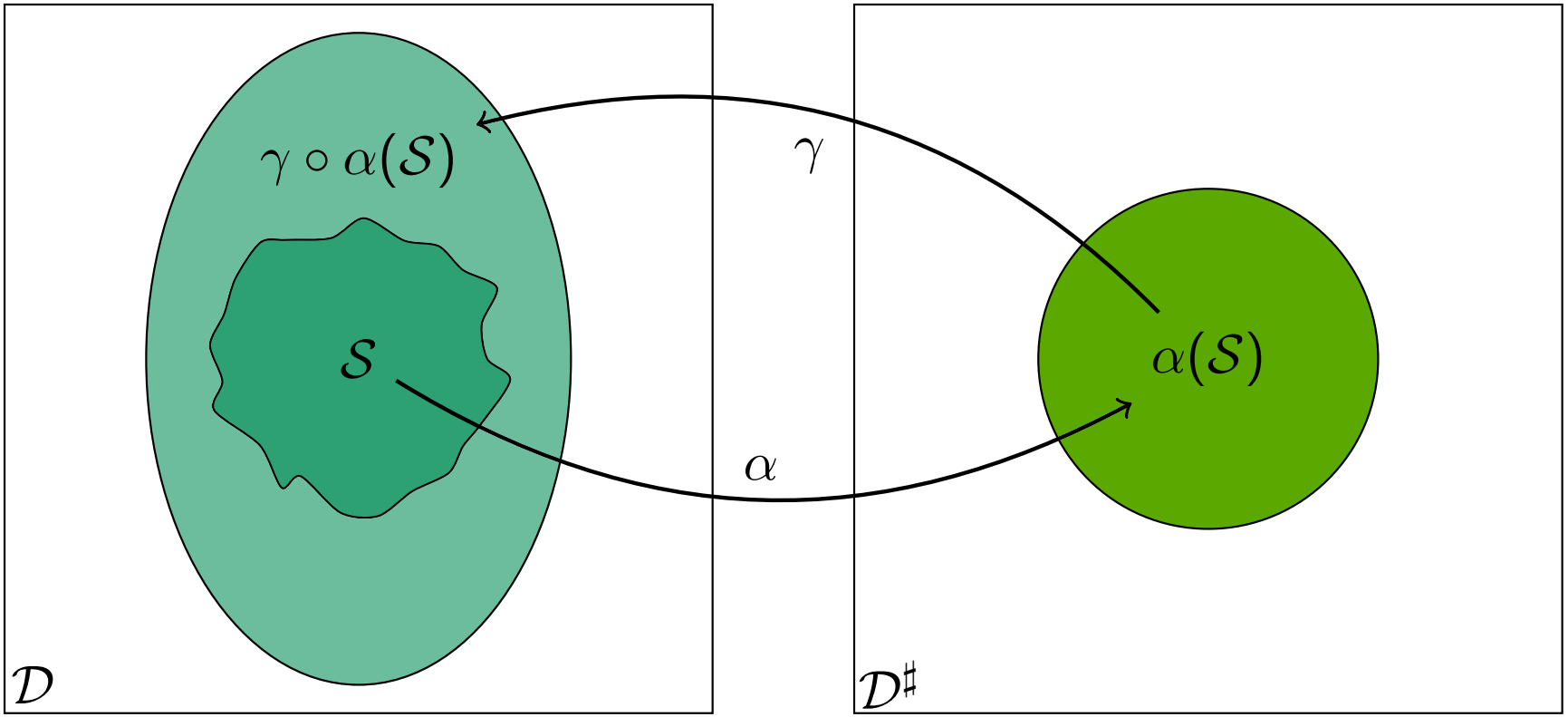
Abstract Interpretation



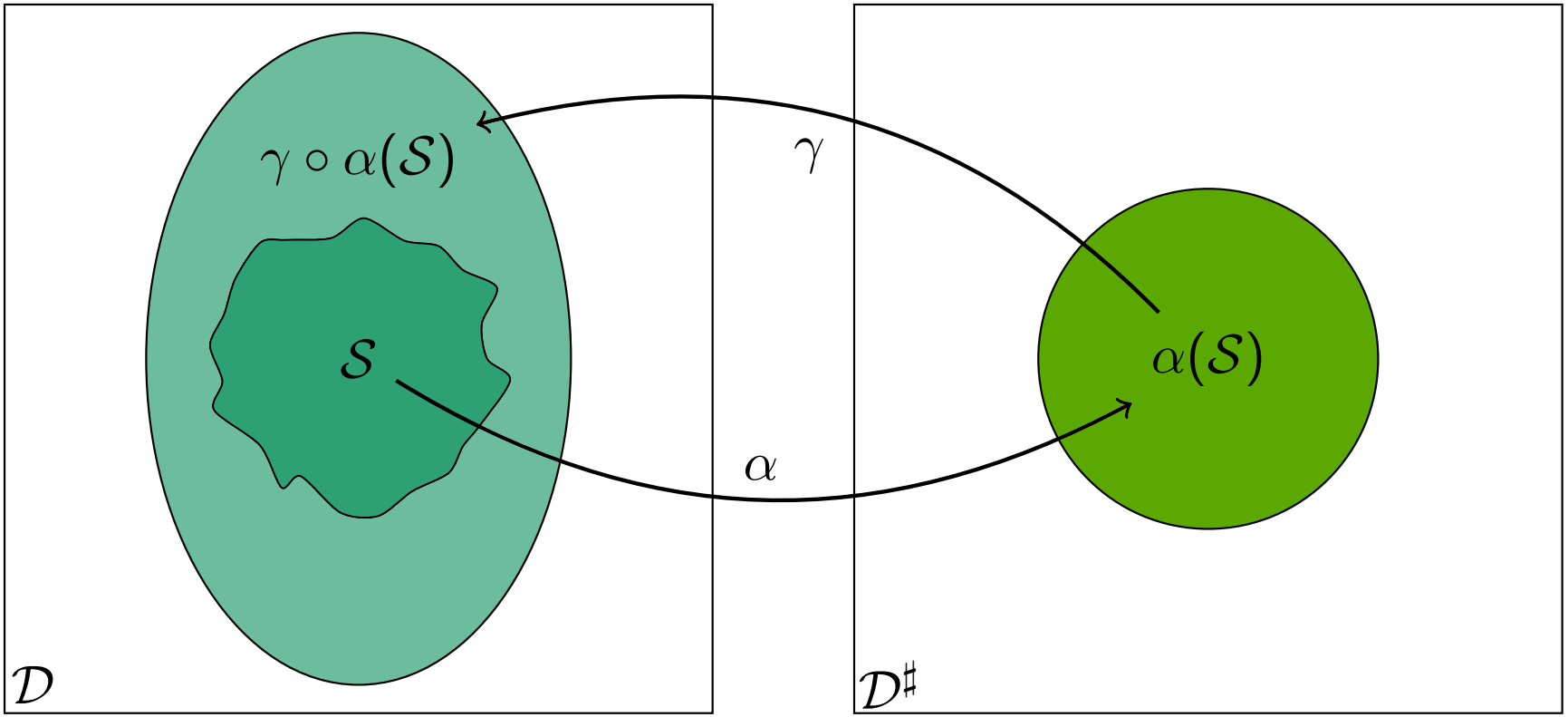
Abstract Interpretation



Abstract Interpretation

