Making it Smart: Converting Static Code into an Interactive Trace Table

Zak Risha
University of Pittsburgh
Pittsburgh, USA
zak.risha@pitt.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, USA
peterb@pitt.edu

ABSTRACT

This paper introduces a new type of smart learning content, an automatically generated trace table, that can easily integrate and adapt to existing curriculum and learning systems for computer science education. In addition to current features of the software, we describe how this tool constructs trace tables using only source code as an input. The potential of this tool is also explored by examining future opportunities in adaptation, feedback, and learning specifications. Last, we report a pilot integration into an existing system to demonstrate interoperability with a tangible use case.

Author Keywords

smart learning content; educational tools; teaching with technology; intelligent tutoring systems; technology integration; dissemination; technology adoption; educational research; computer science education

CCS Concepts

•Social and professional topics → Computing education; •Applied computing → Interactive learning environments;

INTRODUCTION

Computer science educators have long been tasked with developing content and curriculum for their courses and learning experiences, playing a fundamental role in their success. These materials often are created in static forms, such as textbooks, HTML documents, or simple learning systems. While advances in learning technologies have produced smart learning content (SLC) which incorporates interactivity, data collection, and adaptation [1], educators may be reluctant to abandon tried and tested curriculum they have spent precious time developing and refining. Similarly, older systems that utilize static content or have low levels of interactivity might still be valued for their infrastructure or past collected data.

Such issues create barriers to adopting newer SLC, potentially resulting in these new technologies being underutilized. Instances such as these call for content-agnostic tools that can automatically adapt to existing curriculum and materials, rather than forcing educators to use new content or spend time recreating their own. These differ from content-agnostic platforms such as MOOCs and learning management systems mainly concerned with distribution of learning materials [7]. Rather these dedicated tools function as SLC that enhance learning with engaging and interactive material while remaining agnostic to material within that particularly domain. Such tools are also embeddable into many different environments to easily integrate into existing content and methods of delivery.

CODE-AGNOSTIC TOOLS

In the domain of computer science, content-agnostic tools can be conceptualized as *code agnostic*, which we define as producing a learning activity using only source code as an input. While not a panacea for all code, such approaches scale to a wide variety of programs. Prior code-agnostic tools emerged as program visualizations like Python Tutor which used execution traces for line-based visualization of a program's execution [4]. Similarly, Jsvee utilized a transpiler to automatically generate expression-level visualizations from a piece of code [8]. While both excellent examples of code-agnostic tools, they fall into the domain of worked examples which are not well suited for assessing student knowledge or actively engaging them in applying such knowledge. Some programming construction environments like PCRS [10] and Turing Craft's Code Lab (http://turingscraft. com/) are partially code-agnostic but require the writing of test-cases to automatically assess the program which can be time consuming to create. Our proposed approach is to further build off fully automated methods by creating a scaffolded assessment which can collect fine grain data on student's knowledge.

TRACE TABLE

A trace table is a technique used to track variable changes throughout a program's execution by creating columns of variables with their respected values. This technique is applicable to many different type of programs and has long been used via pen and paper. Prior work has identified program tracing as an applied strategy [3] and studied the application of a paper based approach [2, 9] on performance. Our proposed tool creates automatically generated SLC from source code, producing a trace table exercise with interactivity, feedback, and data collection. This is accomplished by utilizing variable change information produced from execution traces to create

Sixth SPLICE Workshop at L@S 2020 "Building an Infrastructure for Computer Science Education Research and Practice at Scale", SPLICE'20, August 12, 2020, Virtual Event

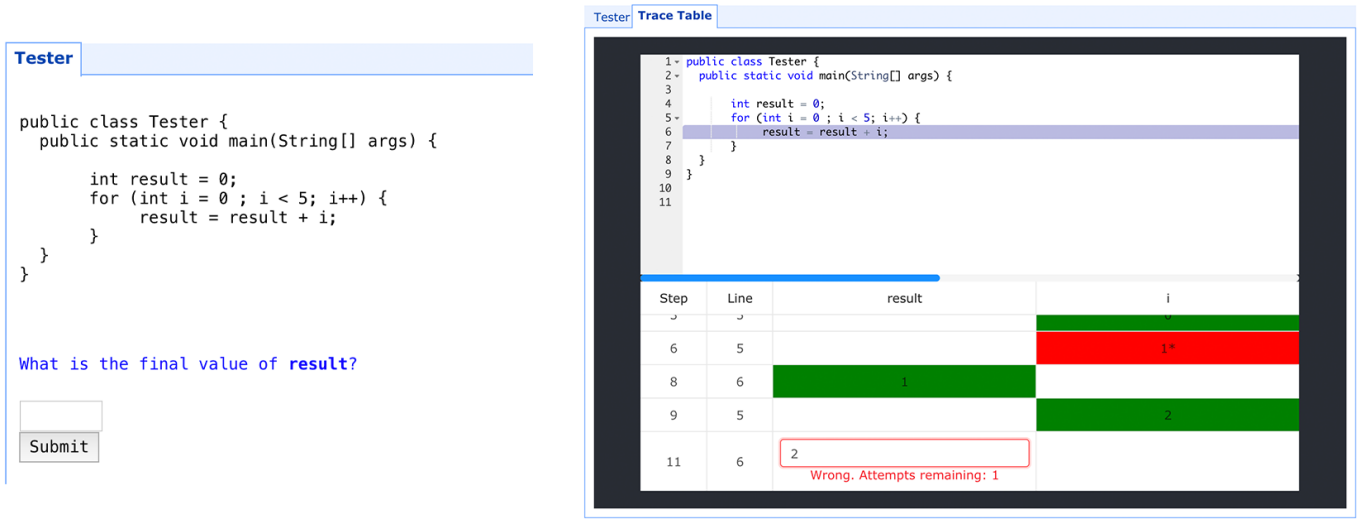

Figure 1. A Quizjet Problem (left) and a Trace Table (right) generated from the same program

an interactive table that prompts users to input the values of variables while stepping through the program.

