Well-typed program don't go wrong? — An Analysis of a common myth

YUFENG PENG, McGill University, Canada

This study aims to investigate the correlation between fixing type errors and grade progression in an introductory functional programming course. The importance of understanding type errors and modern type system concepts for students learning functional programming has been widely acknowledged. However, students often struggle with comprehending and accurately identifying these concepts. The study analyzes the submissions of students in an introductory functional programming course and examines the relationship between fixing type errors and grade progression. Through this analysis, the study provides insights into the effectiveness of the current teaching methods, students' reliance on graders, and novice learners' approach to understanding the type system. The findings of this study contribute to ongoing efforts to improve the teaching and learning of functional programming.

Additional Key Words and Phrases: functional programming, type errors, compile errors

ACM Reference Format:
Yufeng Peng. 2023. Well-typed program don't go wrong? — An Analysis of a common myth. 1, 1 (April 2023), 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern-type system concepts are fundamental for students to learn and understand when first introduced to functional programming, but they can be challenging to comprehend and accurately identify. This issue is particularly pronounced in introductory functional programming courses, where students — often from a background in imperative languages nowadays — struggle to understand concepts related to type checking [Chakravarty and Keller 2004] and interpret the type errors.

The problem has been well-known for years and has recently garnered increased attention. Many reports have examined how novice learners interact with compile-time errors[Munson and Schilling 2016], and numerous studies and experiments have focused on generating higher-quality type error messages that provide more targeted, localized[Sharrad et al. 2018] and understandable feedback[Heeren 2005].

The growing interest in type errors is due in part to the common assumption that well-typed programs generally tend to be more accurate. We aim to demystify this assumption by analyzing the submissions of students in an introductory functional programming course. We also generalize the topic and discuss how students interact with broader compile-time errors and attempt to interpret students' behavior and how they learn through debugging.

In this report, we identify in detail our hypothesis (Section 2), explain the methodology and terminologies, including the characteristics of the data, compile-time errors to focus on, and our approach to split homework submissions by questions for more focused analysis (Section 3), discuss the correlation between fixing errors, in particular type errors, and grade progression. From there, we can further discuss how much students rely on graders and how novice programmers approach learning functional programming and understanding the type system, and if unlimited automatic

Author's address: Yufeng Peng, yufeng.peng@mail.mcgill.ca, McGill University, Canada.

Manuscript submitted to ACM

graders positively assist the overall learning of students [Mitra 2023]. By doing so, we hope to contribute to the ongoing efforts to improve the teaching and learning of functional programming.

2 METHODOLOGY

This section presents an overview of the methodology used in our study. It provides a detailed description of the dataset, Learn-OCaml terminologies, and the error categories that were examined. We also outline the approach used to process and split the data to achieve targeted analysis.

2.1 Data Overview and Terminologies

The dataset for this study was obtained from an introductory functional programming course at McGill University. The course is primarily taught in OCaml, a strongly-typed functional programming language, and is typically taken by second or third-year undergraduate students majoring in Computer Science or related fields. The course covers a range of topics, including binding and scoping, data abstraction, type checking, and functional and logic programming.

To facilitate automated grading and assign exercises, the course utilizes Learn-Ocaml [Canou et al. 2017], an online programming platform. Learn-Ocaml enables students to code, compile, and execute OCaml programs and provides an unlimited grader for them to check the accuracy of their programs. The submission event data of 106 students who consented to have their data used were collected during the Fall 2020 Semester. The course runs a modified version of the platform that stores all submission event data, regardless of success or failure, and sanitizes it [Ceci et al. 2021].

Learn-OCaml supports 3 types of submission events: `compile`, `eval`, and `grade`. Here's an overview of the number of submissions per homework per event:

