Infering Python Types using SMT Solvers

{ Leron Reznikov, Yuval Steinhart }@UC Santa Barbara

Overview and Goals

Python is a dynamically typed language. It uses what is called duck typing, which means that Python does not validate types, and only checks that types have the proper attributes at runtime. This makes life for the programmer easier, allowing for the freedom to do operations that would not be permitted in languages with stricter type systems. Unfortunately, it prevents a significant amount of static analysis, most of which relies on static types to catch bugs before runtime.

For our project, we seek to mitigate this problem and get the best of both worlds by adding static type inference in Python. We build up a constraint system and pass it to the SMT solver and output a satisfying type mapping for the parameters and return value, if one exists. For now, our type analysis works only on simple programs, but we hope that it is an MVP that shows that our solution could feasibly be expanded to support a large subset of the language.

Future Work

As mentioned, we defer several of the challenges to future work. We hope, first, to provide support for container classes (other than strings, which we already support). Containers are challenging because their underlying type must be known in order to use them to reason. We also hope to support non-default types - this could be done by pre-generating attribtues of all known types of the program and encoding them as SMT constraints similar to primitive types. Finally, we also hope to perform program-level, rather than just function-level, reasoning, which requires significantly more program-flow analysis.

Acknowledgments

This project would not have been possible without the advice and guidance of Professor Yu Feng. Thank you for a wonderful quarter!

Challenges

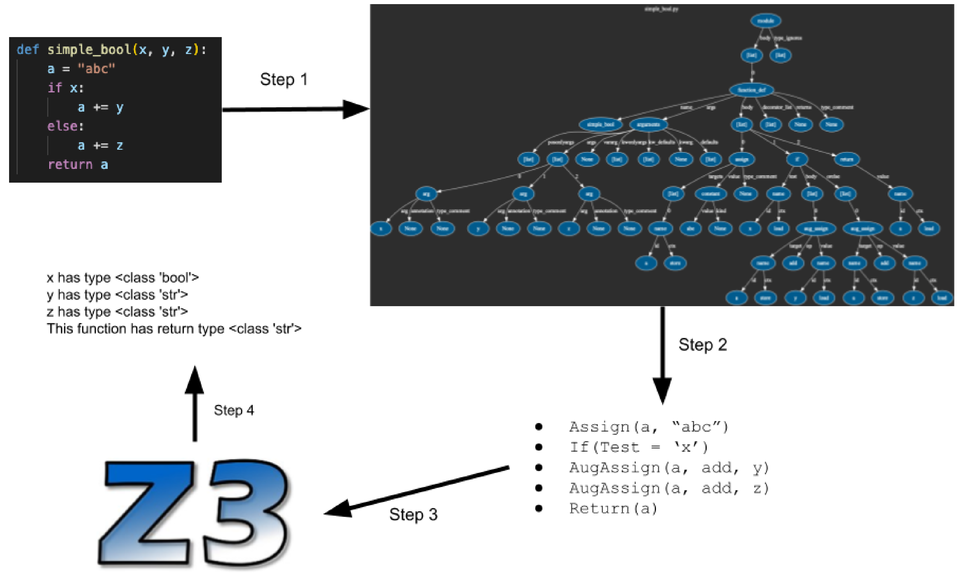
- ***Encode types for the SMT Solver:*** How can we represent types in a way that is understandable for the solver?
- ***Pass type constraints up the AST:*** If we know a particular constant or variable to be of a certain type, how can we "communicate" that if needed to a higher level of the AST?
- ***Subsetting the Language:*** Where do we start? What features of the language are most feasible to include to ensure functionality and present a viable MVP?

Our Solution

We use the following general steps as our framework:

1. Turn the function into an AST using Python's ast library
2. Traverse the AST depth-first. At each "interesting" (potential constraint-generating) node, we add the requisite constraints to the solver. We do this by mapping each node in the AST to a symbolic variable, imposing constraints as relationships between nodes and types. As a simple example, in our `visit_If()` function, we add the constraint that the `test` attribute, or the condition, must be a boolean.
3. Plug these constraints into Python's Z3 interface.
4. If there is a solution to the constraints, print them in a human-readable format. If not, print this fact as well.

Satisfying Example



Unsatisfiable Example