MaxSMT-Based Type Inference for Python 3

Mostafa Hassan, Caterina Urban, Marco Eiler, Peter Müller
Presentation by Jared Okun and John Hulton
Summary

- Using static typing for dynamically-typed languages
- Constraint generation
- Using Z3 to solve constraints
- Experimentation
- Conclusions
- Related and future work
- Critique
Motivation

- Increasing popularity of dynamically-typed languages
- Static typing offers
  - Early error detection
  - Efficient execution
  - Machine-checked code documentation
  - Static analysis and verification approaches
TYPPETE

- Whole-program type inference
- 3 code scans
  - Get classes, names, upper bounds
  - Create subtype and supertype axioms for each class
  - Generate SMT constraints for each dynamic method call
- Solve the SMT constraints using Z3 and MaxRes
  - An outside library
SMT and MAXSMT

- **SMT**
  - Determine if a given logical formula is satisfiable
  - For TYPPETE, determine if constraints can be satisfied
  - These constraints are generated from Axiomatic Semantics

- **MAXSMT**
  - Optional and mandatory clauses: maximize optional while satisfying all mandatory

- **Several different SMT solvers:**
  - Z3, SMT-LIB, etc.
Code Requirements

- Single-typed variables
- Homogenous generic types
- No dynamic code generation
The First Scan

- Globally defined names
- Classes and subclass relations
- Upper bounds on types
The Second Scan

- For each class, generate axiom constraints
- The subtype relations to and from each class
- None class prevents infinite matching loops (in SMT solver)
- Generic Types are represented by constructors and arguments (non-generics are constants)

\[
\forall t. \text{subtype}(\text{class}_{\text{odd}}, t) = (t = \text{class}_{\text{odd}} \lor t = \text{class}_{\text{item}} \lor t = \text{class}_{\text{object}})
\]

\[
\forall t. \text{subtype}(t, \text{class}_{\text{odd}}) = (t = \text{class}_{\text{none}} \lor t = \text{class}_{\text{odd}})
\]
The Third Scan

- Generate SMT constraints for expressions, statements and constructors (methods)
- For item1.compete(item2):

\[
(v_{item1} = \text{class}_{\text{Odd}} \land \text{compete}_{\text{Odd}} = f_2(\text{class}_{\text{Odd}}, \text{arg}, \text{ret}) \land \text{subtype}(v_{item2}, \text{arg}))
\lor (v_{item1} = \text{class}_{\text{Even}} \land \text{compete}_{\text{Even}} = f_2(\text{class}_{\text{Even}}, \text{arg}, \text{ret}) \land \text{subtype}(v_{item2}, \text{arg}))
\]
Third Scan Details

- Context map links names (fields, variables) to SMT variables
- Additional optional constraints for subtyping:
  - $v_{item2} = \text{arg}$
  - Improves performance by decreasing search space and increases precision
Solving the Constraints

- Use SMT solver Z3
- Use MaxRes to solve constraints
- Use e-matching to choose axiom constraints
  - Subtype choices propagate immediately
- Relaxed MaxSMT when MaxSMT unsatisfiable
  - Only subtype constraints are enforced, all others optional
Experiment

• **Specs**
  ○ 2.9 GHZ Intel i5
  ○ 8 GB RAM
  ○ Mac OS High Sierra version 10.13.3
  ○ Z3 version 4.5.1

• **Programs (slightly modified and added 2 errors)**
  ○ adventure (2 modules, 399 lines of code)
  ○ icemu (8 modules, 530 lines of code)
  ○ imp (7 modules, 771 lines of code)
  ○ scion (2 modules, 725 lines of code)
  ○ test suite (47 modules, 1998 lines of code)
## Results

<table>
<thead>
<tr>
<th></th>
<th>T(SMT)</th>
<th>T(MaxSMT)</th>
<th>Unfulfilled</th>
<th>T(Relaxed)</th>
<th>PYTYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>adventure</td>
<td>2.99s / 6.30s</td>
<td>3.27s / 6.76s</td>
<td>42 / 2</td>
<td>1.95s / 8.83s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>icemu</td>
<td>9.45s / 6.79s</td>
<td>9.51s / 3.63s</td>
<td>4 / 2</td>
<td>0.08s / 21.76s</td>
<td>18 [2]</td>
</tr>
<tr>
<td>imp</td>
<td>16.88s / 59.95s</td>
<td>16.91s / 15.87s</td>
<td>67 / 2</td>
<td>0.82s / 82.56s</td>
<td>3 [2]</td>
</tr>
<tr>
<td>scion</td>
<td>4.65s / 3.35s</td>
<td>4.72s / 2.97s</td>
<td>28 / 2</td>
<td>0.16s / 3.39s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>test suite</td>
<td>14.66s / 1.63s</td>
<td>14.66s / 2.17s</td>
<td>-</td>
<td>-</td>
<td>55 [34]</td>
</tr>
</tbody>
</table>