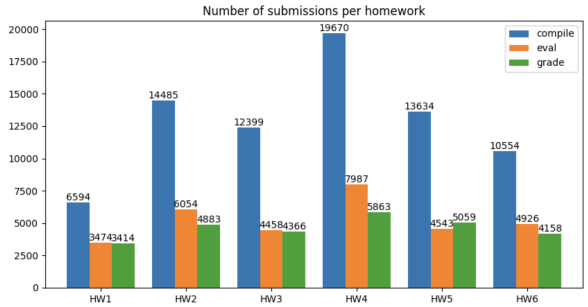

Fig. 1. Distribution of Student Submissions

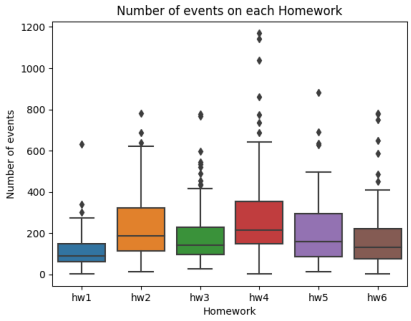

Fig. 2. Number of Grade command submissions

The dataset used in this study is of significant scale, as illustrated in the figures. In order to focus our analysis, in this report, we have selected Homework 1 and Homework 5 for our examination. This decision was made as these particular assignments provide valuable insights into the types of errors students are prone to make at different stages in their learning journey. Specifically, the first homework allows us to identify common errors made by students when they are first introduced to functional programming concepts. On the other hand, Homework 5, being one of the last homework assignments, allows us to explore the types of errors that are still problematic for students even after they have gained some experience and familiarity with the language. In addition, Homework 5 is on topics of control flow

and backtracking, and dependency exists among functions. This gives us a higher level analysis of more complex student submissions.

By analyzing these two assignments, we aim to gain a deeper understanding of the challenges that students face when learning OCaml and the nature of errors they encounter throughout their learning journey.

2.2 Learn-OCaml Error Categories

Errors are collected when any of the submission events cause a compile-time error. To effectively analyze the dataset, we interpret the error message, identify keywords in the messages, and categorize the errors.

For example, for type errors, Learn-OCaml generates a compile-error message as follows:

```
File ""compile_test.ml"", line 27, characters 13-19:
27 |   | _ -> n * fact n - 1
                 ^^^^^^
Error: This expression has type float but an expression was expected of type
         int
```

The messages for type mismatch are generated with the structure of `This expression has type ... but an expression was expected of type ...`. By identifying those keywords, we can assign this error to be of category "Type error".

Below shows the error category distribution for Homework 1 and Homework 5. We observe that the number of Type errors and Syntax errors is largely dominant.

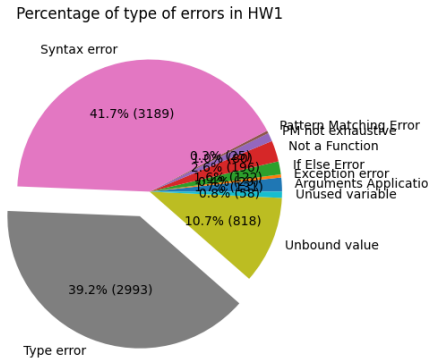

Fig. 3

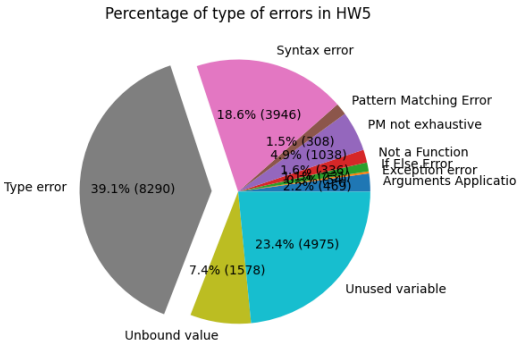

Fig. 4

2.3 Data Processing

Our data processing has two parts: homework split and grade and error collection.

2.3.1 Homework Split. Prior research in the field of functional programming education has often focused on homework-based analysis, such as average grades, time spent, and grade progression for a certain homework. In this study, we aim

to investigate the relationship between individual questions and student performance by performing a more targeted analysis of the data.

To achieve this, we utilize the OCaml library `compiler-lib` to generate the Abstract Syntax Trees (AST) of the submitted code and extract functions from it [ocaml 2023]. This process allows us to examine the code at a more granular level, giving us greater insight into how individual questions were approached by students. We also consider dependent functions, which are functions that are called by other functions, in our analysis.

2.3.2 Grade and Error Collection. We also perform an analysis of the grading and error collection data of the question-split dataset. Our analysis aims to gain insights into the compile-time errors committed by students while solving the programming exercises.

To perform this analysis, we use the `learn-ocaml` command to grade the split-by-question student submissions. We analyze the progression of the students' grades over time, examining whether the grades improve or deteriorate as they continue to work on the homework.

Furthermore, we log compile errors during the grading process and analyze their distribution, along with the syntax errors intercepted in AST generation. We aim to identify patterns and trends in the type errors committed by the students, which can help improve the error messages and instructional materials for future students. Our analysis can also provide insights into the common misconceptions and difficulties faced by students while learning functional programming with OCaml.