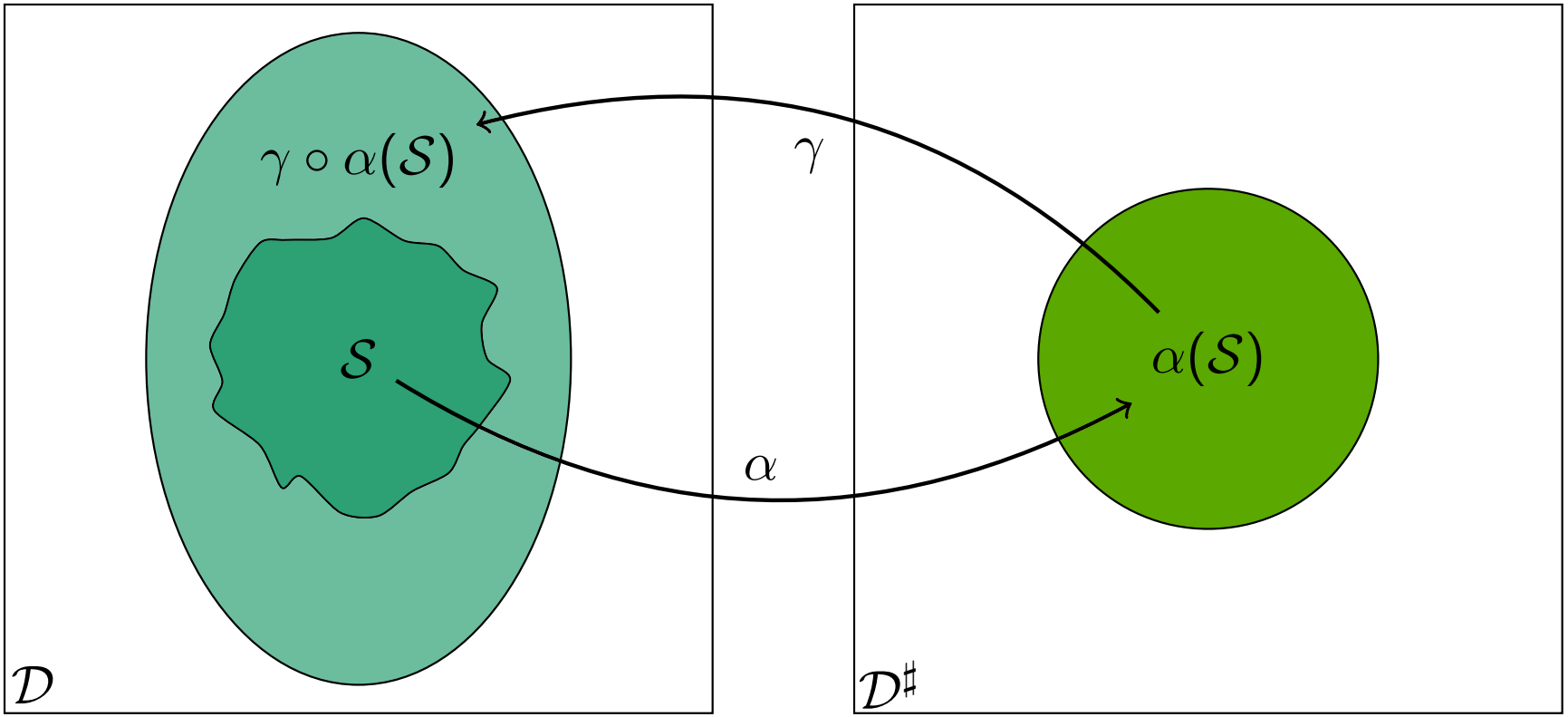


Abstract Interpretation – Numerical Invariants



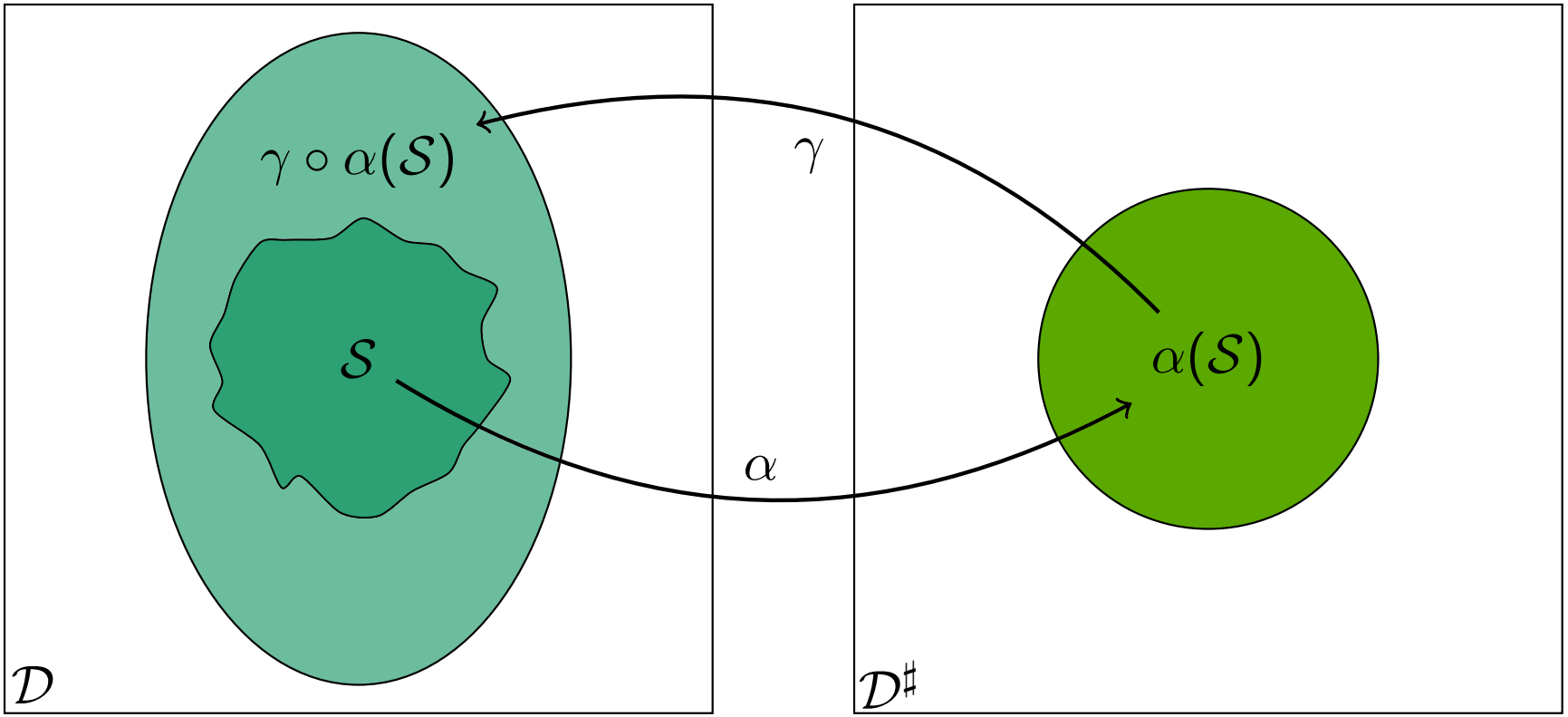
- $\mathcal{D} = \mathcal{P}(\mathbb{Z})$