**Fig. 2.** Evaluation of TYPPETE on small programs and larger open source projects.

- Average runtime over 10 runs
- Unsatisfiable core size and unfulfilled constraints for MaxSMT
- Untyped variables and runnable modules from PYTYPE
Conclusions

<table>
<thead>
<tr>
<th></th>
<th>T(SMT)</th>
<th>T(MAXSMT)</th>
<th>Unfulfilled</th>
<th>T(Relaxed)</th>
<th>PYTYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>adventure</td>
<td>2.99s / 6.30s</td>
<td>3.27s / 6.76s</td>
<td>42 / 2</td>
<td>1.95s / 8.83s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>icemu</td>
<td>9.45s / 6.79s</td>
<td>9.51s / 3.63s</td>
<td>4 / 2</td>
<td>0.08s / 21.76s</td>
<td>18 [2]</td>
</tr>
<tr>
<td>imp</td>
<td>16.88s / 59.95s</td>
<td>16.91s / 15.87s</td>
<td>67 / 2</td>
<td>0.82s / 82.56s</td>
<td>3 [2]</td>
</tr>
<tr>
<td>scion</td>
<td>4.65s / 3.35s</td>
<td>4.72s / 2.97s</td>
<td>28 / 2</td>
<td>0.16s / 3.39s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>test suite</td>
<td>14.66s / 1.63s</td>
<td>14.66s / 2.17s</td>
<td>-</td>
<td>-</td>
<td>55 [34]</td>
</tr>
</tbody>
</table>

Fig. 2. Evaluation of TYPETTE on small programs and larger open source projects.

- Optional type equality constraints reduced Z3 time
- Constraint solving possibly faster with more modules
- TYPETTE fully typed all elements
  - Outperforms PYTYPE for static typing
Related Work

- Other Python type inference tools
  - STARKILLER
    - Infer sets of concrete types for each variable
    - Deriving valid type from set is non-trivial
  - MYPY
    - Perform incomplete type inference on local variables
    - Require type annotations for parameter and return types
  - PYANNOTATE
    - Dynamically tracks types during execution and annotates
    - Annotations not guaranteed to be sound
Related Work

- SMT solvers for type inference not new
  - F*
  - LiquidHaskell
  - Pavlinovic, et. al made one for OCaml
- TYPPETE follows their process
Future Work

● Infer gradual types (like PYTYPE)
● Heuristics for inferring annotations for generic types
● Split type inference into separate problems
Critique-The Process

● Heavy reliance on outside work without explanation
  ○ Z3, e-matching, MaxRes

● No relative execution time information
  ○ What does execution time scale with? How much?

● Lack of detail on implementation
  ○ Only described on a high level
Critique-The Experiment

- Missing comparisons with PYTYPE
  - Did not record runtime of PYTYPE
- Changes to programs could have affected PYTYPE
- Few programs and few trials
- Nonspecific hardware information
  - Give Mac model OR
  - CPU model
  - RAM speed and RAM config
  - Storage type ETC.
Critique-The Experiment

- Did not specify background processes
- Unclear results from table
- Did not compare to more related type inferencers
- Did not use varying error counts
- MaxSMT conclusion doesn’t factor Relaxed
- NO STATISTICAL TESTS

<table>
<thead>
<tr>
<th></th>
<th>T(SMT)</th>
<th>T(MaxSMT)</th>
<th>Unfulfilled</th>
<th>T(Relaxed)</th>
<th>Pytype</th>
</tr>
</thead>
<tbody>
<tr>
<td>adventure</td>
<td>2.99s / 6.30s</td>
<td>3.27s / 6.76s</td>
<td>42 / 2</td>
<td>1.95s / 8.83s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>icemu</td>
<td>9.45s / 6.79s</td>
<td>9.51s / 3.63s</td>
<td>4 / 2</td>
<td>0.08s / 21.76s</td>
<td>18 [2]</td>
</tr>
<tr>
<td>imp</td>
<td>16.88s / 59.95s</td>
<td>16.91s / 15.87s</td>
<td>67 / 2</td>
<td>0.82s / 82.56s</td>
<td>3 [2]</td>
</tr>
<tr>
<td>scion</td>
<td>4.65s / 3.35s</td>
<td>4.72s / 2.97s</td>
<td>28 / 2</td>
<td>0.16s / 3.39s</td>
<td>0 [0]</td>
</tr>
<tr>
<td>test suite</td>
<td>14.66s / 1.63s</td>
<td>14.66s / 2.17s</td>
<td>-</td>
<td>-</td>
<td>55 [34]</td>
</tr>
</tbody>
</table>

Fig. 2. Evaluation of TYPETE on small programs and larger open source projects.
Questions?
Source

- [https://github.com/caterinaurban/Typpete](https://github.com/caterinaurban/Typpete)