3 RESULT

In this section, we present the results of our analysis, focusing on both the grade progression and error distribution of the two selected homework assignments.

First, we observe that with the help of unlimited auto graders, students on average achieve very high final grades. As seen in Figure 5, the grade averages for Homework 1 to 6 are 99.5, 95.0, 98.0, 96.6, 93.5, and 91.0. Therefore, we pay closer attention to the process of how students reach high grades in the end.

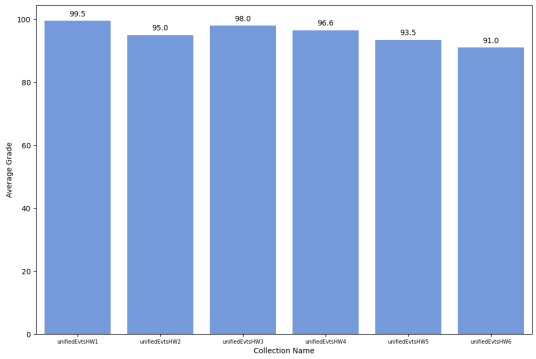

Fig. 5. Homework final grade averages

3.1 Grade Progression

We generated grade progressions for each question for every student in our dataset. To avoid an overpopulation of graphical results, we focus on one student who exemplifies typical student performance.

Following is the grade progression of Homework 5 and homework 5 for a student with anonymized ID '2e8b67ff6ec42ca2a35065901de07638':

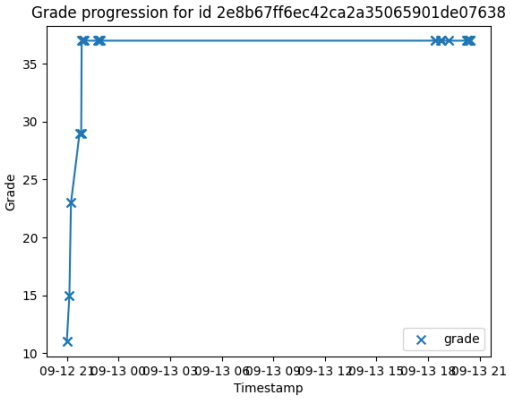

Fig. 6. Homework1 - Question1

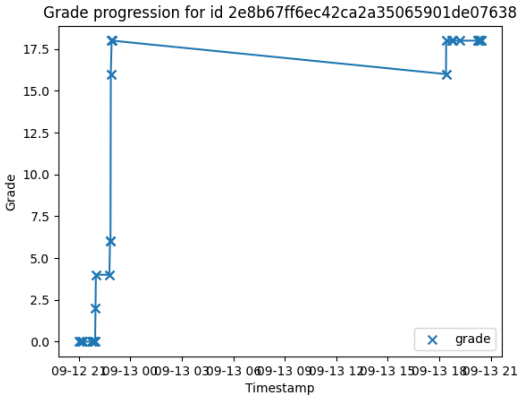

Fig. 7. Homework1 - Question2

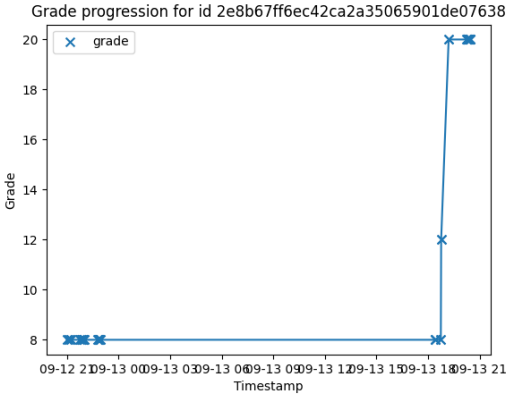

Fig. 8. Homework1 - Question1

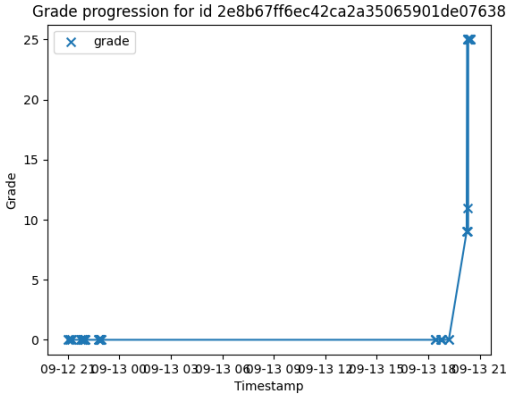

Fig. 9. Homework1 - Question2

A grade of 0 can be caused by either the failure to compile, in which case the grader automatically gives a zero, or the execution of a correctly compiled but logically incorrect program. It is clear to see that once the student leaves the "zero zone", the grade increases positively at a very fast rate. This pattern is consistent with our observations of other students in our dataset.

