Adding Context to Theoretical Computation: A Proposed New Unit to UVA's Theory of Computation Course

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Megan Bishop

date submitted in May 10, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Megan Bishop

Technical advisors: Anil Vullikanti and Yonghwi Kwon, Department of Computer Science

Adding Context to Theoretical Computation

A Proposed New Unit to UVA's Theory of Computation Course

Megan Bishop
University of Virginia
Charlottesville, VA, USA
mb9qm@virginia.edu

ABSTRACT

In spending the last year as a teaching assistant for the UVA Theory of Computation class, it has become clear that students struggle to understand concepts they don't know how to apply practically. Dense theoretical topics in particular are no exception to this rule, as students have a much easier time conceptualizing earlier topics in the class that they can fit into what they've learned in previous CS courses. As the class moves on, topics move away from the courseworks students have already done, so it takes much more effort for them to visualize where this new knowledge can fit in. They would have a much easier time applying the newer concepts if they were provided with the context of how these theoretical topics are actually used in the software development field.

To fix this issue, I'm proposing a new unit to the Theory of Computation curriculum - applications of nondeterminism and finite state machines in software testing. One of the topics that students take the longest to understand conceptually is nondeterminism, introduced in the context of nondeterministic finite automata. Students have little motivation to learn this concept when they know that, logically, multiple choices in a decision cannot be made at once, so they don't see how it may apply in their future careers. In this unit, they will be introduced to non-determinism - specifically, methods of preventing non-determinism - in software testing, getting a glimpse into fields that use the theoretical topics they're studying now.

With the integration of this new unit, students will have a real world context in which to analyze nondeterminism of a system, and a motivation for creating finite state automata for the code they write and systems they help create.

INTRODUCTION

From both the perspective of a student taking a course, and as a teaching assistant helping to teach a course, it is obvious that there's a gap in motivation when a student is not given context, or a reason to care about why they need to learn the given curriculum. This can be as simple as knowing a certain part of the material will be on an upcoming exam, but long term retention of concepts is generally reliant on a broader application of the material. If a student does not think they will use the material they're learning in either a future class or in their eventual career, the chance of carrying that knowledge past the final exam is drastically lowered. With theoretical computation, this problem is especially prevalent, as the material is so different from much of what is taught in other courses. Many students have expressed the feeling that they are wasting their time in the class because they have no context for how they might use the material they study in the course, and simply get it over with for a requirement.

If students were aware that many of the topics that come up in the theory of computation course are actually relevant in the field of software development, they may be more eager to take the course, and learn the material more readily. Currently, engagement in the class is encouraged through the use of group work and examples that contextualize the topics in examples from everyday life. While having a group to participate in may motivate certain students to take the curriculum more seriously, other students are at a detriment from this setting, as they have their group mates to fall back on when they don't understand the material. Those students prone to using the success of their team to carry their own grade are even less likely to retain the topics taught after the end of the course.

As far as simplified examples of the topics, while these anecdotes used to teach the basics of a concept may help the student's original grasp on the information, they don't really contribute to long-term retention of the knowledge. In addition, some students find these examples frustrating, as they can be unrealistic or even impossible applications, so they do little to contextualize the material in their future career.

By implementing a unit directly identifying how the topics they've learned in the class may arise in their careers as software developers, these students will be motivated to get a better understanding of the information presented to them, as well as holding onto that knowledge for longer. When students can see that the curriculum of a course actually matters or will matter in the future, the interest will automatically rise to fit that perceived importance of the information.

BACKGROUND

As the UVA Theory of Computation class is currently taught, students are organized into working groups called "cohorts". Students in the same cohort collaborate on each problem set, and present their solutions to their TA leader on a weekly basis. This model of the theory of computation classroom creates lots of room for students to discuss the topics at hand more deeply than they might if they simply learned the material from lectures and completed problem sets on their own.

This class model also does not implement exams, so there is very little assessment outside of the team setting. Instead, the group model is utilized to motivate learning for the purpose of contribution to the cohort. Each student is asked to explain at least one randomly selected problem from the set each week, so students are encouraged to fill in the gaps in one another's knowledge, as they are all expected to have a good understanding of each group solution.

RELATED WORK

The best attempt at contextualizing difficult theoretical topics as the course is currently taught is the previously mentioned use of simplified real-life examples. For example, a common explanation for non-determinism in the course describes taking a drive to a friend's house, and how you might be able to arrive faster if you were able to non-deterministically take both turns at a fork in the road simultaneously. While this does help to simplify the material, as it stands these can decrease motivation to retain the material, as this is not an application that is possible to replicate in real life. These examples would not be a problem if they were followed by examples within the field of software development, but it is harder for students to stay motivated when their only examples are either very theoretical or unrealistic. They simply don't provide enough reasoning for why the material matters to the student in the long run.

