pyanalyze: a semi-static typechecker

Jelle Zijlstra, November 2021
Quora

What is it?

- Typechecker
- Written in Python
- Supports Python 3.6 through 3.10
  - Though mostly 3.6 because that's what we run
- Takes 13 mins to typecheck 2.6M lines
- https://github.com/quora/pyanalyze
- Development is driven by finding issues that cause bugs in Quora's codebase

What, another typechecker?

- Started in 2015, before most other type checkers
- asynq support was a major motivator to keep rolling our own
- Customizable

```
@asynq()
def all_author_names(aids: List[Aid]) -> List[str]:
    uids = yield [author_of_answer.asynq(aid) for aid in aids]
    names = yield [name_of_user.asynq(uid) for uid in uids]
    return names
```

What makes it different?

- It imports your code!
- But then it uses the AST for type checking
- Uses the runtime function/class objects for getting signatures and annotations
- Only looks at one module at a time

Advantage: Understanding dynamic code

- pyanalyze has no special casing for the `__init__` method on dataclasses, because it sees the generated runtime `__init__` method

```
@dataclass
class ActualArguments:
    positionals: List[Composite]
    star_args: Optional[Value]  # represents
    keywords: Dict[str, Tuple[bool, Composit
    star_kwargs: Optional[Value]  # represen
    kwargs_required: bool
```

Advantage: Calling into user code

- ABC registration, runtime-checkable protocols work automatically
- Plugins can call functions in user code
  - e.g. to look up the database schema

But it’s still a static checker

Unlike a dynamic checker, pyanalyze:

- Checks every code path
- Can track extra information (e.g., NewType)

Disadvantages

- Hard to make an incremental mode
- Can’t check scripts that do work when you import them
- Hard to see which attributes exist on a class
  - Though this is probably fixable
- Can’t understand runtime `@overload`

Typechecking style

- Frequently infers literal types
  - Modules and functions are represented internally as literals
- Allows types to change throughout a function
- Mostly ignores variance
  - Just don’t mutate lists passed as arguments

Supported checks

- Missing `await`
- Names that are undefined in some code paths
- Boolean operations on non-boolable types
  - `if is_it_true:` vs. `if is_it_true():`
- Missing f in an f-string
  - Though the heuristics need more tuning
- Lots of asynq-specific checks
- Unused code finder

Extensions

- Literal supports all types
  - `Literal[some_function]` means a compatible Callable, but this hasn't proven very useful
- `ParameterTypeGuard`: like `TypeGuard`, but works on any parameter
- `no_return_unless`: like `TypeGuard`, but function throws unless the condition is met
- `ExternalType`: reference to non-imported types

Extensions: CustomCheck

- Allows user code to be called and perform arbitrary checks
- Use cases:
  - Allow only literals
  - Allow only picklable objects
  - Disallow Any

CustomCheck example: LiteralOnly

- Exposed as `pyanalyze.extensions.LiteralOnly`
- Implementation just flattens Unions (with flatten_values), then errors for anything other than KnownValue (=Literal)
- To use it: `Annotated[str, LiteralOnly()]`

```python
@dataclass(frozen=True)
class LiteralOnly(CustomCheck):
    def can_assign(self, value: "Value", ctx: "CanAssignContext") -> "CanAssign":
        for subval in pyanalyze.value.flatten_values(value):
            if not isinstance(subval, pyanalyze.value.KnownValue):
                return pyanalyze.value.CanAssignError("Value must be a literal")
        return {}
```

What's missing?

A lot:

- Protocol
- @overload
- ParamSpec
- TypeVar bounds
- match/case

Questions?