Towards Attention-Based Automatic Misconception Identification in Introductory Programming Courses

Conference Paper · March 2024

DOI: 10.1145/3626253.3635575

6 authors, including:

Muntasir Hoq
North Carolina State University
13 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Jessica Vandenberg
North Carolina State University
36 PUBLICATIONS **130** CITATIONS
SEE PROFILE

James Lester
North Carolina State University
112 PUBLICATIONS **707** CITATIONS
SEE PROFILE

Bita Akram
North Carolina State University
26 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Towards Attention-Based Automatic Misconception Identification in Introductory Programming Courses

Muntasir Hoq
North Carolina State University
United States
mhoq@ncsu.edu

Jessica Vandenberg
North Carolina State University
United States
jvanden2@ncsu.edu

Bradford Mott
North Carolina State University
United States
bwmott@ncsu.edu

James Lester
North Carolina State University
United States
lester@ncsu.edu

Narges Norouzi
University of California Berkeley
United States
norouzi@berkeley.edu

Bita Akram
North Carolina State University
United States
bakram@ncsu.edu

ABSTRACT

Identifying misconceptions in student programming solutions is an important step in evaluating their comprehension of fundamental programming concepts. While misconceptions are latent constructs that are hard to evaluate directly from student programs, logical errors can signal their existence in students' understanding. Tracing multiple occurrences of related logical bugs over different problems can provide strong evidence of students' misconceptions. This study presents preliminary results of utilizing an interpretable state-of-the-art Abstract Syntax Tree-based embedding neural network to identify logical mistakes in student code. In this study, we show a proof-of-concept of the errors identified in student programs by classifying correct versus incorrect programs. Our preliminary results show that our framework is able to automatically identify misconceptions without designing and applying a detailed rubric. This approach shows promise for improving the quality of instruction in introductory programming courses by providing educators with a powerful tool that offers personalized feedback while enabling accurate modeling of student misconceptions.

ACM Reference Format:
Muntasir Hoq, Jessica Vandenberg, Bradford Mott, James Lester, Narges Norouzi, and Bita Akram. 2024. Towards Attention-Based Automatic Misconception Identification in Introductory Programming Courses. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635575

1 INTRODUCTION

To effectively support students' learning of introductory programming, it is essential to identify and address their misconceptions of fundamental concepts in a timely manner [7]. However, the disproportionate growth in the number of Computer Science (CS) students compared to teaching staff makes it infeasible to monitor student programs to infer misconceptions in real-time [1]. Automated identification of student misconceptions could be an efficient way to provide individualized feedback either through teachers or automatically. However, automated identification of misconceptions is challenging as misconceptions are primarily latent constructs that cannot be assessed directly. Nevertheless, misconceptions lead to logical errors in student code [2]. Tracing patterns of student errors over time can lead to effectively identifying misconceptions in student programs [4].

Prior studies have attempted semi-automated misconception discovery by highlighting patterns using association rules [3, 5]. However, these studies only focused on simple association rules that fall short in analyzing more complex tasks, i.e., programming [8]. One recent study [8] utilized deep learning models to identify misconceptions automatically. However, their approach required manual labeling of data using an expert-designed rubric. Moreover, their methodology detects misconceptions on a per-rubric-item basis within the student code. This approach does not effectively address misconceptions that may span across multiple rubric items or in cases where rubric items are missing. In this study, we take an initial step towards the automated identification of student misconceptions by showing preliminary results from a deep learning model that can automatically identify multiple logical errors in students' code. In our approach, we build a classifier to group student programs into correct vs. incorrect categories. The dataset is labeled automatically by running programs through a set of test cases. Our framework uses a modified version of a state-of-the-art interpretable deep learning model, Subtree-based Attention Neural Network (SANN) [6]. We use the attention layer of SANN to highlight program snippets in student code that are most influential in classifying code as correct vs. incorrect. A preliminary evaluation of this approach suggests that these patterns are closely associated with logical errors in student code. We demonstrate the outcomes of our framework in this study.

2 DATASET AND METHODOLOGY

We use a publicly-available dataset[1] collected from the CodeWorkout platform. The dataset contains student programming solutions in Java for 50 programming problems from a CS1 course. These submissions were scored in a range of [0,1] based on the number of

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635575

[1] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=3458

test cases passed. To train our model, we categorized the data into correct or incorrect for a binary classification. In this experiment, we work with one problem named *"caughtSpeeding"* containing 1,574 student programs (correct: 617, incorrect: 957). Uncompilable solutions were excluded from the dataset due to their inability to generate Abstract Syntax Trees (ASTs) and chose not to consider syntactical errors in this study. Using SANN, we map student programs' AST representations to their corresponding condensed vector representation. SANN leverages an attention mechanism that enables us to understand which program subtrees are responsible for the model output, providing interpretability power. While the original SANN model used a genetic search optimization process to break down the AST into non-overlapping subtrees of fixed sizes given a specific task, we modified the original SANN model to incorporate all possible subtrees with various sizes. This decision was made because subtrees of various sizes can represent logical errors, and thus, a fixed-size subtree might not comprehensively encapsulate a single error. Furthermore, logical errors can have a hierarchical nature, where a larger subtree representing a logical error can include another subtree representing a smaller error.