Similarly, in the case of the student's submission for Homework 5, we observe a similar pattern once the compile-time errors are fixed.

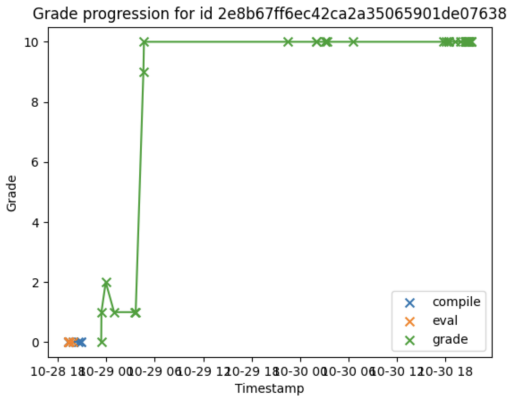

Fig. 10. Homework5 - Question1

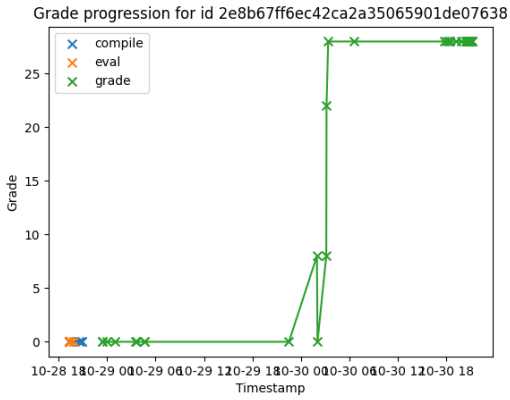

Fig. 11. Homework5 - Question2

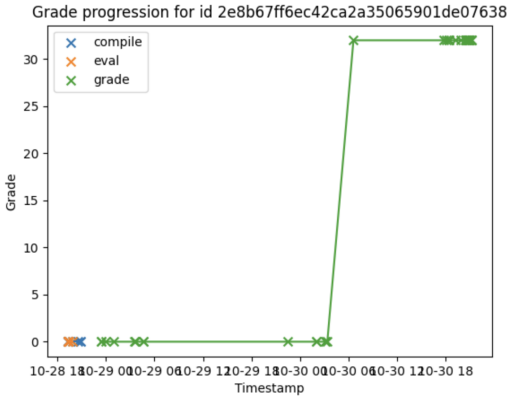

Fig. 12. Homework5 - Question3

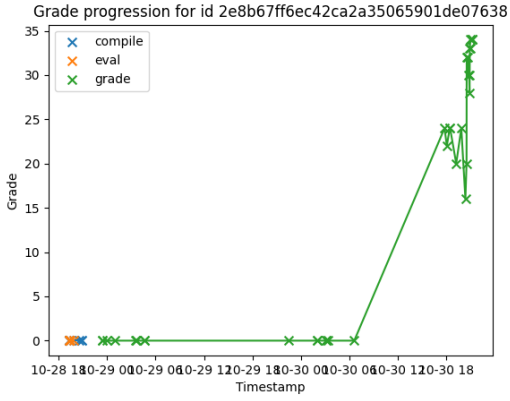

Fig. 13. Homework5 - Question4

We observe that once students fix their errors, their grades increase swiftly, indicating that the students have a good understanding of the course material.

3.2 Error Distribution

Surprisingly, the student in discussion did not have any Type errors for Homework 1. Looking into the dataset, we found that the students encountered Syntax errors numerous times at the beginning. Further investigation revealed that the problems in Homework 1 intentionally give students syntactically incorrect template codes, and part of the goal of the Homework is for the students to identify and fix the syntax errors. Therefore, this is an edge case that should be taken into consideration to achieve improved data analysis (as mentioned in Section 6, Future Work).

On the other hand, the student had a compacted Type error distribution at the very early stage of doing Homework 5. This pattern is consistent with our observation that most students in our dataset had a high concentration of type

errors at the beginning of doing homework. As students progressed through the homework, the number of type errors decreased, indicating, again, a better understanding of the course material.

