Exploring Student Misconceptions about Concurrency Using the Domain-Specific Programing Language Sonic Pi

Giorgio Delzanno
DIBRIS, Univerità di Genova
Italy
giorgio.delzanno@dibris.unige.it

Giovanna Guerrini
DIBRIS, Univerità di Genova
Italy
giovanna.guerrini@dibris.unige.it

Daniele Traversaro
DIBRIS, Univerità di Genova
Italy
daniele.traversaro@dibris.unige.it

ABSTRACT

As the importance of concurrent and multi-threaded programming continues to grow, many universities have incorporated these concepts into their introductory courses. Sonic Pi, a programming language designed for music creation, provides valuable support for exploring concurrency due to its simplified multi-threading abstractions and its domain-specific nature. This paper investigates the combined use of Sonic Pi and Team-Based Learning to mitigate the difficulties in early exposure to concurrency. Sonic Pi provides great support for "playing" with concurrency, and "hearing" common problems such as data races and lack of synchronization among different threads. Our primary research goal is to explore whether the use of Sonic Pi can support students, especially in the early stages, to understand concurrent programming concepts and help them face misconceptions identified in the concurrency education literature. The approach has been applied in teaching experiments with undergraduate students involving 184 participants.

KEYWORDS

Concurrent Programming, Misconceptions, Music and Coding

ACM Reference Format:
Giorgio Delzanno, Giovanna Guerrini, and Daniele Traversaro. 2024. Exploring Student Misconceptions about Concurrency Using the Domain-Specific Programing Language Sonic Pi. 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/10.1145/3626253.3635521

1 MOTIVATION AND CONTRIBUTION

Learning to program is a challenging and demanding process because of its close connection to logical reasoning, problem-solving, and creative thinking. One computational concept that introduces further complexity is concurrency, i.e., the execution of multiple tasks at once by means of multiple execution threads working on a shared memory [3]. Concurrency introduces data races and non determinism: concurrent updates could interfere with each other, and the exact order in which execution threads are scheduled may not be determined a priori.

In traditional computer science curricula, concurrent programming was typically introduced at advanced levels. The relevance of concurrency is growing, for instance, multicore computing is eroding Moore's law, and thus several universities are now introducing concurrency in the first programming courses [2, 5]. However, students often struggle to understand concurrency, which can be particularly challenging when introduced early in the curriculum. Consequently, there is a need for research aimed at supporting students in building and refining their viable mental models. In this setting, domain-specific programming languages can play an essential role in reformulating complex concepts in a more natural way.

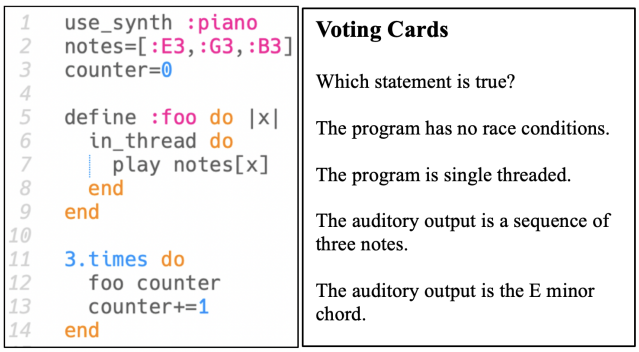
In this paper, we developed a pedagogical experiment related to introducing concurrency and multi-threading at different stages of the CS degree program. Our approach is based on the use of Sonic Pi [1], originally designed for live music coding performances. The Sonic Pi programming language provides abstractions to introduce basic computational concepts as well as advanced concepts such as multi-threading. Sonic Pi offers distinct advantages over traditional computing methods to facilitate the early introduction of multi-threading. The inherent concurrency of music creation, e.g., an orchestra, and the simplicity of the Sonic Pi editor provide a natural and immersive learning experience for concurrency. In particular, live coding and auditory output, two key features of Sonic Pi, provide great support for "playing" with concurrency and "hearing" data races and other common mistakes.

Based on previous research on concurrency education [5], our specific research questions are: Does the use of Sonic Pi help the students face common misconceptions and mistakes in concurrent programming? Is this more helpful for early-stage programmers? Can we apply knowledge transfer from Sonic Pi to more traditional programming languages such as C and C++? What is the effect in terms of learning outcomes?

To answer these questions, we have designed an introductory approach to concurrency for undergraduate students based on Sonic Pi and the collaborative teaching methodology of Team Based Learning (TBL) [4]. The proposed approach is the result of extensive research and classroom experimentation over the past three years. In particular, the approach has been applied to two teaching experiments for undergraduate CS students in the academic year 2022/23, more precisely, 130 first-year students of the computer architectures course (CA) and 54 third-year students of the concurrent and distributed algorithms course (CDA). By collecting and analyzing team responses and tasks, we delved into the students' misconceptions about concurrency. To our knowledge, this study represents the first attempt to use Sonic Pi for these learning goals at the university level.

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

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635521



```
use_synth :piano
notes=[:E3,:G3,:B3]
counter=0

define :foo do |x|
  in_thread do
    play notes[x]
  end
end

3.times do
  foo counter
  counter+=1
end
```

Voting Cards

Which statement is true?

