Finding Minimum Type Error Sources

Zvonimir Pavlinovic    Tim King    Thomas Wies

New York University
Motivating example

```ocaml
let f(lst: move list): (float*float) list = ...

let rec loop lst x y dir acc =
  if lst = [] then acc
  else
    print_string "foo"
  in
List.rev (loop lst 0.0 0.0 0.0 0.0 [(0.0,0.0)])
```

Error: This expression has type ‘a list but an expression was expected of type unit

[Zhang & Myers, POPL ‘14]
Problem?

• Type error reports often not helpful
  • increased debugging time
  • difficult to learn a language by novice programmers

• How to improve?
  • consider all error sources
  • rank them by some useful criterion
  • report top ranked sources to the programmer
Solutions?

• Previous research attempts:
  • slice of type inference deduction [Wand `86, Duggan & Bent `95]
  • program slice involved in error [Tip & Dinesh `01, Gast `05]
  • specially crafted type systems [Chitil `01, Neubauer & Thiemann `03, Chen & Erwig `14]

• Drawbacks:
  • focus on a single ranking criterion
  • focus on a specific type system
  • substantial compiler modifications
Challenge

• Can we enable compilers localize type errors
  • abstracting from a specific ranking criterion
  • for various type systems
  • with modest compiler modifications?

• In this work: *general framework for type error localization using constraint solving*
Error Source

An error source is a set of program expressions that, once corrected, yield a well-typed program
Definitions

Let $x = \text{"hi"}$ in $\quad$?

- Rank sources by some *useful* criterion
  - by assigning weights to expressions

**Minimum Error Source**

*An error source with minimum cumulative weight*
Ranking criteria - example

- Prefer error sources requiring fewer corrections?
  - assign weights equal to expression’s AST size

\[
\begin{align*}
&? \ (5) \\
&? \ (1) \\
&? \ (3) \\
&? \ (1) \\
&? \ (5)
\end{align*}
\]
Problem definition

Computing Minimum Error Sources

Given a program and a compiler-provided ranking criterion, find a minimum error source subject to the criterion

• How?
  • search through all hole versions of an input program
  • reduction to constraint solving: weighted maximum satisfiability modulo theories
Framework

• Support for various type systems due to SMT
• Modest compiler modifications
• Abstracts from specific ranking criteria
Weighted MaxSMT

**Input:**
- hard clauses
  - must hold
- soft clauses
  - each clause assigned a weight
- each clause belongs to a fixed first-order theory
  - algebraic data types, linear integer arithmetic, ...

**Output:**
- A satisfiable subset of soft clauses with maximum cumulative weight (hard clauses must hold)
Reduction to MaxSMT

\[
\begin{align*}
T_{let} & \implies \left( \alpha_{let} = \alpha_o \land 
\right. \\
T_x & \implies \left( \alpha_x = \text{string} \land 
\right. \\
T_{app} & \implies \left( \alpha_{app} = \text{fun}(\alpha_i, \alpha_o) \land 
\right. \\
T_{not} & \implies \left( \alpha_{not} = \alpha_{app} \land 
\right. \\
T_i & \implies \left( \alpha_i = \alpha_x \right) \land \\
T_{not \; impl} & \implies \left( \alpha_{not} = \text{fun}(\text{bool}, \text{bool}) \right) \land \\
&T_{let} \land T_x \land T_{app} \land T_{not} \land T_i \land T_{not \; impl}
\end{align*}
\]
Implementation

- Subset of OCaml (Caml)
  - Hindley-Milner type system
- Constraint generation using EasyOCaml
  - constraint encoded in the theory of inductive data types
- Weighted MaxSMT procedures using Sat4j and CVC4
  - circa 500 lines of Java code
- Evaluation on ~350 programs from [Lerner et al., `07]
Evaluation

15% increase in accuracy over OCaml

- ranking criterion: size of expression

<table>
<thead>
<tr>
<th>Program Size in LOC</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-50</td>
<td>47</td>
</tr>
<tr>
<td>50-100</td>
<td>102</td>
</tr>
<tr>
<td>100-150</td>
<td>65</td>
</tr>
<tr>
<td>150-200</td>
<td>57</td>
</tr>
<tr>
<td>200-250</td>
<td>53</td>
</tr>
<tr>
<td>250-300</td>
<td>28</td>
</tr>
<tr>
<td>300-350</td>
<td>3</td>
</tr>
<tr>
<td>350-400</td>
<td>1</td>
</tr>
</tbody>
</table>

seconds

min
median
max

program size in LOC
Constraint size

![Graph showing the relationship between program size and number of assertions. The x-axis represents program size in lines of code, ranging from 0 to 250. The y-axis represents the number of assertions, ranging from 10 to 100,000. The graph displays a scatter plot with data points indicating the constraint size.](image-url)
Performance improvements

• Type checking EXPTIME-complete [Mairson `90, Kfoury et al. `90]

• Solution?
  • Lazy quantifier-based instantiation
  • Lazy unification-based instantiation

• Further optimizations
  • Constraint slicing
  • Preemptive cutting
Contributions

• Clean problem formulation
  • type error localization as an optimization problem
  • abstracts from ranking criteria

• General algorithm
  • reduction to weighted MaxSMT constraint solving
  • supports various type systems due to SMT
  • modest compiler modifications

Thank you!