Improving Compiler Errors Using Type Annotation

Jiani Huang, Soomin Lee, Yijun Zhang
UCSD CSE Early Research Scholars Program (ERSP)

**Problem Statement**
It is well-known in community that functional programming languages are challenging, but less is understood about why. Our goal is to understand how beginners learn OCaml and improve compiler error messages for type errors to make their learning process more efficient.

**Background**
There are many many differences between object-oriented and functional programming languages (FP) that become bottlenecks for students learning the latter.

1. Types - Java vs. OCaml:
   Types are explicitly declared in Java, but not in OCaml; all types are completely inferred.

2. Iteration vs. recursion:
   Iteration (loops) in FP are usually achieved through recursion and high-order functions, where the functions can be passed in as arguments.

**Data Collection**
In order to track students’ progress, we collected data from homework assignments from CSE 130, which included snapshots of their entire file every thirty seconds. Collected as JSON files, they included the following:

- Unix timestamp
- Body of file
- Section of code that was send to compiler
- Output error message, if any

**Data Analysis**
Initial data analysis was done to see how much time the students spend on each homework problem to find out which concepts they struggle with the most.

Time taken to complete a method vs. Percentage of students who completed the problem.

Students struggled with the first problem (blue) the most. Only 70% of students were able to compile it within a reasonable amount of time.

We decided to look at compiler errors which can be broken down into two different categories – type and syntax.

And from our collected data, we found out that type errors were the more challenging form of compiler output for beginners.

So our goal is to help students minimize type errors.

**Methods**
After looking at type errors, in order to see how the student’s code changes over time, we tracked the progress of each function, and labeled them as bad/fix pairs:

We proceeded to see if the actual fixed location matches the error location that compiler outputs and found that the error messages are often misleading.

In order to improve error messages, we decided to use OCaml’s type annotation.

From top-level, typing an OCaml function generates its type annotation. It prints out what it thinks are the types of declared function. The general format is:

```
fun : arg1 -> arg2 -> ... -> argn -> return_type
```

For example,

```
# let average a b = (a +. b) /. 2.0;

val average : float -> float -> float = <fun>
```

And we used this information to annotate each function and its fixed version, to see if this would lead to OCaml compiler outputting more accurate error location:

**Results**
We randomly picked 100 samples from out dataset to check whether the annotated code produces better error message or not.

### Improvement:
- Generic types replaced by specific types when possible and appropriate
- Eliminates redundant information about recursion
- Compiler can actually catch the type error in the base case of a recursion

### Limitation:
- Unable to improve the type error messages caused by misusing generic library functions, such as List.fold. Since we cannot add specific type annotations for those library functions, the compiler will always point to them regardless of where the actual type error is.

**References**
Tutorials & FAQ Retrieved from ocaml.org

**Acknowledgements**
Thanks to the Early Research Scholars Program, UCSD CSE department, advisors Professor Christine Alvarado, Professor Ranjit Jhala, mentors Michael Barrow, Shelby Thomas, and PhD students Eric Seidel, Ben Cosman and Panagiotis Vekris. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1339335.