Similar problems arise with the use of the cohort model of the class to keep students engaged. While there is not an exam for which the students retain the material and then forget soon after, once a student no longer holds responsibility to their group to assist with problem sets, their understanding of the material no longer matters as much to them. Even more, there is no reason given to the student to expand on their learning beyond the group setting, so extensions on the material that do not get addressed in the group discussion go unexplored by the student. A future motivation to retain the material may spark more of an interest in the material for a student, and encourage individual exploration on deeper facets of the subject.

DESIGN

To encourage deeper understanding and longer retention of the theory of computation curriculum, a proposed new unit, exploring uses of several different theoretical concepts discussed in the class was added. Referred to in the lecture material as "Non-Determinism in the Real World of Software Development", this unit dives into the practical application of the concept of non-determinism, as well as the use of finite state machines in the field of software testing. Drawing on material from the UVA software testing course, it explains first how some broad testing methods are based on the material learned through theory of computation, then gives an example of a commonly used testing structure in modern software development that builds on the integrated material from both courses.

Work on this new unit was started from the connection of course material from theory of computation and software testing, and ensuing research on how these topics could build together to new information not introduced in either course. The effects of non-determinism when testing software were explored, adding context to a rather broad theoretical topic introduced in the theory of computation course. From here, a relevant unit of the software testing curriculum was selected to bridge the gap between the two courses. Finally, a real world example of how these concepts come together in testing of actual software implementations was studied, using the context established by existing topics from the courses to break down the more complicated concepts of this new testing method.

Alongside the lecture material, a problem set in the style of existing theory of computation assignments was designed, aiming to have students apply their knowledge from the new unit to example implementations. A variety of proofs,

discussion questions, and testing procedure implementations were included in the assignment to reinforce students' understanding of all concepts covered in the applications unit.

1 Lecture Material Development

As this unit is planned to be implemented as the final unit in the theory of computation course, the lecture material begins with a brief review of the relevant topics - finite state automata and non-determinism - which are introduced much earlier in the semester. The differences between deterministic and non-deterministic state machines are highlighted to freshen the concept in the students' minds, then a simple example is used to demonstrate the differences in practice.

With non-determinism refreshed, the students are then introduced to new topics for theory of computation. The current version of the course explains the concept of nondeterminism without showing how it really applies in software development - specifically, it doesn't show the problems caused by non-determinism when testing a system. The new lecture material acknowledges that non-determinism cannot be realistically applied to daily life, but gives examples of types of software that can be negatively affected by non-determinism, as well as where non-determinism can be intentionally implemented in a system.

After the discussion of real world software non-determinism, these concepts are contextualized in the field of software testing, introducing the concept of a non-deterministic software test, and explaining how this can be detrimental to a system. These topics are discussed in detail to show students how the material is applied outside of the classroom, and how they may run into non-determinism in their future career, therefore providing motivation to engage with the lecture and retain the information longer.

Next, the lecture discusses a method of software testing they may not have been introduced to: model-based software testing. This unit of the software testing curriculum is specifically chosen to help students with their understanding of topics introduced later in the material. A description of model-based testing and the various approaches to it is given. A simple example of model-based testing is used to demonstrate how this testing approach uses their knowledge of finite state machines to create more comprehensive tests for a system, and overall reinforce understanding. Some more complicated implementations of model-based testing are briefly acknowledged to give students some direction to further their learning on their own time before moving on from the topic.

To finish out the lecture material, Petri-Nets are introduced to the students at a high level. This topic was chosen as an example of a real world testing procedure used for a variety of systems that any student may eventually work on. It also implements and provides a real software development context for each of the theoretical topics discussed earlier in the lecture. The differences between data flow graphs and the structure of Petri-Nets are discussed in the context of what students have already learned, and a small example is provided to show students how these testing models are used in larger, non-deterministic systems.

2 Problem Set Development

This unit works with the current cohort model of the course to create a collaborative problem set for each group to work on together. It seeks to both check the student's understanding of the material introduced in the unit, as well as give students a chance to expand on their learning through group discussion.

The assignment starts with a review problem to ensure that students have retained their learning on deterministic and non-deterministic finite state automata, as well as reinforce the connection between determinism and non-determinism by asking students to build and then convert the machines between the two types.

The problems then ask the students to apply their new knowledge of model-based testing, building on what they already know about finite state machines. They are asked to replicate the example from the lecture with a different code segment to check their understanding of the model-based testing process. They must evaluate the model they create from both the node coverage and edge coverage criteria so they may observe for themselves the difference between the approaches.

The next question is an expansion opportunity, asking students to examine a feature of model-based testing that wasn't explicitly discussed in the lecture. This is done to ensure that the students can go beyond a surface level understanding of this testing procedure.

