Formal Methods and Computing Identity-based Mentorship for Early Stage Researchers

Mark Santolucito
Yale University
New Haven, CT
mark.santolucito@yale.edu

Ruzica Piskac
Yale University
New Haven, CT
ruzica.piskac@yale.edu

ABSTRACT

The field of formal methods relies on a large body of background knowledge that can dissuade researchers from engaging with younger students, such as undergraduates or high school students. However, we have found that formal methods can be an excellent entry point to computer science research - especially in the framing of Computing Identity-based Mentorship. We report on our experience in using a cascading mentorship model to involve early stage researchers in formal methods, covering our process with these students from recruitment to publication. We present case studies (N=12) of our cascading mentorship and how we were able to integrate formal methods research with the students' own interests. We outline some key strategies that have led to success and reflect on strategies that have been, in our experience, inefficient.

ACM Reference Format:
Mark Santolucito and Ruzica Piskac. 2020. Formal Methods and Computing Identity-based Mentorship for Early Stage Researchers. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366957

1 INTRODUCTION

Engaging in early mentorship of students in computer science has been shown to be an effective means to improve outcomes in diversity, retention, and performance [13]. In this paper we report on our experiences in mentoring early stage researchers (such as high school students and undergraduate students). In our report, the mentors are based in Yale University, however the mentored students came from various schools, even including from abroad. We introduced the students to research in formal methods, which is a subfield of computer science focused on improving software reliability using formal mathematical techniques. Traditionally, research in program verification and formal methods is considered a part of computer science with a high entry bar [7, 31]. Conducting research in formal methods requires practical expertise in programming languages and systems, as well as training in theoretical mathematical foundations and logic. In addition, courses in formal methods are usually taught only at universities and on a graduate level. Therefore, we first needed to introduce the students to the field of formal methods, and after that, additionally find a suitable project for each student.

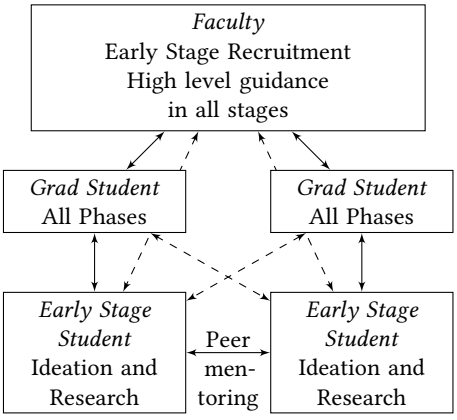

Figure 1: Our mentorship model engages all participants with each other.

We report on our experiences mentoring students in formal methods, with a focus on program synthesis, using the lens of Computing Identity-based Mentoring [3]. Computing Identity-based Mentorship posits that by engaging students' own backgrounds and interests we can help students build a sense of identity around computing. By engaging student with their own interests, it is possible to build a computing identity that helps to further engage and retain students in the field [4]. Through a number of case studies of our mentorship experiences, we provide anecdotal evidence of how the field of formal methods can be leveraged to engage students in their own interests, while also building a common community of practice [30]. Encouraging communities of practice in computer science has been shown to lead to stronger student outcomes [8, 15, 21].

We frame our mentorship model as a type of *cascading mentorship* [20], implemented similarly to prior mentorship models for early stage computer scientists [28]. The cascading mentorship model reimagines the mentor-mentee roles as interchangeable - asking mentees to act as mentors through mentoring younger students (as in [20]), or through peer mentorship.

Our mentorship model, derived from the work of Tashakkori et al [28], is shown in Fig. 1. The faculty mentor oversees the graduate students, who themselves closely work with a small number of early stage researchers. In addition to the graduate student's direct

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366957

mentees, every graduate student also engages to some degree with the mentees of other graduate students as well. Additionally, the younger students are also put in a position to act as peer mentors to other early stage researchers. All the while, the faculty gives an opportunity to the younger students to more formally present their progress. We report in our case studies how, for some students, we have seen this strongly connected social graph within the lab environment lead to a stronger sense of community.

