Karlsruher Institut für Technologie

Typinferenz und Vollständigkeitsüberprüfung für Structural Pattern Matching in Python

Adrian Freund

Lehrstuhl Programmierparadigmen, IPD Snelting

Typisiertes Python

```
def factorial(n      )        :
    res       = 1
    for i in range(n):
        res *= i + 1

    return res


n       = input("Please enter a number: ")
print(factorial(n))
```

Typisiertes Python

```python
def factorial(n: int) -> int:
    res       = 1
    for i in range(n):
        res *= i + 1

    return res


n       = input("Please enter a number: ")
print(factorial(n))
```

Typisiertes Python

```
def factorial(n: int) -> int:
    res: int = 1
    for i in range(n):
        res *= i + 1

    return res


n: int = input("Please enter a number: ")
print(factorial(n))
```

Typchecker für Python

- Kein statisches Typchecking in Python
- Verschiedene externe Typchecker
  - mypy
  - Pyright
  - Pyre
  - pytype

Typchecker für Python

```
def factorial(n: int) -> int:
    res: int = 1
    for i in range(n):
        res *= i + 1

    return res


n: int =     input("Please enter a number: ")
print(factorial(n))
```

Typchecker für Python

```python
def factorial(n: int) -> int:
    res: int = 1
    for i in range(n):
        res *= i + 1

    return res


n: int = input("Please enter a number: ")
print(factorial(n))
```

```
example.py:9: error: Incompatible types in assignment
(expression has type "str", variable has type "int")
Found 1 error in 1 file (checked 1 source file)
```

Typchecker für Python

```python
def factorial(n: int) -> int:
    res: int = 1
    for i in range(n):
        res *= i + 1

    return res


n: int = int(input("Please enter a number: "))
print(factorial(n))
```

```
example.py:9: error: Incompatible types in assignment
(expression has type "str", variable has type "int")
Found 1 error in 1 file (checked 1 source file)
```

Pattern Matching in Python

```
match event.get():
    case Click(position=(x, y)):
        handle_click_at(x, y)
    case KeyPress(key_name="Q") | Quit():
        game.quit()
    case KeyPress(key_name=k) if k.isdigit():
        game.select_slot(int(k))
    ...
    case KeyPress():
        pass # Ignore other keystrokes
    case _:
        raise ValueError()
```

Zielsetzung (1)

- Python 3.10 veröffentlich am 4. Oktober 2021
- Match-Statement noch nicht von Typcheckern unterstützt
- Ziele:
  - Typinferenz
    - Neu gebundene Variablen
    - Narrowing des Match-Subjekts
  - Vollständigkeitsüberprüfung
    - Insbesondere Enum-Typen

Zielsetzung (2)

```
from enum import Enum
class Color(Enum):
    RED = 0
    GREEN = 1
    BLUE = 2


def get_color_name(color: Color) -> str:
    match color:
        case Color.RED:
            return "Red"
        case Color.GREEN:
            return "Green"
        case Color.BLUE:
            return "Blue"
```

Zielsetzung (2)

```
from enum import Enum
class Color(Enum):
    RED = 0
    GREEN = 1
    BLUE = 2
    PURPLE = 3

def get_color_name(color: Color) -> str:
    match color:
        case Color.RED:
            return "Red"
        case Color.GREEN:
            return "Green"
        case Color.BLUE:
            return "Blue"
```

Formalisierung (1)

- Herleitungsbäume
  - Typumgebungen Γ, Γ_+ und Γ_-
  - Herleitungsregeln $case_p$ für Pattern-Art p

