Finding Minimum Type Error Sources

Zvonimir Pavlinovic (New York University)

Motivation

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)])

Compilers consider only a single potential error source. Therefore, type error reports are often not useful. Better: rank error sources by some useful criterion and then show the top ranked sources to the programmer.

Problem Definition

Error source: a set of program expressions that, once corrected, yield a well-typed program.

let x = "hi" in not x  (not well-typed)
let x = "hi" in not x  (a hole expression representing a generic fix)

Compilers provide a ranking criterion to prefer type error sources of particular interest. A ranking criterion assigns weights to program expressions. A smaller weight indicates that the expression more likely contributes to the error. For example, to prefer error sources that require fewer corrections, assign a weight equal to the expression’s size.

Finding minimum type error sources problem: given an input program and a compiler-provided ranking criterion, find a minimum error source subject to the criterion.

How? Reduce the problem to weighted maximum satisfiability modulo theories (MaxSMT). We propose a general framework for type error localization using constraint solving.

Framework

Input program and ranking criterion
Typing Constraint Generation
Weighted MaxSMT Solver

Compiler

minimum error source

other potential error sources

Weighted MaxSMT

Let $S$ and $H$ be two sets of clauses over some fixed first-order theory where each clause from $S$ is assigned a weight. The weighted MaxSMT problem is to find a subset $N$ of $S$ with maximum cumulative weight such that $N \cup H$ is satisfiable. The clauses in $S$ are referred to as soft clauses, and the clauses in $H$ hard clauses.

Constraint Generation

The constraint generation is done using typing rules. The constraint for our running example is as follows.

$$ST \equiv T_{let} \land T_x \land T_{app} \land T_{not} \land T_i \land T_{not \impl}$$

$$h \land 5 \land 1 \land 3 \land 1 \land 1 \land 1$$

Implementation and Evaluation

- Framework instantiated for a subset of OCaml (Hindley-Milner type system)
- Typing constraint generated using the EasyOCaml system
- Weighted MaxSMT solver implemented using CVC4 and Sat4j
- Implementation evaluated on ~350 OCaml scripts from [Lerner et al., ’07]

Contributions

A general framework for type error localization that
- abstracts from the particular ranking criterion
- supports various type systems by appropriately instantiating the SMT solver
- requires no substantial compiler modifications due to use of SMT solvers