The trace table interface, as shown in Figure 1, contains the code to be traced, progress bar, and a dynamically generated table. The current line being evaluated is highlighted and an input field appears in the correspond cell of the variable to be updated. The step and line number act as an index for the rows of the table as students gradually fill out the cells to trace the execution of code. Feedback is currently limited to correct or incorrect, first providing a number of wrong attempts until marking the cell incorrect and providing the answer. The software utilizes execution traces (produced by debuggers) from the backend of Python Tutor [4] to build the table using a single-page application framework. This allows for the dynamic construction of the trace table, adding rows as students progress in an effort to reduce cognitive load.

Similarly, the web based technology is embedable into other platforms, tutors, and web environments. Inclusion of a JavaScript file provides an API to load code, configure settings, and access student data. Reporting of interactions is achieved through a state management framework which can utilizes middleware to flexibly track user interface changes. Knowledge components (KCs) are automatically associated with each line by utilizing the abstract syntax tree when parsing the code [5]. This allows for a more fine-grain evaluation of student knowledge, identifying the specific line and associated concepts where a user might struggle. The end result is a code-agnostic tool which can augment existing systems with scaffolding, assessment, and student modeling.

PILOT INTEGRATION

To evaluate the utility of our trace table software, we piloted integration into an existing system that utilized static code. Quizjet [6] is a platform that provides parameterized questions and automated assessment for predicting the end result variable or system output of a piece of code. This system has been used for over a decade to author problems and assess student knowledge, generating more than 45,000 transactions from over 600 students. Quizjet's means of assessment is holistic, rather than fine-grained, resulting in less accurate assessment of student errors. In addition, the approach is less interactive than the trace table which engages students through a step-by-step analysis of the code.

In this manner, the trace table was embedded into the system (shown in Figure 1) to augment Quizjet's problems by offering additional scaffolding to students for their prediction tasks. Even though Quizjet is over 10 years old, integrating the trace table took little effort and was able to cover 66 existing Quizjet problems. Upon integration, we began using the system in classroom studies and also recitations. The trace table has now been used by 115 students and over 500 tables have been completed. This has generated a substantial amount of fine-grained data for an optional activity. In less than a year, over 7,500 transactions have been collected.

SUMMARY AND FUTURE WORK

In future work, we will report the impact the trace table had on student performance and learning. In addition, this tool has many opportunities for optimizing efficiency while remaining code-agnostic. Since the domain modeling is automated, we can implement various adaptations based on student knowledge to reduce the time of the task and keep students engaged. Similarly, automatically generated feedback could help clarify errors for students. Incorporating other forms of interaction, such as self explanations, might also yield deeper levels of comprehension. Such features will further augment the trace table to align more with those found in intelligent tutoring systems while requiring little effort to adapt to existing content. These improvements, and the incorporation of standards and protocols such as xAPI and LTI, would further make this an ideal tool for collaborating and sharing with the community.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1822752.

REFERENCES

[1] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, Jaime Urquiza, Arto Vihavainen, and Michael Wollowski. 2014. Increasing Adoption of Smart Learning Content for Computer Science Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR '14)*. Association for Computing Machinery, Uppsala, Sweden, 31–57. DOI:http://dx.doi.org/10.1145/2713609.2713611

[2] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. Association for Computing Machinery, Tacoma, Washington, USA, 164–172. DOI: http://dx.doi.org/10.1145/3105726.3106190

[3] Sue Fitzgerald, Beth Simon, and Lynda Thomas. 2005. Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the first international workshop on Computing education research (ICER '05)*. Association for Computing Machinery, Seattle, WA, USA, 69–80. DOI: http://dx.doi.org/10.1145/1089786.1089793

[4] Philip J. Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, Denver, Colorado, USA, 579–584. DOI: http://dx.doi.org/10.1145/2445196.2445368

[5] R. Hosseini and P. Brusilovsky. 2013. JavaParser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings*, Vol. 1009. University of Pittsburgh, 60–63. http://d.scholarship.pitt.edu/26270/

[6] I.-Han Hsiao, Peter Brusilovsky, and Sergey Sosnovsky. 2008. Web-based Parameterized Questions for Object-Oriented Programming. Association for the Advancement of Computing in Education (AACE), 3728–3735. https://www.learntechlib.org/primary/p/30206/

[7] Ido Roll, Daniel M. Russell, and Dragan Gašević. 2018. Learning at Scale. *International Journal of Artificial Intelligence in Education* 28, 4 (Dec. 2018), 471–477. DOI:http://dx.doi.org/10.1007/s40593-018-0170-7

[8] Teemu Sirkiä. 2018. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process* 30, 2 (2018), e1924. DOI: http://dx.doi.org/10.1002/smr.1924 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1924.

[9] Benjamin Xie, Greg L. Nelson, and Andrew J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, Baltimore, Maryland, USA, 344–349. DOI: http://dx.doi.org/10.1145/3159450.3159527

[10] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. 2013. Facilitating code-writing in PI classes. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, Denver, Colorado, USA, 585–590. DOI: http://dx.doi.org/10.1145/2445196.2445369