Γ

```
match s:
    case p:
        Γ+
    Γ−
    case p2:
        ...
```

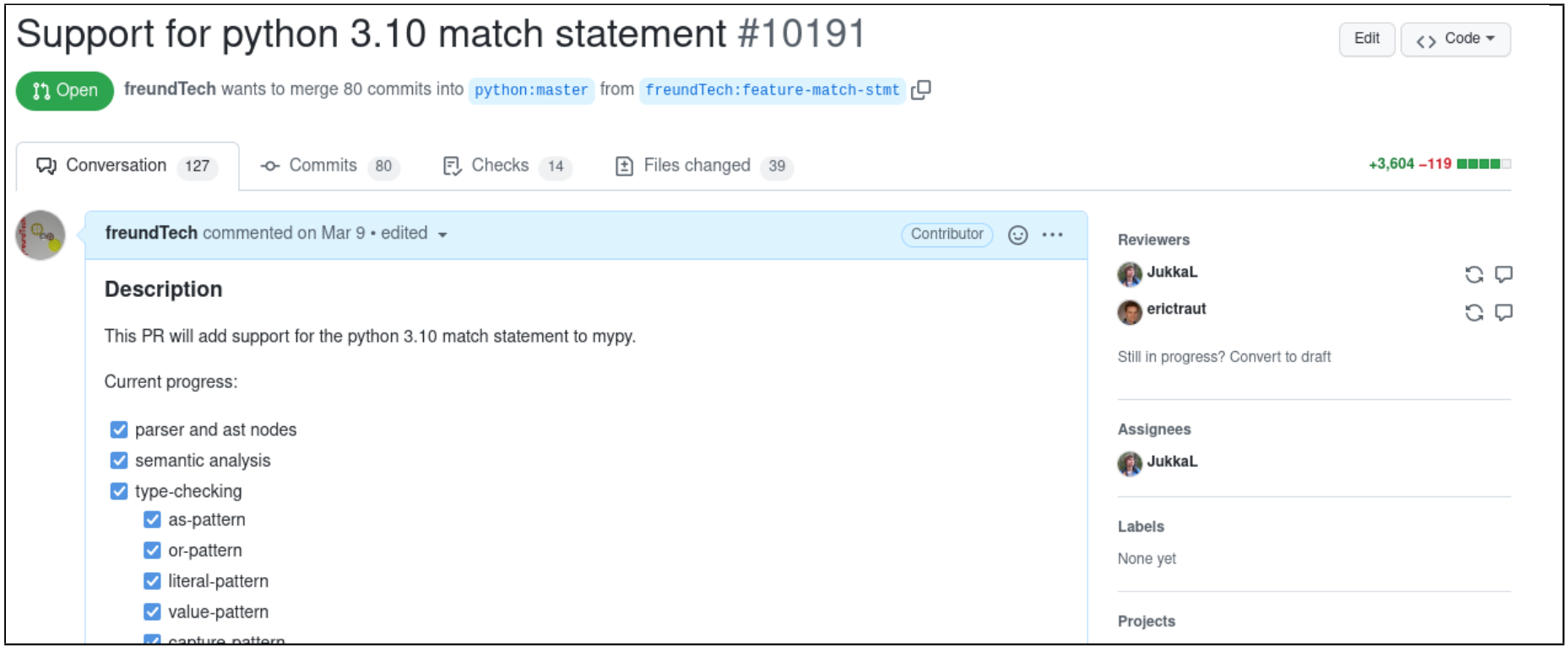
Formalisierung (2)

```
match s:
    case p1 as x:
        ...
```

$$case_{as}\frac{\Gamma \vdash s : \tau_s}{\Gamma_+ = \Gamma_{p_1+}, x : \tau_s \qquad \Gamma_- = \Gamma_{p_1-} \qquad \Gamma \vdash s_{p_1} : \tau_s}$$

Implementierung

- Auf Basis von Mypy
  - Integration in Upstream-Code
  - Pull-Request gerade in Review



Evaluation

- Testsuite mit 118 Test
- Manuelle Tests
  - CPython Testsuite
  - Scala
  - Pyright

Beispiele (1)

```
class A:
    a: Union[str, bytes]
    b: list[int]

m: object

match m:
    case A(a=str() as i, b=[x, y]):
        reveal_type(m)
        reveal_type(i)
        reveal_type(x)
        reveal_type(y)
```

Beispiele (1)

```
class A:
    a: Union[str, bytes]
    b: list[int]

m: object

match m:
    case A(a=str() as i, b=[x, y]):
        reveal_type(m)
        reveal_type(i)
        reveal_type(x)
        reveal_type(y)
```

```
eval1.py:9: note: Revealed type is "eval1.A"
```