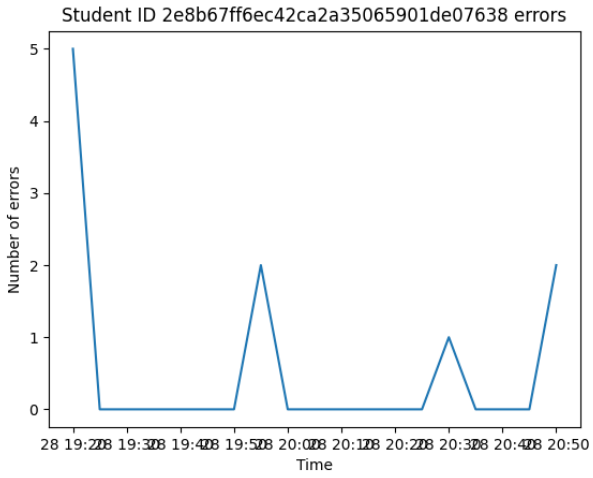

Fig. 14. Homework5 Type Error counts

3.3 Discussion

In this study, we investigated two hypotheses regarding the relationship between type errors, compile errors, and students' grades in functional programming assignments. Firstly, we hypothesized that once a program is well-typed, its accuracy progresses positively. Then we further expanded and generalized the topic, and we hypothesized that students gain a better understanding of the program through debugging compile errors, leading to a positive grade progression once the program successfully compiles.

Our results support both hypotheses. We found that the frequency of compile errors decreases rapidly once students start to gain non-zero grades, and once a program is well-typed, the frequency of recurring type errors is low. Additionally, we observed that once a program is compile-error-free, students' grades generally increase steadily.

Furthermore, we found that compile errors most frequently occur in the early batches of submissions, both in terms of time and order of submission. This suggests that early detection and correction of compile errors is crucial for students' success in functional programming assignments.

4 RELATED WORK

The field of computer science education has seen a significant amount of research in recent years. One of the main areas of research in this field is the analysis of student errors and misconceptions when learning programming concepts. Our research specifically explored common errors that students make when first exposed to functional programming concepts, and how they proceed to interact with compile-time errors to improve their understanding and performance.

Several studies have investigated the use of auto-graders and their effectiveness in programming education. For example, work by Joydeep analyzed the impact of immediate feedback from auto-graders on student performance

in introductory programming courses and found that the use of an auto-grader led to an improvement in student performance [Mitra 2023]. It has been argued that auto-graders do not discourage students from acquiring independent testing skills, which some may have thought. On the contrary, the feedback provided by these tools can assist students, especially those from underrepresented groups such as women, in improving their understanding and gaining confidence.

Other studies have examined specific error types that students commonly make in introductory functional programming courses. For instance, Tirronen et al. (2015) discovered that mistakes with types (CouldntMatch, NoInstance) is one of the leading error categories, along with difficulties with syntax, followed by run-time errors [TIRRONEN et al. 2015]. Our study complements this research by focusing specifically on errors related to functional programming concepts, such as type errors, and further exploration into other specific errors, such as pattern matching errors, is greatly encouraged (Section 5, Future Work).

Moreover, there has been a variety of research exploring generating higher quality error messages to better facilitate the learning experience. Specific to type errors, many researchers have developed useful type debugging tools and algorithms. Tsushima and Asai created an OCaml type debugger that is interactive and capable of pinpointing the source of type errors. It achieves this by asking a series of questions to determine if an expression has the intended type [Ishii and Asai 2014]. For more concise explanations of type errors, works on identifying multiple errors in the form of error slices with a novel type inference algorithm have been done [Kustanto and Kameyama 2010].

In summary, our study contributes to the existing literature on programming education by analyzing common errors in functional programming and exploring how students interact with compile-time errors to improve their understanding and performance. Our findings can inform the design of better error messages, instructional materials, and feedback mechanisms to support students' learning in functional programming courses.