Abstract Interpretation – Numerical Invariants



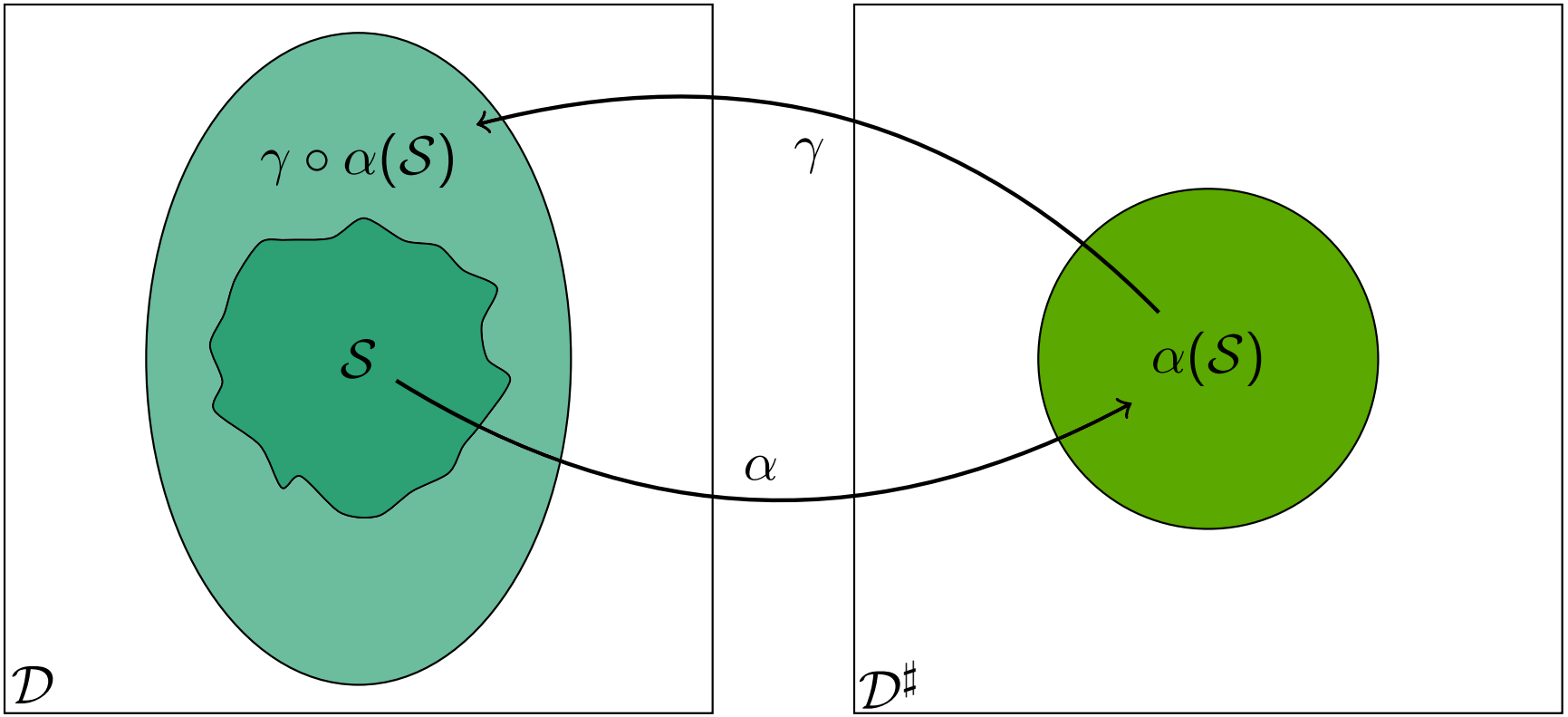
- $\mathcal{D} = \mathcal{P}(\mathbb{Z})$
- $\mathcal{D}^\sharp = \{\, [a, b] \mid a \in \mathbb{Z} \cup \{\, -\infty \,\}, b \in \mathbb{Z} \cup \{\, +\infty \,\}, a \leq b \,\} \cup \{\, \bot \,\}$

Abstract Interpretation – Numerical Invariants



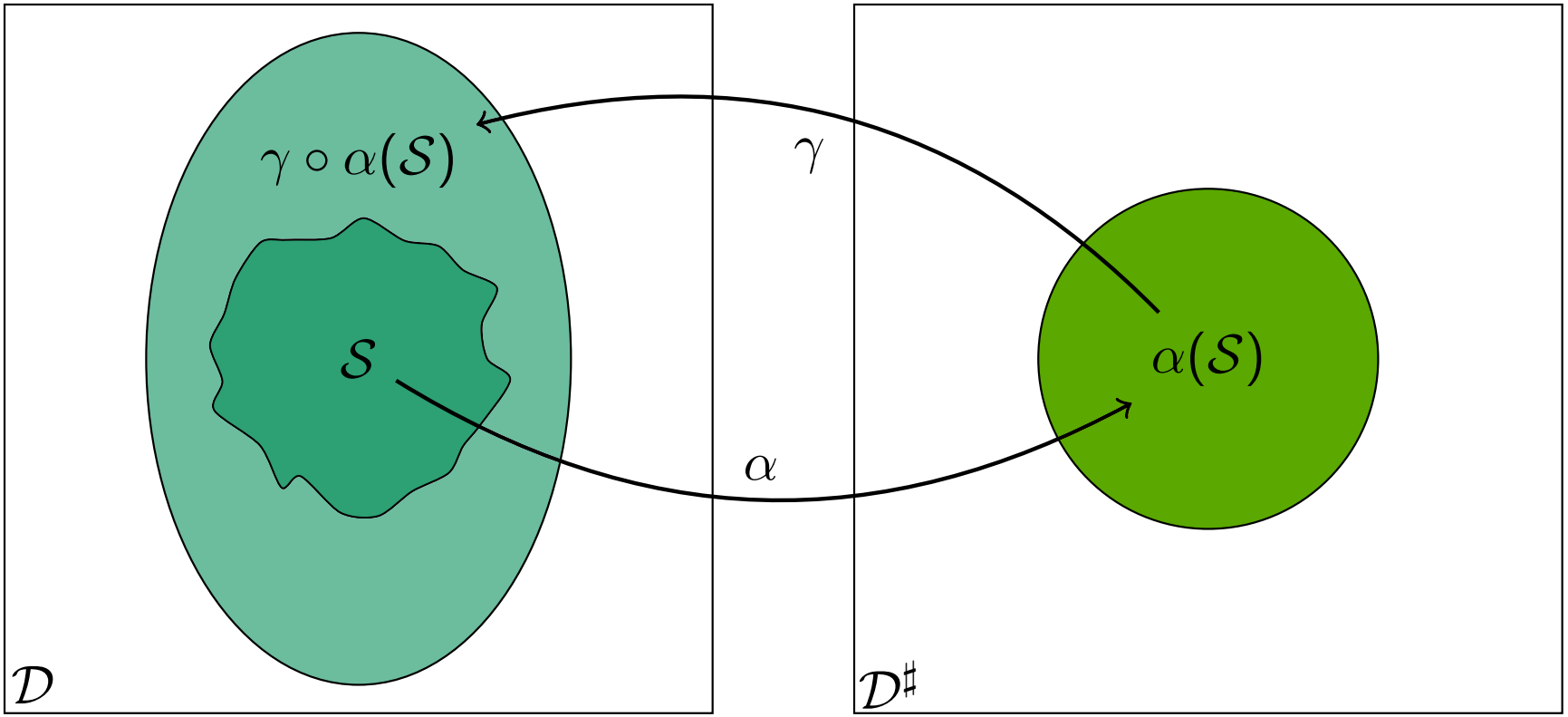
- $\mathcal{D} = \mathcal{P}(\mathbb{Z})$
- $\mathcal{D}^\sharp = \{\, [a, b] \mid a \in \mathbb{Z} \cup \{\, -\infty \,\}, b \in \mathbb{Z} \cup \{\, +\infty \,\}, a \leq b \,\} \cup \{\, \bot \,\}$
- $\mathcal{S} = \{\, 1, 3, 5 \,\}$