Beispiele (1)

```
class A:
    a: Union[str, bytes]
    b: list[int]

m: object

match m:
    case A(a=str() as i, b=[x, y]):
        reveal_type(m)
        reveal_type(i)
        reveal_type(x)
        reveal_type(y)
```

```
eval1.py:9: note: Revealed type is "eval1.A"
eval1.py:10: note: Revealed type is "builtins.str"
```

Beispiele (1)

```
class A:
    a: Union[str, bytes]
    b: list[int]

m: object

match m:
    case A(a=str() as i, b=[x, y]):
        reveal_type(m)
        reveal_type(i)
        reveal_type(x)
        reveal_type(y)
```

```
eval1.py:9: note: Revealed type is "eval1.A"
eval1.py:10: note: Revealed type is "builtins.str"
eval1.py:11: note: Revealed type is "builtins.int"
eval1.py:12: note: Revealed type is "builtins.int"
```

Beispiele (2)

```
from typing import List, Union
m: Union[List[List[str]], str]

match m:
    case [list(['str'])]:
        reveal_type(m)
```

Beispiele (2)

```
from typing import List, Union
m: Union[List[List[str]], str]

match m:
    case [list(['str'])]:
        reveal_type(m)
```

```
eval2.py:6: note: Revealed type is
"Union[builtins.list[builtins.list[builtins.str]], str]"
```

Beispiele (2)

```
from typing import List, Union
m: Union[List[List[str]], str]

match m:
    case [list(['str'])]:
        reveal_type(m)
```

```
eval2.py:6: note: Revealed type is
"builtins.list[builtins.list[builtins.str]]"
```

Zusammenfassung

- Typüberprüfung mit externen Typcheckern
- Neues Match-Statement in Python 3.10
- Formalisierung mit Herleitungsbäumen
- Implementierung auf Basis von Mypy
  - Pull Request in Review

Arten von Pattern

- Literal-Pattern
- Capture-Pattern
- Wildcard-Pattern
- Value-Pattern
- Sequence-Pattern
- Mapping-Pattern
- Class-Pattern
- AS-Pattern
- OR-Pattern
- Group-Pattern

Literal-Pattern

```
match s:
    case 1:
        print('s matches literal 1')
    case "Hello world"
        print('s matches literal "Hello World"')
```

Code 1: Literal-Pattern

$$case_{literal}\ \frac{\Gamma \vdash c : \tau_c \qquad \Gamma \vdash s : \tau_s}{\Gamma_+ = \Gamma, s : cond_+(\tau_s, \tau_c) \qquad \Gamma_- = \Gamma, s : cond_-(\tau_s, \tau_c)}$$

Capture-Pattern

```
match s:
    case x:
        print("Match subject was " + x)
```

Code 2: Capture-Pattern

$$case_{capture} \frac{\Gamma \vdash s : \tau_s}{\Gamma_+ = \Gamma, x : \tau_s \qquad \Gamma_- = \Gamma, s : \bot}$$

Wildcard-Pattern

```
match s:
    case _:
        print("This always matches")
```

Code 3: Wildcard-Pattern

$$\textit{case}_{\textit{wildcard}}\ \frac{}{\Gamma_+ = \Gamma \qquad \Gamma_- = \Gamma, s : \bot}$$

Value-Pattern

```
match s:
    case HTTPStatus.OK:
        print("Match subject matches value of
              HTTPStatus.OK")
```

Code 4: Value-Pattern

$$case_{value}\ \frac{\Gamma \vdash v : \tau_v \qquad \Gamma \vdash s : \tau_s}{\Gamma_+ = \Gamma, s : cond_+(\tau_s, \tau_v) \qquad \Gamma_- = \Gamma, s : cont_-(\tau_s, \tau_v)}$$

Sequence-Pattern

```
match s:
    case [p1, p2]:
        print("s matches Sequence [p1, p2]")
```