3 RESULTS AND DISCUSSION

In this study, we train the modified SANN model on student program correctness (incorrect: 0, correct: 1). We divide the dataset into train, validation, and test sets (60%, 20%, 20%). We set the embedding size to 64 and kept the other parameters with their default values. The accuracy of the model on the test set is 91%.

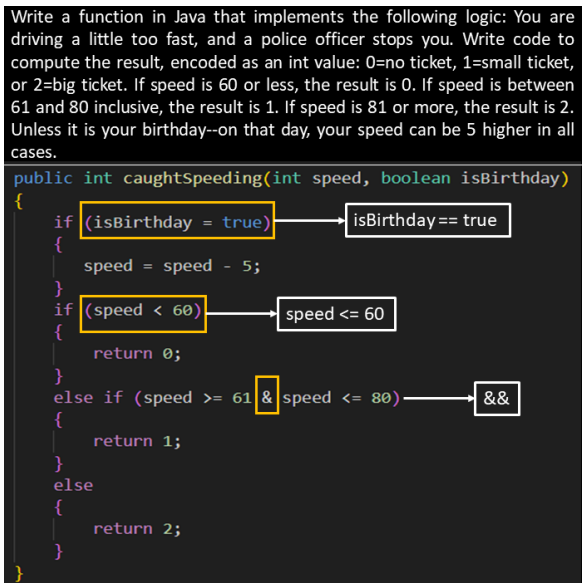

Figure 1: Incorrect student solution for caughtSpeeding.

Using the trained model, we extract attention weights for each possible subtree to highlight the most influential subtrees for the predictions made by the model. We hypothesized that subtrees representing logical errors should receive high weights by the attention mechanism. We did a proof of concept to investigate our hypotheses further. Based on our observations, the attention mechanism correctly identifies erroneous parts of incorrect programs. Figure 1 illustrates one instance of an incorrect student solution with three errors to the *caughtSpeeding* problem. The errors are marked in orange, and their corrected versions are marked in white in the figure. As shown, the model successfully identifies all of the errors in the code. This study is a stepping stone to a generalized programming misconception identifier for CS1. This automated approach has the potential to help enhance instruction quality by enabling educators to pinpoint and address individual student misconceptions, thus promoting a more effective and tailored learning experience.

4 LIMITATIONS AND FUTURE DIRECTIONS

The results of this preliminary study show promise for utilizing our model to highlight logical errors in students' solutions. This can serve as a first step towards automated identification of students' misconceptions and highlight key areas of difficulty when their progress is tracked over time. Utilizing the results of this model, we plan to employ clustering techniques to group similar logical errors to facilitate feedback propagation for instructors. Furthermore, an aggregated report on students' common logical errors and misconceptions could assist educators in identifying and assessing the most challenging concepts, prompting a reevaluation of the corresponding course materials. In the future, we plan to formally evaluate the effectiveness of our model by comparing its results against a set of ground truth logical errors identified by experts for each problem. Our results show our model's effectiveness in identifying logical errors from solutions to a single problem. We plan to evaluate the generalizability of our model by applying it to a dataset that contains solutions to multiple problems. Furthermore, our model can identify the absence of an important programming construct (i.e., a student missing a part of the solution) by highlighting the whole Abstract Syntax Tree for that part. We plan to incorporate information from correct solutions to identify missing constructs in incorrect solutions as well.

5 ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (NSF) under Grants DUE-2236195 and DUE-2331965.

REFERENCES

[1] Bita Akram, Wookhe Min, Eric Wiebe, Anam Navied, Bradford Mott, Kristy Elizabeth Boyer, and James Lester. 2020. Automated assessment of computer science competencies from student programs with gaussian process regression. In *EDM'20*.
[2] Ella Albrecht and Jens Grabowski. 2020. Sometimes It's Just Sloppiness-Studying Students' Programming Errors and Misconceptions. In *SIGCSE'20*. 340–345.
[3] Myse Elmadani, Moffat Mathews, and Antonija Mitrovic. 2012. Data-driven misconception discovery in constraint-based intelligent tutoring systems. (2012).
[4] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common logic errors made by novice programmers. In *ACE'18*. 83–89.
[5] Eduardo Guzmán, Ricardo Conejo, and Jaime Gálvez. 2010. A data-driven technique for misconception elicitation. In *UMAP'10*. 243–254.
[6] Muntasir Hoq, Sushanth Reddy Chilla, Melika Ahmadi Ranjbar, Peter Brusilovsky, and Bita Akram. 2023. SANN: Programming Code Representation Using Attention Neural Network with Optimized Subtree Extraction. In *CIKM'23*. 783–792.
[7] Samiha Marwan, Bita Akram, Tiffany Barnes, and Thomas W Price. 2022. Adaptive immediate feedback for block-based programming: Design and evaluation. *IEEE Transactions on Learning Technologies* 15, 3 (2022), 406–420.
[8] Yang Shi, Krupal Shah, Wengran Wang, Samiha Marwan, Poorvaja Penmetsa, and Thomas Price. 2021. Toward semi-automatic misconception discovery using code embeddings. In *LAK'21*. 606–612.