The program has no race conditions.

The program is single threaded.

The auditory output is a sequence of three notes.

The auditory output is the E minor chord.

Figure 1: Example of Team App task.

2 DETAILS ABOUT THE APPROACH

In the context of the CA and CDA courses, we introduced students to multi-threading through a series of short practical tasks, each designed to address specific misconceptions about concurrency, allowing students to gradually build their understanding while addressing concepts such as thread local variables, race conditions, correctness and termination, and inter-thread synchronization. Some tasks focused on program comprehension, while others involved code writing. The first experiment was conducted just before students in the CA course were introduced to multi-threading in C. This timing allowed students to gain a basic understanding of concurrency through the Sonic Pi-based approach. The second experiment took place at the beginning of the CDA course, where students were introduced to concurrency concepts more in depth. The activities did not contribute to the student's final summative assessment, they only had formative value.

During the activities, we collected RAT quiz scores and team App tasks to analyze students' understanding of concurrency and any misconceptions that might arise. We also collected data through an anonymous individual post-questionnaire on students' appreciation/perception of the activity. Finally, we collected the results of a post-individual test to investigate knowledge transfer. The test consisted of three exercises formulated either in pseudo-code or in the C programming language. These exercises proposed a similar concurrency scenario, but in other languages, and provided practical opportunities for students to apply their understanding of concurrency in a different learning context.

Familiarisation with the TBL methodology and the Sonic Pi language is achieved by means of tutorials assigned to the students one week before the in-class activity. About 2-3 hours of individual study are estimated for the preparation before the class sessions. The RAT quiz consists of five multiple-choice questions that assessed the participants' understanding of the Sonic Pi language syntax and concurrency concepts. Following the TBL methodology, the quiz was completed first individually and then in teams.

The team App tasks included a variety of question formats, including multiple-choice questions (voting cards) and open-ended tasks that required students to implement Sonic Pi code. Each exercise focused on specific misconceptions related to concurrent programming. We focused on a range of concurrent concepts, such as data races and synchronization mechanisms, as well as fundamental programming concepts like variable scope and function calls.

An example of task that shows how to use Sonic Pi to "hear" misconceptions is shown in Fig. 1. The global array variable `notes` contains three notes represented by their code (the E note in the 3rd octave, and so on). Function `foo`, with parameter `x`, spawns a thread that plays the `x`th note in the array. The function is called three times in a loop. The `note` array is shared among the threads, however each thread has access to a different cell. When executed, the three notes are played in some casual order by the corresponding threads. In Sonic Pi, the execution of a sequence of `play` instructions is perceived as a single chord. Therefore, the auditory output of the program corresponds to the E minor chord.

3 DATA ANALYSIS AND CONCLUSIONS

To evaluate the effectiveness in terms of learning outcomes, we considered TBL data, in particular team responses to the TBL tasks. Additionally, we gather preliminary empirical evidence on the transferability of the concurrent programming knowledge, skills, and abilities that students acquire in our Sonic Pi-based learning environment to other traditional programming languages, such as C. Knowledge transfer can pose challenges such as differences in the underlying notional machines of the considered programming languages.

To investigate knowledge transfer, we designed a written test consisting of three exercises in either C or pseudo-code, focusing on concurrency aspects covered in the music code-based domain. The test was proposed one week after the experiment. Furthermore, to assess the effectiveness of the activity in terms of engagement and appreciation we considered students' overall perceptions through a post-questionnaire. The misconceptions emerged from the TBL collected data correspond to those identified in the literature, supporting the adequacy of Sonic Pi in this application.

The proposed activities were recognized useful by the students: the median value concerning the evaluation of the usefulness of the proposed activities was 4 on the Likert scale from 1 to 5. In particular, participants found the activity of great help in detecting and learning how to avoid the misconceptions presented in the exercises. Around 80% of the students reported a fair level of difficulty. Exercises using C syntax required more effort to adapt a mental model from one programming language to another with a different notional machine. Further research is however needed to explore knowledge transfer from Sonic Pi to other concurrent languages.

REFERENCES

[1] Samuel Aaron, Alan F. Blackwell, and Pamela Burnard. 2016. The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music Technology and Education* 9, 1 (3 2016), 75–94. https://doi.org/10.17863/CAM.8369

[2] Daniel Ernst and Daniel Stevenson. 2008. Concurrent CS: preparing students for a multicore world. *ACM SIGCSE Bulletin* 40 (06 2008), 230–234. https://doi.org/10.1145/1384271.1384333

[3] Maurice Herlihy. 1993. A Methodology for Implementing Highly Concurrent Data Objects. *ACM Trans. Program. Lang. Syst.* 15, 5 (November 1993), 745–770. https://doi.org/10.1145/161468.161469

[4] Larry K. Michaelsen, Warren E. Watson, John P. Cragin, and L. Dee Fink. 1982. Team-based learning: A potential solution to the problems of large classes. *Exchange: The Organizational Behavior Teaching Journal* 7, 4 (1982), 18–33.

[5] Filip Strömbäck. 2023. *Teaching and Learning Concurrent Programming in the Shared Memory Model.* Ph. D. Dissertation. Linköping University, Sweden. https://doi.org/10.3384/9789180750011