In our presentation of our case studies (N=12), we break down the mentorship into three phases. First, we explain the recruiting phase to give insight into how the opportunity for mentorship began. Second, we look at the ideation stage of the research, where the faculty member, the graduate students, and early stage mentees collaborate to define and scope an appropriate project for the allotted time. Third, we look at the research stage itself, where the mentee is actively involving in producing novel results. Finally, in the presentation of each case study, we reflect on the outcomes of the mentorship, in terms of concrete benefits both for the mentees and the mentors. In many cases, the research resulted in publication.

When examining the positive outcomes of the mentorship experiences, we frame the student experience's in the context of the Thomas principles of mentoring success [29]. For the sake of clarity and concision, we in particular focus on the following principles:

(1) *Identity Development* - forming and reinforcing a Computing Identity.
(2) *Academic Support* - ensuring efficient access to expertise within the lab setting for guiding mentees and ensuring continued progress.
(3) *Sense of Belonging* - building communities of practice in our research field of formal methods.
(4) *Leadership Development* - developing mentees' sense of leadership and local expertise within our community of practice encourages further engagement.

2 FORMAL METHODS AS INTRODUCTORY RESEARCH

One of the key insights of prior work was that students seeing a clear impact of their work can have a positive impact on retention [4]. In this report, we focus on our own research programs in the field of formal methods and the potential for students to see impact.

The main goal of research in formal methods is to gain better insight into code by using formal mathematical reasoning. Research in formal methods began with development of computer science: for instance, works of Alan Turing already talk about formal proofs of program correctness. However, only recently we have seen formal methods being applied to industrial software. Today, formal methods are used in many major software companies, including Google [1], Amazon [2], Facebook [9] and Microsoft [18].

The range of applications where formal methods are used covers almost all aspects of computer science: we can obtain guarantees that servers are secure [23], ensure that data centers will not crash [32], assist student programmers with bugs [6, 16], or write programs that write programs [12] - to name a few applications.

Taking into account the increasing need for scientists working in formal methods, there has been significant community effort in developing programs to assist early stage researchers in covering the basics of the field [10, 19]. These programs are also designed to help lower the barrier to entry for formal methods research. This is in part because, within the field itself, formal methods is seen as having too high an entry bar to make it an appropriate field for students' initial experience with research without significant preparation.

While this perceived high barrier to entry may at first seem to imply it is not a good fit for younger students with limited programming experience, we have found the opposite to be true. In the subsequent sections we describe a number of case studies of our collaborations with high school and undergraduate students in our lab. All these projects were defined after initial discussions with the students and tailor-made to fit around the student's application domain of interest.

Of all the fields of formal methods, we found that students were particularly interested in software synthesis and its applications. The goal of software synthesis is to automatically derive code based on given specifications. A specification describes *what* the program should do without going into details of *how* it should be implemented. There are various ways of providing the specification: one of the most commonly used approaches is to take a set of examples that clearly illustrative the intended behavior of the code.

Based on these examples, a synthesis tool should automatically generate corresponding code. This branch of synthesis is known as "programming by example" [5]. While program synthesis might have looked like an unreachable goal a decade ago, today – due to the development of formal methods – the class of programs that can be automatically synthesized has dramatically increased. Program synthesis is even used in industrial software: it is vital part of the Flash Fill feature in Microsoft's Excel [14].

3 RESEARCH CONDUCTED BY HIGH SCHOOL STUDENTS

In this section we describe the projects that were conducted by high school students ($n = 4$). All these projects are related to software synthesis, as the concept of program synthesis was the easiest for us to describe to high school students, and they could immediately see the impact of their work.

Of the four projects we present here, one used the research experience as part of a course credit, while the other three included formalized summer internship experiences. We conducted semi-structured exit interviews with these three students. The projects described here were largely defined in content and in scope by us, but the technical solutions were driven by the students. In all cases, the project topic was found as an intersection between our group's ongoing work on program synthesis, and the students' background and existing interests in computing.

3.1 Synthesis + HCI

Recruiting: Our first high school research intern came to us from a cold email (as the student entered 11th grade). This student reached out, as part of a research course at their high school, to seek out opportunities for collaboration. The student mentioned our publications on program synthesis in their email, and was interested in the potential benefit of synthesis for new programmers. The student

had prior research experience, having done a Human Computer Interaction (HCI) study as a school project.