Abstract Interpretation – Numerical Invariants



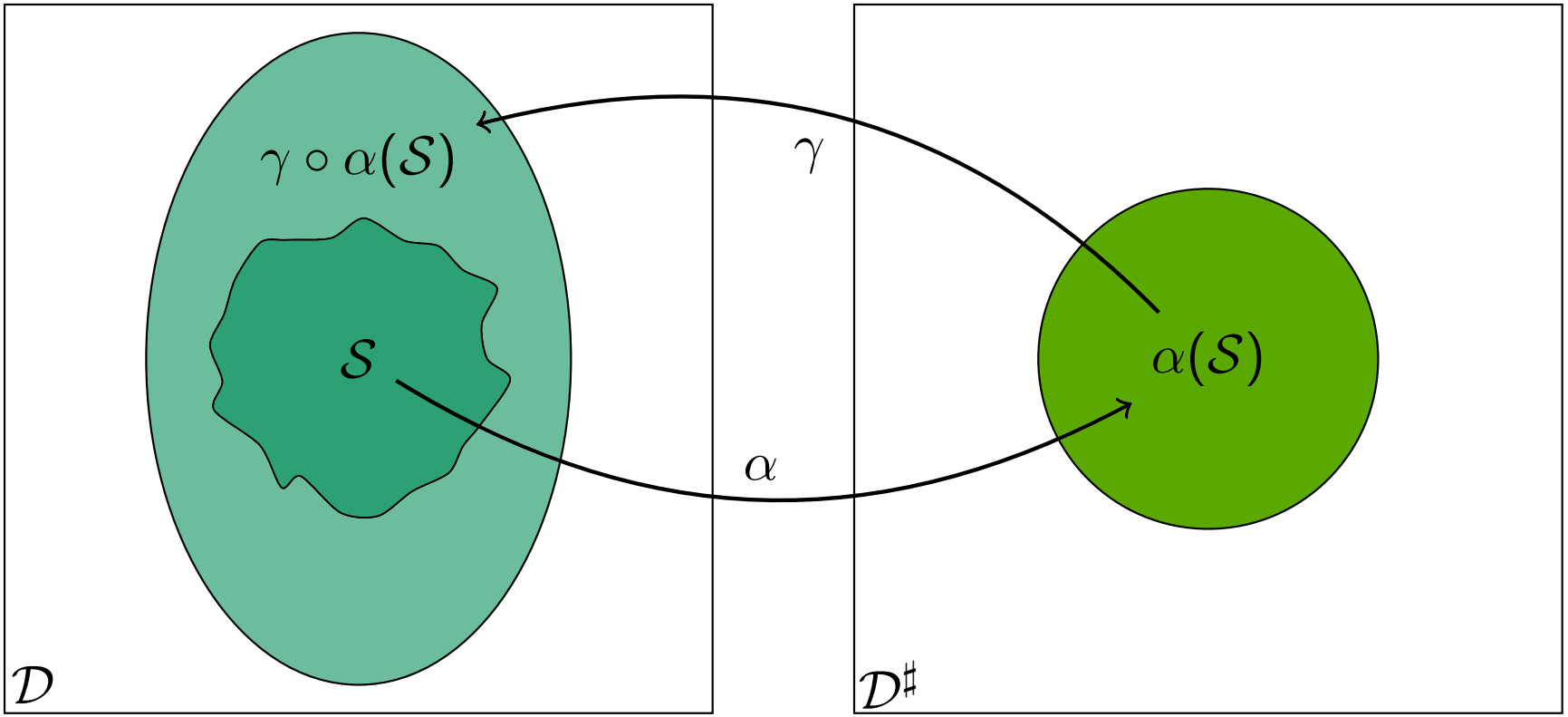
- $\mathcal{D} = \mathcal{P}(\mathbb{Z})$
- $\mathcal{D}^\sharp = \{\, [a, b] \mid a \in \mathbb{Z} \cup \{\, -\infty \,\}, b \in \mathbb{Z} \cup \{\, +\infty \,\}, a \leq b \,\} \cup \{\, \bot \,\}$
- $\mathcal{S} = \{\, 1, 3, 5 \,\}$
- $\alpha(\mathcal{S}) = [1, 5]$

Abstract Interpretation – Numerical Invariants



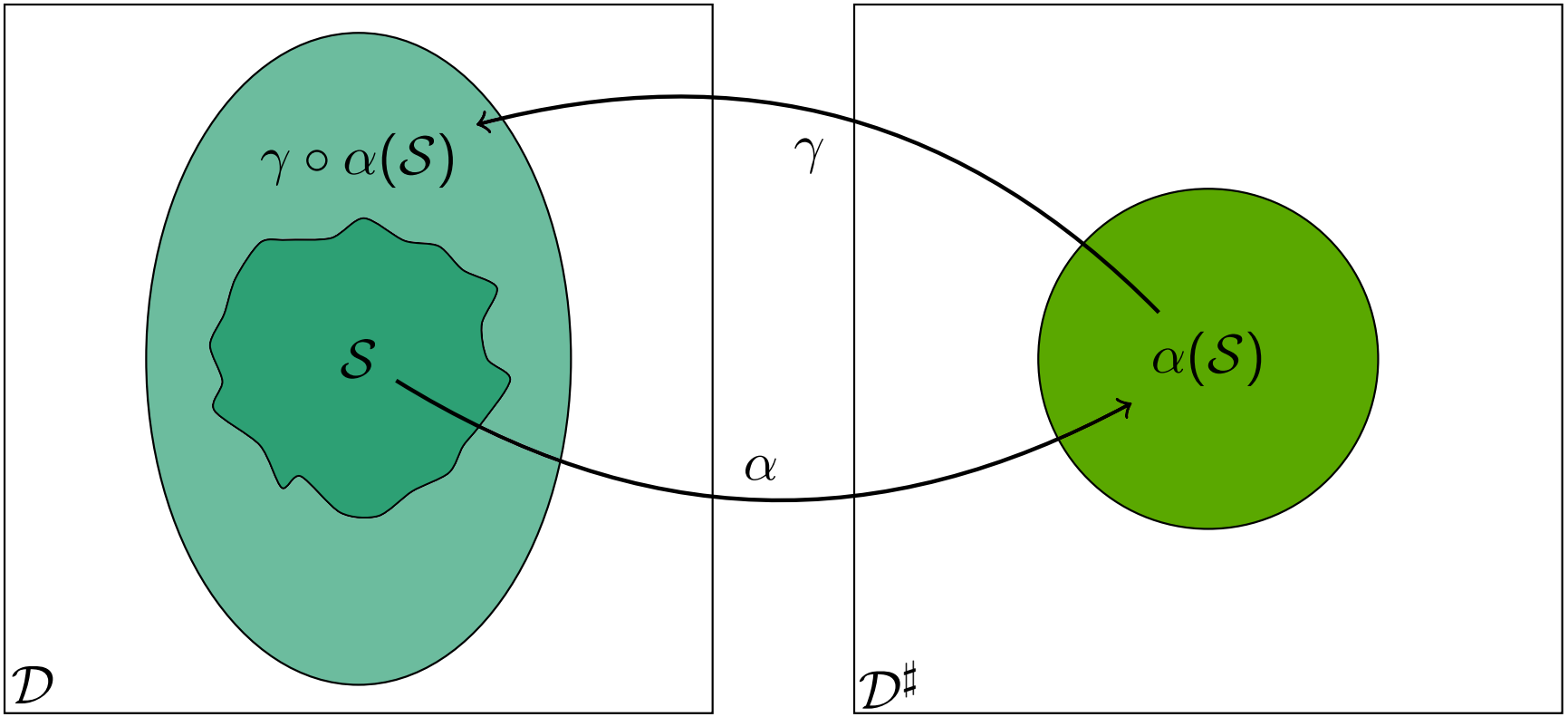
- $\mathcal{D} = \mathcal{P}(\mathbb{Z})$
- $\mathcal{D}^\sharp = \{\, [a, b] \mid a \in \mathbb{Z} \cup \{\, -\infty \,\}, b \in \mathbb{Z} \cup \{\, +\infty \,\}, a \leq b \,\} \cup \{\, \bot \,\}$
- $\mathcal{S} = \{\, 1, 3, 5 \,\}$
- $\alpha(\mathcal{S}) = [1, 5]$
- $\gamma \circ \alpha(\mathcal{S}) = \{\, 1, 2, 3, 4, 5 \,\}$