5 FUTURE WORK

While our study provides valuable insights into the progression of student grades and type error distribution over the course of two specific homework assignments, there are still areas of potential improvement and further exploration that could be addressed in future work. Some of these areas include:

- More In-Depth Analysis: While our analysis of grade progression and error distribution provides a general overview of student performance, there are many more angles of data analysis that could be explored to gain a more comprehensive understanding of student learning. For example, we could investigate how different types of errors relate to specific programming concepts, explore how student performance varies based on factors such as prior programming experience or course load, or examine grade regression as complementary to grade progression.
- Cross-Homework Comparison: Our study focuses on two specific homework assignments, but it would be interesting to see how student performance varies across a broader range of assignments. By comparing student performance across multiple assignments, we could gain insights into how different programming concepts and problem-solving strategies build on each other over time.
- Detailed Analysis of Specific Error Types: While we observed trends in type error distribution over time, we did not perform a detailed analysis of how students learn from fixing specific types of errors. For example, we could investigate how students gain a better understanding of pattern matching by fixing "Pattern Match Not Exhaustive" errors.

- Continuous Work: Finally, it's important to note that our study represents just a snapshot of student learning over a specific period of time. Continuous work on this dataset could reveal even more insights into how students learn and progress over time, and could help inform the design of better error messages, instructional materials, and teaching strategies to support student learning in functional programming.

6 CONCLUSION

In conclusion, our study provides insights into the relationship between type errors, or more generally, compile errors, and students' grades in functional programming assignments. Our findings support the hypothesis that once a program is well-typed, its accuracy progresses positively. The progression also indicates that students gain a better understanding of the program through debugging compile errors. Additionally, our study highlights the importance of early detection and correction of compile errors, as well as early debugging of type errors, for students' success in functional programming assignments.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Pientka for her guidance and support throughout the semester, Professor Si for his pertinent technical assistance, Scarlett for her generous transfer of knowledge of the Learn-Ocaml platform, and Jerry for contributing features to the analysis scripts. Their contributions were essential in ensuring the accuracy and reliability of this study.

I would like to thank the LearnOcaml research group for the opportunity to work with them. It has been a valuable experience, and I appreciate their feedback and support. This research project has been an incredible learning opportunity, and I am grateful for the knowledge and skills gained during this time.

REFERENCES

Benjamin Canou, Roberto Di Cosmo, and Grégoire Henry. 2017. Scaling up Functional Programming Education: Under the Hood of the OCaml MOOC. *Proc. ACM Program. Lang.* 1, ICFP, Article 4 (aug 2017), 25 pages. https://doi.org/10.1145/3110248

Alana Ceci, Hanneli C. A. Tavante, Brigitte Pientka, and Xujie Si. 2021. Data Collection for the Learn-OCaml Programming Platform: Modelling How Students Develop Typed Functional Programs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 1341. https://doi.org/10.1145/3408877.3439579

Manuel M. T. Chakravarty and Gabriele Keller. 2004. The risks and benefits of teaching purely functional programming in first year. *Journal of Functional Programming* 14 (2004), 113 – 123.

Bastiaan Johannes Heeren. 2005. *Top Quality Type Error Messages*. s.n.

Yuki Ishii and Kenichi Asai. 2014. Report on a User Test and Extension of a Type Debugger for Novice Programmers. In *Proceedings 3rd International Workshop on Trends in Functional Programming in Education, TFPIE 2014, Soesterberg, The Netherlands, 25th May 2014 (EPTCS, Vol. 170)*, James L. Caldwell, Philip K. F. Hölzenspies, and Peter Achten (Eds.). 1–18. https://doi.org/10.4204/EPTCS.170.1

Cynthia Kustanto and Yukiyoshi Kameyama. 2010. Improving Error Messages in Type System. *IPSJ Online Transactions* 3 (2010), 125–138. https://doi.org/10.2197/ipsjtrans.3.125

Joydeep Mitra. 2023. Studying the Impact of Auto-Graders Giving Immediate Feedback in Programming Assignments *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 388–394. https://doi.org/10.1145/3545945.3569726

Jonathan P. Munson and Elizabeth A. Schilling. 2016. Analyzing Novice Programmers' Response to Compiler Error Messages. 31, 3 (jan 2016), 53–61.

ocaml. 2023. ocaml. https://github.com/ocaml/ocaml/blob/trunk/parsing/parsetree.mli.

Joanna Sharrad, Olaf Chitil, and Meng Wang. 2018. Delta Debugging Type Errors with a Blackbox Compiler. In *Proceedings of the 30th Symposium on Implementation and Application of Functional Languages* (Lowell, MA, USA) *(IFL 2018)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/3310232.3310243

VILLE TIRRONEN, SAMUEL UUSI-MÄKELÄ, and VILLE ISOMÖTTÖNEN. 2015. Understanding beginners' mistakes with Haskell. *Journal of Functional Programming* 25 (2015), e11. https://doi.org/10.1017/S0956796815000179