Ideation: In initial discussions, the student demonstrated interest in HCI. We had an existing program synthesis tool implemented to apply the programming-by-example paradigm to PowerShell scripts. While we had already developed the tool itself, we were lacking any usability studies. To complement the students' existing interests, we planned to develop a user study to learn about what users expect and want from a programming-by-example engine.

Research: In developing the user study, the student made a point to include both timing information (how quickly participants complete the tasks) as well as how helpful the participants found the tool. To our surprise, we found that while participants completed tasks more quickly with programming-by-example, they found manually writing code to be more helpful [25].

Outcomes: We published these results with the high school student and his teacher. While we largely handled the writing ourselves, the student drove the analysis and interpretation of the results with the help of his teacher. This allowed the student to actively participate in the publication process, without the prohibitive overhead of learning scientific writing. This helped the student develop a leadership role as the local expert on user study analysis.

We first presented the work at a workshop without proceedings, and had a graduate student present a talk. We later published the work at another workshop, and the high school student used a recording of the graduate student's talk as guide in the preparation of their own presentation. In having the student present the work, we helped form a sense of computing identity in the student, such that the student is now majoring in Computer Science at college.

3.2 Synthesis + Machine Learning

Recruiting: One high school student was recruited (while the student was in 11th grade) during one of our outreach activities, at a hackathon in Bermuda [11]. At this hackathon, we were running a workshop on machine learning techniques for local Bermudian high school students with prior experience with programming. We contextualized the machine learning workshop within our research in order to both deliver the content and expose the students to novel applications of the material. One student was particularly interested in the application of machine learning to program synthesis, so we offered to stay in contact to begin a research collaboration.

Ideation: During the hackathon, the student had already discussed possible applications of their own expertise, machine learning, to our work on program synthesis. Through the research stage, we iteratively refined the topic such that this became its own research project that was not reliant on any of our existing work. The project was still framed in such a way so that it directly helped our ongoing work.

Research: We worked with this student remotely after the hackathon in December for four months, and in May invited the student to join us to work in person for a four week internship in the summer. Being both underage and an international student, the process of formalizing the internship presented challenges. The best solution we found in the end was to have this student's stay in New Haven accompanied by their parent. The student's work was largely self-directed in technique, while we provided guidance in scoping the project appropriately. We were aiming for a scope such that we could have a measured impact, but also complete an initial evaluation by the end of the students' visit to campus.

Outcomes: The student was able to develop a tool with strong results and subsequently published a paper [22]. In our exit interview with the student, we saw that the student connected to the impact of their work in the larger research community.

> "If you're very interested in the topic while you're in high school, I think [research experience] is beneficial due to the fact that it allows you to really experience how real world problem solving happens in a very real environment."

We also saw the student developed a sense of belonging that was conducive to productive research conversations. When asked about the benefits of the internship experience, the students responded:

> "Just being around the office and having fun conversations about things I'm interested in, and that being okay, just being able to have fun in what you're interested in."

3.3 Synthesis + Apps

Recruiting: Another high school student came to us from a cold email (as the student entered 12th grade) with an interest in developing mobile applications. This student reached out on the recommendation of friends to email professors to find research opportunities. The student reported reaching out roughly 100 professors, and reported hearing back at all from about five, including our group. We were able to find intersection with our research on reactive systems synthesis [12].

Ideation: In our initial Skype sessions, we worked with the student to design the high level goal of the project and layout the way in which we could find an overlap with the student's existing interests. The student had sent a resume which mentioned a course that involved VeriLog, a programming language popular in the 'reactive synthesis" subfield of formal methods As such we searched for projects that overlapped with our own work on reactive synthesis The goal of this student's project was to reuse the existing program synthesis infrastructure we had built for general reactive synthesis problems, and adapt it to synthesize an Android app in Kotlin.

Research: We initially worked with the student over Skype, then invited the student to work with us over for four weeks over the summer. Again, as an underage international student, this student's stay in New Haven was accompanied by their parent. This project was a good fit for the student, who had limited programming experience, as it the programming aspect mostly involved editing existing parsing code, and following predefined patterns to generate program code. The student had a stronger math background, and was able to explore more of the theoretical problems with reactive synthesis for Kotlin. We did not anticipate having the student address these theory issues, but the student in fact took the project in this direction themselves after completing the programming aspect of the project.