Abstract Interpretation – Typing of the λ-calculus



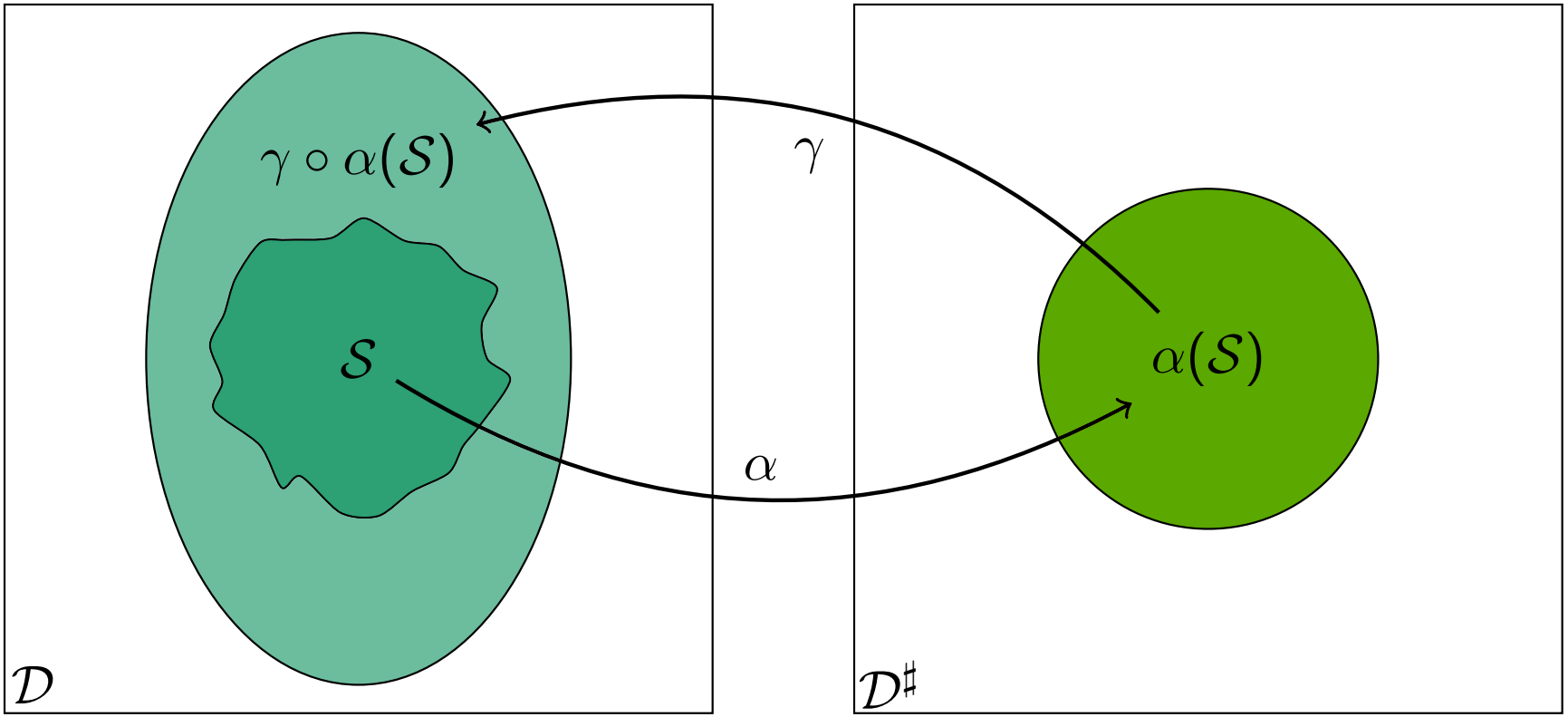
- $\mathcal{D}$ = semantics of the lambda-calculus

Abstract Interpretation – Typing of the λ-calculus



- $\mathcal{D}$ = semantics of the lambda-calculus
- $\mathcal{D}^\sharp$ = types of the lambda-calculus