Code 5: Sequence-Pattern

$$case_{sequence_1} \frac{\Gamma \vdash s : Sequence[x]}{\Gamma \vdash s_{p_1} : x \wedge \cdots \wedge \Gamma \vdash s_{p_n} : x}$$

$$case_{sequence_2} \frac{\Gamma \nvdash s : Sequence[x]}{\Gamma \vdash s_{p_1} : \top \wedge \cdots \wedge \Gamma \vdash s_{p_n} : \top}$$

$$case_{sequence_3} \frac{\Gamma \vdash s : \tau_s \qquad \Gamma_{p_1+} \vdash p_1 : \tau_{p_1} \ \cdots \ \Gamma_{p_n+} \vdash p_n : \tau_{p_n}}{\Gamma_+ = (\bigcup_{x=1}^{n} \Gamma_{p_n+}), s : cond_+(\tau_s, Sequence[join(\tau_{p_1} \cdots \tau_{p_n})]) \qquad \Gamma_- = \Gamma}$$

Mapping-Pattern

```
match s:
    case {1: p1, "foo": p2}:
        print("s matches Sequence [p1, p2]")
```

Code 6: Mapping-Pattern

$$\textit{case}_{\textit{mapping}_1} \frac{\Gamma \vdash k_i : \tau_k \qquad \Gamma \vdash s.get : x \to y \qquad \tau_k \leq x}{\Gamma \vdash s_{p_n} : y}$$

$$\textit{case}_{\textit{mapping}_2} \frac{}{\Gamma_+ = \bigcup_{x=1}^{n} \Gamma_{p_n +} \qquad \Gamma_- = \Gamma}$$

Class-Pattern

```
match s:
    case A(p1, attr=p2):
        print("s is of class A with attributes
               matching p1 and p2")
```

Code 7: Class-Pattern

$$case_{class_1} \frac{\Gamma \vdash s.a_i : x}{\Gamma \vdash s_{p_n} : x}$$

$$case_{class_2} \frac{\Gamma, s : \tau_s \qquad \Gamma_{p_1-} \vdash s : \bot \ \ldots \ \Gamma_{p_n-} \vdash s : \bot}{\Gamma_+ = \Gamma, s : cond_+(\tau_s, t) \qquad \Gamma_- = \Gamma, cond_-(\tau_s, t)}$$

$$case_{class_3} \frac{\Gamma, s : \tau_s \qquad \Gamma_{p_i-} \nvdash s : \bot}{\Gamma_+ = \Gamma, s : cond_+(\tau_s, t) \qquad \Gamma_- = \Gamma}$$

AS-Pattern

```
match s:
    case p1 as x:
        print("Match-Subject was " + x)
```

Code 8: AS-Pattern

$$case_{as}\frac{\Gamma \vdash s : \tau_s}{\Gamma_+ = \Gamma_{p_1+}, x : \tau_s \qquad \Gamma_- = \Gamma_{p_1-} \qquad \Gamma \vdash s_{p_1} : \tau_s}$$

OR-Pattern

```
match s:
    case p1 | p2:
        print("p1 or p2 matches")
```

Code 9: OR-Pattern

$$case_{or_1} \frac{\Gamma \vdash s : \tau_s}{\Gamma \vdash s_{p_1} : \tau_s}$$

$$case_{or_2} \frac{\Gamma_{p_i-} \vdash s : \tau_s}{\Gamma \vdash s_{p_{i+1}} : \tau_s}$$

$$case_{or_3} \frac{\Gamma_{p_1+} \vdash s : \tau_{p_1} \quad \ldots \quad \Gamma_{p_n+} \vdash s : \tau_{p_n}}{\Gamma_+ = (\bigcup_{i=1}^{n} \Gamma_{p_i+}), s : \mathit{UnionType}(\tau_{p_1} \ldots \tau_{p_n})}$$

$$case_{or_4} \frac{\Gamma_{p_n 1-} \vdash s : \tau_{p_n}}{\Gamma_- = \Gamma, s : \tau_{p_n}}$$

Group-Pattern

```
match s:
    case (p1):
        print("s matches subpattern p1")
```

Code 10: Group-Pattern

$$case_{group}\ \frac{\Gamma_+ = \Gamma_{p_1+} \qquad \Gamma_- = \Gamma_{p_1-}}{}$$