Outcomes: The majority of this student's research experience was during an onsite internship that was overlapping with the

student from Sec. 3.2. Organizing overlapping internships was recommended a key lesson learned in organizing internships from prior work [13]. We confirmed that this is a productive insight in our own experience. Throughout their time on campus, the two students worked closely together on both projects - helping each other with both technical and conceptual issues. When asked to describe some benefits of the internship experience, the student said:

> "I'm very comfortable and the environment is such that I can go out and say 'Hey, do you think you could help me with this problem' and everyone is super open to helping me."

3.4 Synthesis + Hardware

Recruitment: We recruited one high school student (as the student entered 10th grade) from the local New Haven area. This student had been working on writing a compiler for Basic as a side project and reached out for some guidance on ways to get more involved in computer science.

Ideation: We worked with this student for roughly a year - mostly over email with occasional in-person meetings. Because this student was interested in low-level language details, we designed a project working with intermittent computing and program synthesis. The goal of this project was to synthesis code that is able to run on specialized hardware devices that run on harvested ambient (e.g. radiowaves) energy.

Research: The student's project has two components - a hardware side focused setting up a test framework on the specialized hardware - and the software component of program synthesis. The student first setup the hardware and gained basic familiarity with work in this domain. The student's first approach to synthesis was to utilize their existing work on writing a compiler. While this was not an efficient solution in the end, it gave the student exposure to the technical details necessary for working in this domain. With that background, the student was able to reframe and rescope their work into a new direction.

Outcomes: While the hardware setup was less of a novel research project, it gave the student a strong sense of local expertise with these devices. The student worked primarily with one graduate student, but was able to get feedback from another graduate student with a background in electrical engineering.

As a local student, the collaboration is logistically simple to facilitate as an ongoing project. We also hosted this student for a four week internship during the same time period as the students in Sec. 3.2 and Sec. 3.3. One negative results we found with this setup was that, while the students occasionally worked together, the sense of peer mentorship was not as strong with this student. We hypothesis the age gap may have contributed to this, as well as the student's prior local commitments decreasing the amount of interaction time with the other two interns.

4 RESEARCH CONDUCTED BY UNDERGRADUATE STUDENTS

In this section we describe projects involving undergraduate students ($n = 8$). While these projects might not seem more demanding or complex than the projects conducted by high school students, undergraduate students were given more freedom in defining their research agenda. Their projects were initially less clearly defined. They were given a general description of a problem and led the discussion about possible research directions. In this way, they helped to outline the project, based on their own research interests, which made the students additionally motivated to participate in the project. Their research interests were often defined either by courses that they had taken until that point, or by their extracurricular activities. All the projects presented in this section are either published at a top venue, or are currently under a submission to a top tier conference. The students conducted their research either as a part of a summer research internship, or received course credits, or were working on these projects in their own free time.

4.1 Synthesis + HCI

Recruitment: One student (a rising senior at the time of recruitment) came to us at the end of the school year looking to gain research experience, with the intention of applying to graduate school. The student expressed an interest in topics of HCI, but did not have a concrete project in mind. As much of our group's work in formal methods focuses on applications intended to help developers, we are nearly always in need of further user studies.

Ideation: We presented a number of our projects to this student, and asked the student to identify a preference and direction within the HCI space. The student was interested in our work on synthesis in a "live" environment, whereby program synthesis runs in realtime to assist developers as they are writing code. We had some initial work on developing the tooling for this project, but lacked any user studies to drive our interface decision making.

Research: The student worked in a 12-week onsite internship to design and implement an online version of the live synthesis interface in order to then deploy the user study to a wider audience. The student also designed the user study, and conducted numerous pilot studies with early iterations of the online interface and the study design.

Outcomes: At the time of writing, the study is currently being deployed, and the student is continuing to work with us to prepare a paper submission. When asked to reflect on the impact of the internship experience in an exit interview, the student said:

> "I think one reason that I find going into academia, as a Chicano, so important is that I want to see other people like me in these fields and sometimes it's difficult to see yourself in these places, see yourself in academia, going to grad school when you don't see many leaders like that. But through this experience I found there is a lot of great and supportive people within the field and through that I feel so much more confident about getting into academia, applying to grad school, and cranking out a thesis one day"

4.2 Synthesis + Systems

Recruitment: Two undergraduate students were working on this project. One student approached us after taking the course (in their 3rd year) given by our group and asked to work on a verification-related project. The other student was a personal friend of group

members so initially became interested in the project through discussions with group members. As the project progressed, this student (in their 4th year) became more involved and took a lead for certain parts of the project. Both students were supervised by a PhD student, using the cascading mentorship model.