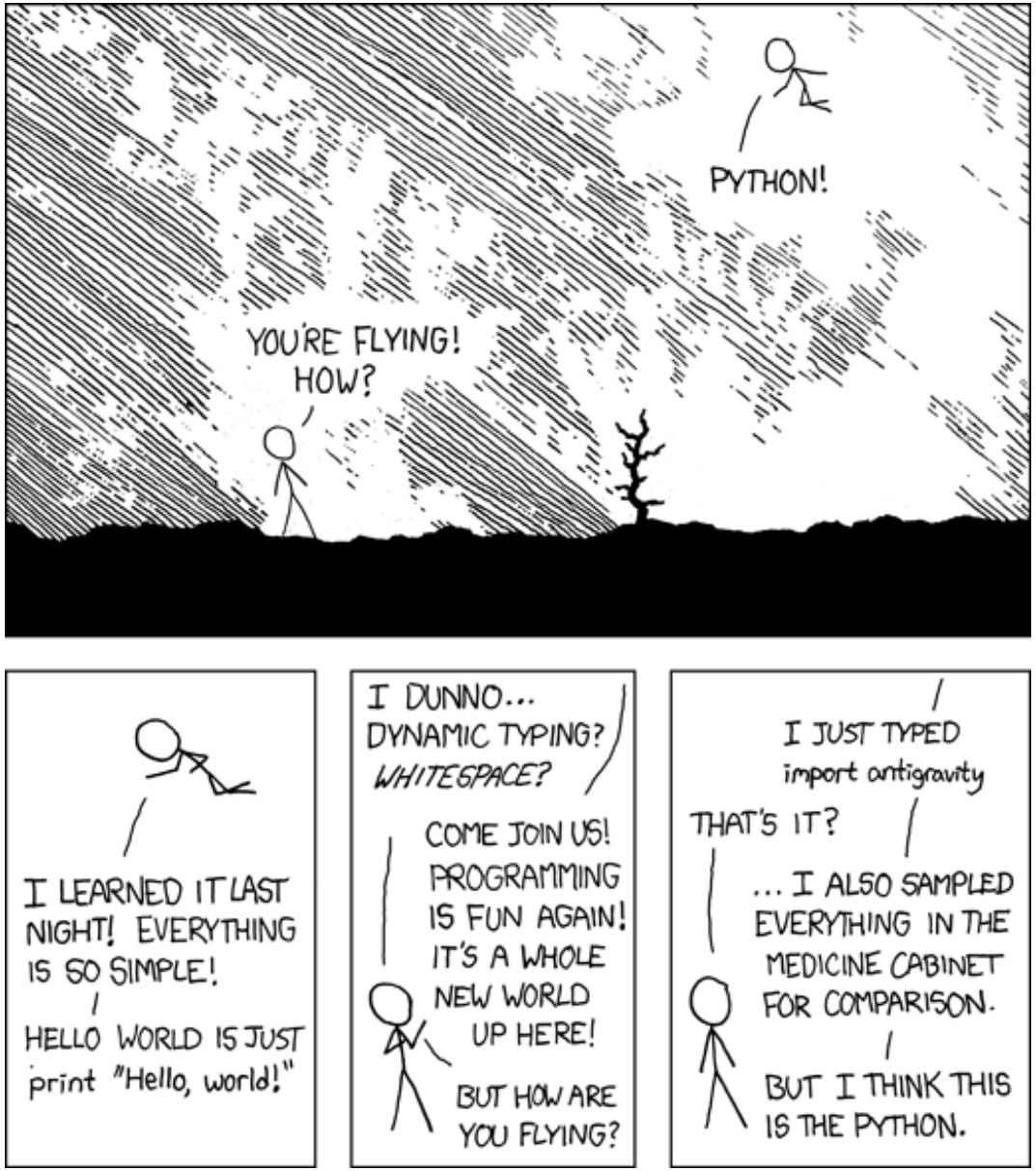
Abstract Interpretation – Typing of the λ-calculus



- $\mathcal{D}$ = semantics of the lambda-calculus
- $\mathcal{D}^\sharp$ = types of the lambda-calculus

$\Longrightarrow$ more in the report

Python



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

Python is Popular

- used in the teaching and scientific communities
- used for popular projects: Django, SageMath

Python

Python is Popular

- used in the teaching and scientific communities
- used for popular projects: Django, SageMath

Python is a Dynamic Programming Language

- interpreted language
- everything is object
- introspection
- eval

Python

Python is Popular

- used in the teaching and scientific communities
- used for popular projects: Django, SageMath

Python is a Dynamic Programming Language

- interpreted language
- everything is object
- introspection
- eval

Two Notions of Typing

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

Python

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("...")
```

Python

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("...")
```

Signature?

Python

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("...")
```

Signature?

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

Python

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("...")
```

Signature?

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

Objects having method $__\mathit{fspath}__$, itself returning type α

Ideas and Inspirations

Static analyses are especially helpful on dynamic languages.

Change Python's value analysis[1] into:

- A more abstract analysis (types as abstraction of values)
- A more precise analysis (relationality)

[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.

Goals

- Collecting types and type errors for Python programs. Contrary to a type system:
  - Naturally flow-sensitive analysis.
  - No rejection of untypable programs.
- Performing a sound analysis: no alarm $\implies$ no bug.
- Performing a relational analysis, supporting bounded parametric polymorphism.
- Accepting all Python semantics (no restriction such as "variables have a fixed type").

Concrete Semantics of Python

Concrete Semantics of Python

- ✗ No formal semantics (and no standard) defined for Python.
- CPython's implementation defines the semantics.
- We use a slight modification of the semantics of Fromherz, Ouadjaout, and Miné[2].

[2] 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 +

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

Evaluate e_1

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

Semantics of +

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

Evaluate e_2

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

Semantics of +

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

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

If e_1 has method $__add__$

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

Semantics of +

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

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

and if $__add__$ does not return `NotImpl`

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

Semantics of +

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

return the result of the $__add__$ call

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 $(f, \epsilon, \sigma, addr_{\texttt{None}})$ 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 $f_3, \epsilon_3, \sigma_3, a_3$

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

otherwise, if e_1 and e_2 have different types

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

Semantics of +

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

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 $(f, \epsilon, \sigma, addr_{\texttt{None}})$ 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 $\texttt{TypeError}(f_2, \epsilon_2, \sigma_2)$

raise a type error if nothing works

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

Type Abstract Domain

Types: what do we need?

Types: what do we need?

Top and bottom types

Types: what do we need?

Top and bottom types

Class declarations

Types: what do we need?

Top and bottom types

Class declarations

Instances

```
class A: pass
x = A()

if *:
  x.o = 3
  x.u = 4
else:
  x.u = 4
ze = x.o
z = x.u
```

Types: what do we need?

Top and bottom types

Class declarations

Instances

```
class A: pass
x = A()

if *:
  x.o = 3
  x.u = 4
else:
  x.u = 4
ze = x.o
z = x.u
```

Types: what do we need?

Top and bottom types

Class declarations

Instances

```
class A: pass
x = A()

if *:
  x.o = 3
  x.u = 4
else:
  x.u = 4
ze = x.o
z = x.u
```

Types: what do we need?

Top and bottom types

Class declarations

Instances

```
class A: pass
x = A()

if *:
  x.o = 3
  x.u = 4
else:
  x.u = 4
ze = x.o
z = x.u
```

Types: what do we need?

Top and bottom types

Class declarations

Instances

```
class A: pass
x = A()

if *:
  x.o = 3
  x.u = 4
else:
  x.u = 4
ze = x.o
z = x.u
```

$$\Longrightarrow \mathit{Instance}[v, \mathit{mustAttributes}, \mathit{mayAttributes}]$$

Types: what do we need?

Type variables

- `l = True if * else 42`

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x`

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \rightarrow \alpha$", $\alpha \in \{ \top \}$

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \to \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \rightarrow \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \rightarrow \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

- `l = [1, 2]`

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \rightarrow \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

- `l = [1, 2]` $\rightsquigarrow \mathit{List}[\mathit{int}]$

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \to \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

- `l = [1, 2]` $\rightsquigarrow \mathit{List}[\mathit{int}]$
- `h = ['a', 4, [3.14]]`

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \rightarrow \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

- `l = [1, 2]` $\rightsquigarrow \mathit{List}[\mathit{int}]$
- `h = ['a', 4, [3.14]]` $\rightsquigarrow \mathit{List}[\alpha], \alpha \in \{ \mathit{str}, \mathit{int}, \mathit{List}[\mathit{float}] \}$

Types: what do we need?