The assignment's next few questions aim to connect the major concepts from the theory of computation and software testing halves of the unit. First, they are asked to apply non-determinism to the model they have created, or explain why this is an impossible task, if that is found to be the

case. Following this, they are asked to update the given code to reflect changes made to their model (if they were able to make the specified changes at all). This question ensures that students understand how non-determinism can present itself in a system they implement, and reinforces a greater understanding of how to avoid this problem when writing code in the future. Finally, students are asked to provide a proof as to why it would be impossible to use complete path coverage on any code/data flow graph that uses a loop. Students should be able to identify that the path can always be expanded so long as there is a loop, so there is an infinite number of larger and larger (mostly redundant) paths that would need to be traversed in order to complete the criteria. This question checks both the student's proof writing skills and their understanding of criteria fulfilment when using model based testing procedures.

The assignment ends with a series of discussion questions, meant to spark deeper conversations within their cohort about the topics they have been introduced to. They are asked to identify potential real-world applications of non-determinism beyond those mentioned in the lecture, encouraging them to think critically about the functions of a system that may cause it to run differently at different times. Next, they are asked to analyze the two types of model based testing introduced, and connect them with the concept of non-determinism, encouraging more connections between the two topics. Finally, they must analyze the new version of transitions introduced when discussing Petri-nets, and how this can help a system to test deterministically, even if the system is by nature non-deterministic. This last discussion drives home the main theme of the unit, bringing the wider context of applications in to reinforce or expand their learning of theoretical topics.

3 Connection of Theory of Computation and Software Testing Course Materials

Since students currently taking theory of computation may not have taken the software testing elective yet (or may never plan to take that course), the lecture material thoroughly follows the surface level introduction of model based testing, and how it is used to develop test cases. However, it is introduced somewhat differently than it would be in the software testing elective alone. To emphasize the connection of the two courses, it introduces this testing method in the context of a theory of computation topic: finite state automata. A careful note is made, highlighting the differences between the finite state machines as they have learned them, and the implementation of finite state machines (data flow graphs) used for model based testing. Only after this connection is established does the material explain how to use model based testing as a testing medium.

This connection is further reinforced by the problem set, particularly the later model-based testing questions, which ask the students to apply the theory of computation topics of non-determinism as well as proof techniques to the newly introduced topic of model based software testing. This goes even further with the discussion questions at the end of the assignment, as each asks the students to apply their knowledge of non-determinism to the context of software testing, and make sure they understand how these two concepts connect in the software development field.

IMPLEMENTATION

The unit is intended to be implemented as one of the final units in the course, concluding the study of theoretical computation with the motivation to maintain their understanding of what they've just learned. Currently, the final unit of the course is a wrap-up/review unit, in which the students put together some sort of creative project to display their understanding of one of the topics discussed in the course. While creativity can be a good thing to encourage students to apply their knowledge outside of the theoretical context, many students end up taking this unit to simply create something entertaining, not delving any deeper than surface level into the concepts of the course. This new unit could be used as a wrap-up that fully encourages the students to apply and contextualize their knowledge, reinforcing it in their mind for future use. It provides a slightly larger challenge to a student, but ultimately a challenge that pays off in motivation of further learning.

CONCLUSIONS

The proposed new unit to the UVA theory of computation course was designed to provide students with a deeper understanding for the context of theoretical concepts. This new unit aims to solve the problem of students being unmotivated to learn and retain certain concepts from the course by showing them how they can apply in the greater context of software development, as well as where they might encounter them in their future careers. With the integration of elements from the software testing course, students will be able to frame their learning in realistic development scenarios, closer to what they experience in other, non-theoretical courses. With successful implementation, this unit could provide students with the

motivation to extend their learning beyond what they need to pass the class, and explore further applications of the topics they have learned about in the course.

FUTURE WORK

Beyond implementing the application unit to theory of computation, students' learning could be further enhanced by implementing small context examples throughout the course, expanding on more than just non-determinism and finite state automata. While these are some of the most important topics to provide students with context, as they are some of the more theoretical rather than practical topics, other units, such as uncomputability and turing machines could use some practical extensions to round out their units of the course. This way, students could be introduced to practical applications little by little throughout the learning process rather than wait until the end to find out the context of the topics they've been studying. This could be somewhat more effective at helping students to retain their knowledge of these theoretical concepts at their first introduction, rather than fortify them with a reintroduction weeks later. It could also help to make the whole course feel more practical to the students as they build an understanding of theoretical computation.

REFERENCES

[1] Donald Firesmith, 2017. The Challenges of Testing in a Non-Deterministic World. *SEI Blog,* Carnegie Mellon University.

[2] Donald Firesmith, 2017. Seven Recommendations for Testing in a Non-Deterministic World. *SEI Blog,* Carnegie Mellon University.

[3] Mustafa Bozkurt, Mark Harman, Youssef Hassoun. Testing Web Services: A Survey. *Centre for Research on Evolution, Search, and Testing*. King's College London. Technical Report: TR-10-01.