Python Type Hints Are Turing Complete (Artifact)

Ori Roth
Department of Computer Science, Technion, Haifa, Israel

Abstract

The artifact comprises a Docker image (virtual environment) containing the source code and experiments setup mentioned in the paper. The artifact is available on Zenodo[1]. The anonymous version submitted to the ECOOP Artifact Evaluation Committee (AEC) is also available on Zenodo[2]. The project is maintained on GitHub[3].

2012 ACM Subject Classification Software and its engineering → General programming languages

Keywords and phrases nominal Subtyping with Variance, Python

Digital Object Identifier 10.4230/DARTS.9.2.1

Related Article Ori Roth, "Python Type Hints Are Turing Complete", in 37th European Conference on Object-Oriented Programming (ECOOP 2023), LIPIcs, Vol. 263, pp. 44:1–44:15, 2023.
`https://doi.org/10.4230/LIPIcs.ECOOP.2023.44`

Related Conference 37th European Conference on Object-Oriented Programming (ECOOP 2023), July 17–21, 2023, Seattle, Washington, United States

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2023 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The artifact contains:
1. The Python implementation of Grigore's reduction mentioned in Sections 2 and 5 of the paper.
2. The Python implementation of our real-time simulation described in Section 4 and mentioned in Section 5.
3. The experiment setup for obtaining the data depicted in Figure 3 and Table 3.
4. An example program supporting the claim that Pyright is unsound, made in Table 3 and Section 3.

2 Content

The artifact package includes:
- A Docker image `python-typing-machines.tar`, containing:
  - The project source code in `/app/typing_machines/`.
  - The Figure 3 experiment setup in `/app/typing_machines/experiment/`.
  - Variance examples for Python type checkers in `/app/motivation/`.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://doi.org/10.5281/zenodo.7898753`.

[1] `https://doi.org/10.5281/zenodo.7898753`
[2] `https://doi.org/10.5281/zenodo.7004898`
[3] `https://github.com/OriRoth/python-typing-machines`

DAGSTUHL ARTIFACTS SERIES
Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany

4 Tested platforms

Install Docker (`https://www.docker.com/`) to use the artifact.

5 License

The artifact is available under the Apache License 2.0 (`https://www.apache.org/licenses/LICENSE-2.0`).

6 MD5 sum of the artifact

9f1a321947a77a3144d2dfc75f72a5b7

7 Size of the artifact

0.645 GiB

A Getting started

To install the Docker image run:

```
docker load -i python-typing-machines.tar
```

To run the Docker image run:

```
docker run --rm -it --entrypoint bash python-typing-machines:1.1
```

Then type the commands described below.

B Claims supported by the artifact

B.1 Artifact functionality

1. The Python implementation of Grigore's reduction mentioned in Sections 2 and 5 can be found in `/app/typing_machines/`. Run:

   ```
   cd /app/typing_machines/
   source /venvs/mypy/bin/activate
   python main.py Grigore abbabba > tm.py
   mypy tm.py
   rm tm.py
   deactivate
   ```

   These commands run Grigore's reduction on a Turing machine accepting palindromes over $\{a, b\}$ and input word *abbabba*. To modify the Turing machine, edit `/app/typing_machines/main.py`. To change the input word, edit the command above. In this case, Mypy reports no error since *abbabba* is a palindrome. If we change the input word to *abbabaa*, which is not a palindrome, Mypy reports an error.
2. The Python implementation of our real-time simulation described in Section 4 and mentioned in Section 5 can be found in `/app/typing_machines/`. To run our reduction, run the command in the previous bullet but replace `Grigore` with `Roth`.

3. The data in the graph depicted in Figure 2 can be obtained by running

```
cd /app/typing_machines/experiment/
source /venvs/mypy/bin/activate
python stack_size_experiment.py
deactivate
```

Expected output:

```
mypy requires 9M stack size with algorithm Grigore and palindrome of length 10
mypy requires 13M stack size with algorithm Grigore and palindrome of length 12
mypy requires 21M stack size with algorithm Grigore and palindrome of length 14
mypy requires 37M stack size with algorithm Grigore and palindrome of length 16
Grigore's results:
Grigore 10 9
Grigore 12 13
Grigore 14 21
Grigore 16 37
mypy requires 5M stack size with algorithm Roth and palindrome of length 10
mypy requires 5M stack size with algorithm Roth and palindrome of length 20
mypy requires 5M stack size with algorithm Roth and palindrome of length 30
mypy requires 6M stack size with algorithm Roth and palindrome of length 40
mypy requires 9M stack size with algorithm Roth and palindrome of length 50
mypy requires 9M stack size with algorithm Roth and palindrome of length 60
mypy requires 13M stack size with algorithm Roth and palindrome of length 70
mypy requires 21M stack size with algorithm Roth and palindrome of length 80
mypy requires 21M stack size with algorithm Roth and palindrome of length 90
Roth's results:
Roth 10 5
Roth 20 5
Roth 30 5
Roth 40 6
Roth 50 9
Roth 60 9
Roth 70 13
Roth 80 21
Roth 90 21
```

4. The data in Table 3 can be obtained by running the type checkers on the code in `/app/motivation`. To test the static type checkers run:

```
cd /app/motivation/static/
source /venvs/mypy/bin/activate
mypy inf_sub_static.py
deactivate
```

These commands run Mypy on a code with an infinite subtyping cycle. You can test the other type checkers by replacing `mypy` with `pyre`, `pyright`, `pytype`, and `pyanalyze` in the commands above (at two locations; run `pyre` instead of `pyre inf_sub_static.py`).

- Mypy fully supports variance so it gets a segmentation fault.
- Pyre fully supports variance so it gets an internal error.
- Pyright is unsound (see below).
- Pytype reports that it does not support variance.
- Pyanalyze reports no error because it does not support variance.

To test the dynamic type checkers run:

```
cd /app/motivation/dynamic/
source /venvs/pydantic/bin/activate
python inf_sub_pydantic.py
deactivate
```

These commands run Pydantic on a code with an infinite subtyping cycle. You can test the other type checkers by replacing `pydantic` with `pytypes`, `typeguard`, and `typical` in the commands above (at two locations).
- Pydantic reports no error because it does not support variance.
- Pytypes reports an unrelated error because it does not support generics.
- Typeguard reports no error because it does not support variance.
- Typical reports an unrelated error because it does not support generics with type forward refrences.

5. The claim in Table 3 and Section 3 that Pyright is unsound is proven by running the following commands:

```
cd /app/motivation/pyright/
source /venvs/mypy/bin/activate
mypy pyright_unsound.py
deactivate
source /venvs/pyre/bin/activate
pyre
deactivate
source /venvs/pyright/bin/activate
pyright pyright_unsound.py
deactivate
```

The code in `pyright_unsound.py` is typed correctly. Mypy and Pyre report no error when checking the file. Pyright reports a single, false error when checking the file.

B.2 Artifact reusability

- The implementation can easily be modified to use a different algorithm than the one described in Section 4 of the paper. A new algorithm should use a similar interface to that of `/app/typing_machines/compilers/compiler.py` and `/app/typing_machines/compilers/compiler_g.py`.
  You can add your new algorithm to the experiment by:
  - adding support for the new algorithm in `/app/typing_machines/app.py`, and
  - editing `/app/typing_machines/experiment/stack_size_experiment.py`.
- The implementation can easily be modified to use a different language than Python by editing the compilers in `/app/typing_machines/compilers/`.