Type variables

- `l = True if * else 42` $\rightsquigarrow \alpha, \alpha \in \{ \mathit{bool}, \mathit{int} \}$
- `def f(x): return x` $\rightsquigarrow$ "$\alpha \to \alpha$", $\alpha \in \{ \top \}$

$$\implies \alpha \in V_\Sigma$$

Standard containers

- `l = [1, 2]` $\rightsquigarrow \mathit{List}[\mathit{int}]$
- `h = ['a', 4, [3.14]]` $\rightsquigarrow \mathit{List}[\alpha], \alpha \in \{ \mathit{str}, \mathit{int}, \mathit{List}[\mathit{float}] \}$

$$\implies \mathit{List}[v], v \in V^\sharp$$

Types

Polymorphic types

$$
\begin{aligned}
V^\sharp = \bot \mid \top \\
&\mid c, c \in \mathit{Classes} \\
&\mid \mathit{List}[v] \mid \mathit{Tuple}[v] \mid \ \ldots, v \in V^\sharp \\
&\mid \mathit{Instance}[v, \mathit{mustAttributes}, \mathit{mayAttributes}], v \in V^\sharp \\
&\mid V_\Sigma \in \{\alpha, \beta, \ldots\}
\end{aligned}
$$

Monormophic types

$$V_m^\sharp = V^\sharp \text{ without type variables}$$

Structure of the Abstract Domain

$\mathcal{D}_1^\sharp$ partitions the environments in packs of variables having the same type. Partitions are refined to take aliasing into account.

$$\mathcal{D}^\sharp = \mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp \times \mathcal{D}_3^\sharp$$

$$\mathcal{D}_1^\sharp = \mathrm{Id} \rightharpoonup \mathit{TypeId} \times \mathit{Aliasing}$$

$$\mathcal{D}_2^\sharp = \mathit{TypeId} \rightharpoonup V^\sharp$$

$$\mathcal{D}_3^\sharp = V_\Sigma \rightharpoonup \mathcal{P}(V_m^\sharp)$$

Example

$$d_1 = x \mapsto (0, \bot), y \mapsto (0, \bot), z \mapsto (1, \bot)$$

Structure of the Abstract Domain

$\mathcal{D}_2^\sharp$ associates to each partition number a polymorphic type.

$$\mathcal{D}^\sharp = \mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp \times \mathcal{D}_3^\sharp$$
$$\mathcal{D}_1^\sharp = \mathbf{Id} \rightharpoonup \mathit{Typeld} \times \mathit{Aliasing}$$
$$\boldsymbol{\mathcal{D}_2^\sharp = \mathit{Typeld} \rightharpoonup V^\sharp}$$
$$\mathcal{D}_3^\sharp = V_\Sigma \rightharpoonup \mathcal{P}(V_m^\sharp)$$

Example

$$\begin{cases} d_1 & = x \mapsto (0, \bot), y \mapsto (0, \bot), z \mapsto (1, \bot) \\ d_2 & = 0 \mapsto \alpha, 1 \mapsto \mathit{List}[\alpha] \end{cases}$$

Structure of the Abstract Domain

$\mathcal{D}_3^\sharp$ is a type variable environment.

$$\mathcal{D}^\sharp = \mathcal{D}_1^\sharp \times \mathcal{D}_2^\sharp \times \mathcal{D}_3^\sharp$$
$$\mathcal{D}_1^\sharp = \mathbf{Id} \rightharpoonup \mathit{Typeld} \times \mathit{Aliasing}$$
$$\mathcal{D}_2^\sharp = \mathit{Typeld} \rightharpoonup V^\sharp$$
$$\mathcal{D}_3^\sharp = V_\Sigma \rightharpoonup \mathcal{P}(V_m^\sharp)$$

Example

$$\begin{cases} d_1 & = x \mapsto (0, \bot), y \mapsto (0, \bot), z \mapsto (1, \bot) \\ d_2 & = 0 \mapsto \alpha, 1 \mapsto \mathit{List}[\alpha] \\ d_3 & = \alpha \mapsto \{\, \mathit{int}, \mathit{float} \,\} \end{cases}$$

Concretizations & Abstract Operators

Concretization:

$$\gamma : \mathcal{D}^\sharp \to \mathcal{P}(\mathcal{E} \times \mathcal{H})$$

Concretizations & Abstract Operators

Concretization:

$$\gamma : \mathcal{D}^\sharp \to \mathcal{P}(\mathcal{E} \times \mathcal{H})$$

Abstract Operators:

- Subset testing
- Join ($\sqcup$)
- Widening
- Filtering
  - instances
  - attributes

Concretizations & Abstract Operators

Concretization:

$$\gamma : \mathcal{D}^\sharp \to \mathcal{P}(\mathcal{E} \times \mathcal{H})$$

Abstract Operators:

- Subset testing
- Join ($\sqcup$)
- Widening
- Filtering
  - instances
  - attributes

Soundness:

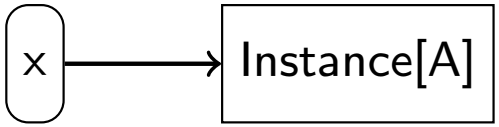
$$\forall d, d' \in \mathcal{D}^\sharp, \gamma(d) \cup \gamma(d') \subseteq \gamma(d \sqcup d')$$

Examples

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```

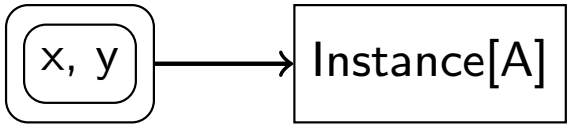


$$\begin{cases} d_1 & = x \mapsto (0, \bot) \\ d_2 & = 0 \mapsto \mathit{Instance}[A] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```

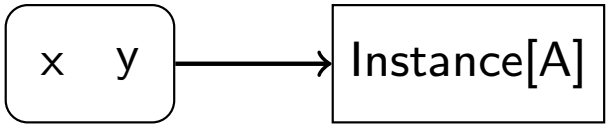


$$\begin{cases} d_1 & = x \mapsto (0, a), y \mapsto (0, a) \\ d_2 & = 0 \mapsto \mathit{Instance}[A] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```

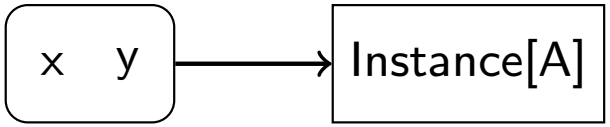


$$\begin{cases} d_1 & = x \mapsto (0, \bot), y \mapsto (0, \bot) \\ d_2 & = 0 \mapsto \mathit{Instance}[A] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```

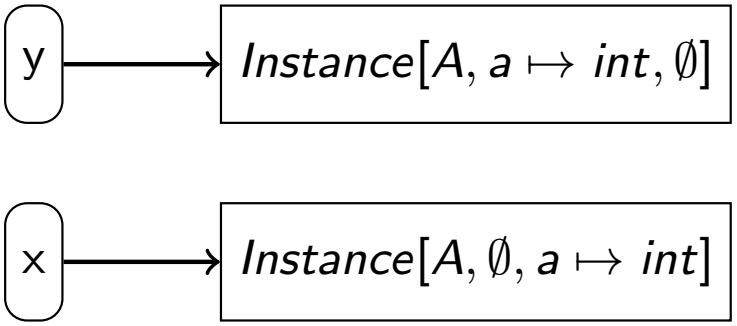


$$\begin{cases} d_1 & = x \mapsto (0, \bot), y \mapsto (0, \bot) \\ d_2 & = 0 \mapsto \mathit{Instance}[A] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```



$$\begin{cases} d_1 & = y \mapsto (0, \bot), x \mapsto (1, \bot) \\ d_2 & = 0 \mapsto \mathit{Instance}[A, a \mapsto \mathit{int}, \emptyset], \\ & \quad 1 \mapsto \mathit{Instance}[A, \emptyset, a \mapsto \mathit{int}] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
z = x.a
```

Attribute Error:

$$x \mapsto \mathit{Instance}[A, \emptyset, a \mapsto \mathit{int}]$$

$$\begin{cases} d_1 & = y \mapsto (0, \bot), x \mapsto (1, \bot), \mathit{exn_var} \mapsto (2, \bot) \\ d_2 & = \ldots, 2 \mapsto \mathit{Instance}[\mathit{AttributeError}] \\ d_3 & = _ \end{cases}$$

Examples

```
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("...")

