General Type Error Diagnostics Using MaxSMT

Zvonimir Pavlinovic (graduate student)
New York University (zvonimir@cs.nyu.edu)
251 Mercer Street, Room 410, New York, 10012, NY
ACM student number: 9628507 Adviser: Thomas Wies

1. Problem and Motivation

Automatic type inference is a popular feature of functional programming languages. Unfortunately, if the compiler cannot infer a correct typing of a program, the produced error message does not always help the programmer to fix the error. The compiler typically reports a single program location in the error message. This location is the point where the type inference failed, but not necessarily the actual cause of the error. Other potential error causes are not even considered. Hence, the compiler often misses the true error cause and, consequently, increases debugging time for the programmer. The goal of this work is to provide high quality type error diagnostics that can help speed up the software development process.

2. Background

Typical type inference algorithms infer types of program expressions on the fly. If a currently analyzed program expression is used with a different type than previously, the inference algorithm immediately stops and reports an error at the current program location. Although fast in practice, this approach often produces poor error diagnostics. In particular, it might be the case that a programmer made a mistake with the previous usages of the offending expression, or with some other related expressions. For example, consider the following simple OCaml program:

```ocaml
1 type 'a lst = Null | Cons of 'a * 'a lst
2 let x = Cons (3, Null)
3 4 let _ = print_string x
```

The standard OCaml compiler [2] reports a type mismatch error for expression `x` on line 5. However, the programmer might have defined `x` incorrectly on line 4 or misused the `print_string` function. As `x` is defined just before the error line, it seems more likely that the error is caused by a misuse of `print_string`. In fact, the student author of this code confirmed that this is the real source of the error. This simple example clearly suggests that compilers need to consider several possible error causes and rank them by their relevance in order to generate useful error reports. Hence, there is a need for an infrastructure that (1) supplies compilers with all possible error sources and (2) can incorporate compiler-specific ranking heuristics that report the location that is most likely the real error cause. In this work, we propose a general constraint solving framework that provides such an infrastructure.

3. Approach and Uniqueness

Unlike typical type inference algorithms, we do not simply stop the type inference as soon as a type inconsistency is observed. Instead, we provide compilers with all minimal sets of expressions each of which, once corrected, yields a type correct program.

Our approach builds on existing work that rephrases type inference as a constraint solving problem [3, 15]. Program expressions are assigned with type variables, where typing information of a program is captured in terms of constraints over those variables. If an input program has a type error, then the corresponding set of typing constraints is unsatisfiable. The crux of our approach is to use MaxSMT solvers to extract all maximal satisfiable subsets of such typing constraints. As constraints directly map to program expressions, the complements of these maximal sets represent all minimal sets of program expressions that may have caused the type error. Our approach is inspired by the Bug-Assist tool [11], which uses a MaxSAT procedure for fault localization in imperative programs.

The use of SMT solvers has several additional advantages. First, it allows support for a variety of type systems by instantiating the MaxSMT solver with an adequate reasoning theory. Second, our framework provides a mechanism for incorporating compiler-specific heuristics for filtering and ranking the error causes. In particular, compilers can assign weights to constraints that correspond to specific program expressions. A weighted MaxSMT procedure [7] then finds all satisfiable subsets of the constraints with maximal cumulative weight, effectively favoring causes of particular interest. Finally, the framework does not introduce a substantial implementation overhead since the MaxSMT solver can be used as a black box.

4. Related Work

Previous research on localization of type errors mainly focused on designing concrete systems for generating quality type error messages. Existing approaches range from showing a relevant portion of a failed type inference trace [8, 17], a program slice involved in the error [9, 16], to specially crafted type systems [5, 6, 13]. Each of these approaches either presents questionably useful information to programmers, requires significant compiler modifications, or fails to consider all possible error sources. More closely related to our approach is the Seminal [12] tool, which computes several possible error sources by repeated calls to the type checker. This approach, however, heuristically searches for all error sources, providing no formal guarantee that it finds all of them. Myers et al. [18] encode typing information for Hindley-Milner type systems in terms of graphs, which can then be analyzed in order to find locations that should be reported to programmers. It is unclear how this approach would support more expressive type systems. Previous approaches based on constraint solving are either unable to enumerate all minimal error sources [10] or are designed for a specific type system [14].

5. Results and Contribution

Our technical contributions can be summarized as follows:
We propose a novel framework for producing quality type error diagnostics. The framework supplies compilers with all minimal error sources by employing MaxSMT solvers.

Our framework supports incorporating compiler-specific ranking heuristics for filtering a source most likely to be a true error cause.

Also, the framework supports a variety of type systems without introducing substantial complexity to existing language implementations.

We have implemented an instance of our framework targeted at Hindley-Milner style type systems. Here, type inference reduces to constraint satisfaction modulo the theory of algebraic data types. Our implementation builds on top of the EasyOCaml [1] system and the SMT solver CVC4 [4]. We have evaluated our implementation on existing OCaml benchmarks [12] for type error localization. Our experiments show that our approach can efficiently produce all minimal sources of a type error for real world programs.

References