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

Interprocedural analysis?

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining
- Bottom-up analysis (as in OCaml)

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining
- Bottom-up analysis (as in OCaml)
- Summary-based, top-down analysis

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining
- Bottom-up analysis (as in OCaml)
- **Summary-based, top-down analysis**

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining
- Bottom-up analysis (as in OCaml)
- **Summary-based, top-down analysis**

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

Examples

```
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("...")

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

Interprocedural analysis?

- Inlining
- Bottom-up analysis (as in OCaml)
- **Summary-based, top-down analysis**

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

$$\Longrightarrow \; Summary[f, (InputArgs, OutputArgs)\ list] \in V^{\sharp}$$

Examples

```
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("...")

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

$Summary[fspath, []]$

Examples

```
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("...")

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

$Summary[fspath, [(str, str)]]$

Examples

```
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("...")

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

$$F = Instance[FSPpath, [__fspath__], \emptyset]$$

$$Summary[fspath, [\quad (str, str) \quad (F, str)]]$$

Examples

```
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("...")

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

$$F = \mathit{Instance}[\mathit{FSPpath}, [__\mathit{fspath}__], \emptyset]$$

$$\begin{aligned} \mathit{Summary}[\mathit{fspath}, [\quad & (\mathit{str}, \mathit{str}) \\ & (F, \mathit{str}) \\ & (\mathit{bytes}, \mathit{bytes})]] \end{aligned}$$

$\rightsquigarrow$

$$\begin{aligned} \mathit{Summary}[\mathit{fspath}, [\quad & (\alpha, \alpha), \alpha \in \{\, \mathit{str}, \mathit{bytes} \,\} \\ & (F, \mathit{str})]] \end{aligned}$$

Implementation

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

Implementation into MOPSA

In the type analysis, the following abstract domains are used:

- Generic control-flow ones
- One handling function calls through inlining
- Generic ones handling Python objects, class, functions, ...

Implementation into MOPSA

In the type analysis, the following abstract domains are used:

- Generic control-flow ones
- One handling function calls through inlining
- Generic ones handling Python objects, class, functions, ...
- Abstract domain of types

Implementation into MOPSA

In the type analysis, the following abstract domains are used:

- Generic control-flow ones
- One handling function calls through inlining
- Generic ones handling Python objects, class, functions, ...
- Abstract domain of types

About 1500 lines of added OCaml.

✓ I was able to reuse some abstract domains.

✗ Some abstract domains were not modular enough, I generalized them or wrote alternatives.

Related Work

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

Conclusion

Conclusion

- Design of a static type analysis by abstract interpretation
- Relational abstract domain, supporting bounded parametric polymorphism
- Concretization function, giving concrete meaning to the abstract domain

Conclusion

- Design of a static type analysis by abstract interpretation
- Relational abstract domain, supporting bounded parametric polymorphism
- Concretization function, giving concrete meaning to the abstract domain

Future work:

- Implementation (summary-based analysis, larger Python support)
- Combining this analysis with the value analysis

Appendix

Non-Goals

We are not designing a type system.

- No rejection of programs, only error collection.

Non-Goals

We are not designing a type system.

- No rejection of programs, only error collection.
- Abstract interpretation based soundness proof.

Non-Goals

We are not designing a type system.

- No rejection of programs, only error collection.
- Abstract interpretation based soundness proof.
- Our analysis is naturally flow-sensitive.

Non-Goals

We are not designing a type system.

- No rejection of programs, only error collection.
- Abstract interpretation based soundness proof.
- Our analysis is naturally flow-sensitive.
- Our analysis is not as modular as most type systems are.

Environments

$$\mathcal{E} \stackrel{\text{def}}{=} \textbf{Id} \rightharpoonup \textbf{Addr} \cup \texttt{Undef}$$

$$\mathcal{H} \stackrel{\text{def}}{=} \textbf{Addr} \rightharpoonup \textbf{Obj} \times \textbf{Val}$$

$$\mathcal{F} \stackrel{\text{def}}{=} \{\, \mathit{cur}, \mathit{ret}, \mathit{brk}, \mathit{cont}, \mathit{exn} \,\}$$

$$\textbf{Id} \subseteq \mathit{string}$$

$$\textbf{Obj} \stackrel{\text{def}}{=} \mathit{string} \rightharpoonup \textbf{Addr}$$

$$\textbf{Val} \stackrel{\text{def}}{=} \mathbb{Z} \cup \mathit{string} \cup \{\texttt{True}, \texttt{False}, \texttt{None}, \texttt{NotImpl}\}$$

---

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

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
try:
  z = x.a
except AttributeError:
  z = y.a
```

$$exn, \begin{cases} d_1 & = y \mapsto (0, \bot), x \mapsto (1, \bot), exn_var \mapsto (2, \bot) \\ d_2 & = \ldots, 2 \mapsto Instance[AttributeError] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
try:
  z = x.a
except AttributeError:
  z = y.a
```

$$cur, \begin{cases} d_1 & = y \mapsto (0, \bot), x \mapsto (1, \bot) \\ d_2 & = 0 \mapsto Instance[A, a \mapsto int, \emptyset], \\ & \quad 1 \mapsto Instance[A, \emptyset, a \mapsto int] \\ d_3 & = _ \end{cases}$$

Examples

```
class A: pass

x = A()
if *:
  y = x
else:
  y = A()
y.a = 2
try:
  z = x.a
except AttributeError:
  z = y.a
```

$$cur, \begin{cases} d_1 & = y \mapsto (0, \bot), x \mapsto (1, \bot), z \mapsto (2, \bot) \\ d_2 & = \ldots, 2 \mapsto int \\ d_3 & = _ \end{cases}$$

Value Analysis of Python

Value Analysis[3]:

- incomparable with our analysis (due to the relationality)

[3] 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[3]:

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

[3] 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[3]:

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

[3] 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.

Other Type Analyses

- Typpete [4]: uses an SMT solver to perform type inference

[4] Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).
[5] 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 [4]: uses an SMT solver to perform type inference
- Work of Fritz and Hage[5]: analysis written in terms of data-flow equations

[4] Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).
[5] 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 [4]: uses an SMT solver to perform type inference
- Work of Fritz and Hage[5]: analysis written in terms of data-flow equations
- Pytype: from engineers at Google, no technical reference

[4] Hassan et al. "MaxSMT-Based Type Inference for Python 3". CAV (2).
[5] 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.