Ideation: Our group had previously worked on configuration file analysis. Incorrectly setting up a configuration file has been found to cause more server outages than bugs in the code. These students worked on learning specifications of correctness of configurations by analyzing a large number of existing configuration files. From that corpus we learned properties about configuration files, and used them as a specification, enabling us to formally verify configuration files and detect previously unknown bugs.

Outcomes: This work resulted in a paper at a top venue [27]. Both students were exposed to the formal methods community either at a conference, or by attending summer schools. While both students worked as software engineers upon graduating and completing this project, one of them has recently returned to a non-CS graduate school at Yale.

4.3 Synthesis + Music

Recruitment: One non-traditional application field we have found rich collaborations from is computer music. We have had three students (at the time of their recruitment, a 4th year, and two 2nd years) work with us on a project combining program synthesis with music. One student participated in a 12-week internship, one participated in this project as their senior project, and one is currently participating as an extracurricular activity.

Ideation: We initially started this project because one PhD student was interested in the intersection of formal methods and music. The goal of the project was to build a programming-by-example engine for audio files. The engine takes two audio wave files, and automatically synthesizes digital signal processing programs that transform the input to the output audio file. From a technical perspective, this work focused on combining machine learning techniques with formal methods for application in music.

Research: The first student worked on this over the summer and was mainly focused on implementation, building much of the core codebase of the project that handles the machine learning components. The senior thesis project focused on developing a theory of how to apply formal methods techniques to the project. Currently, the most recent undergraduate researcher is working on implementing and extending this theory.

Outcomes: The work of two of the students on this project was published at a specialized computer music workshop that focuses on programming language design [26]. In the vein of institutional academic support, one student was able to use this project to gain credit for their senior thesis.

4.4 Symbolic Execution Engine

Recruiting: This student started to attend our group meetings immediately after joining Yale (a freshman at the time of recruitment) in order to learn more about research.

Ideation: Due the opportunity of getting this student involved early, we gave the student an especially challenging project of developing a symbolic execution engine for Haskell programming language. The student was directly supervised by a PhD student from the group, but also worked independently, developing their own research agenda.

Research: A symbolic execution engine takes as input a program, and instead of executing the program on some concrete values, it runs the program using symbolic values. The result is a mathematical formula that describes the program's behavior. Although symbolic execution engines are well studied, languages like Haskell, based on lazy semantics, had no efficient symbolic execution engine. The problem required a deep understanding of Haskell's semantics: the research combined some foundational theoretical problems, but was also implementation-intensive.

Outcomes: While the student kept working with our group throughout their undergraduate program, they also developed other research interests at the intersection of mathematical reasoning and computer science. Nevertheless, the symbolic execution engine work resulted in papers accepted and presented at two top ACM sponsored venues [16, 17]. In addition, we helped the student explore other research directions, and the student did three research internships: one at a different university, one at an international research institute, and one at a research lab of a large company. The student presented work at various international meetings and volunteered at conferences, developing a strong sense of helping the community. The student also received the 2019 NSF Graduate Research Fellowship, was accepted to several PhD programs, and is currently attending one of them.

4.5 Program Repair

Recruiting: The student took a course (during their 3rd year) from our research group and asked directly about a possibility to conduct an independent research study. The student worked directly with a professor, as the group had no graduate students at the time.

Ideation: The student was doing a double major in math and computer science, and was interested in finding a way to leverage both these backgrounds.

Research: When writing code, a user might be sure about what they want to write, but we are not sure about the right ordering of all arguments when invoking library functions. When programming, a user might write code that does not compile but clearly outlines their intentions. The student used their expertise in graph algorithms to develop a tool that repaired these errors.

Outcomes: This work was published and presented by the student at a top conference [24]. The student also presented a poster at the ACM Student Research Competition, and received second place. The student was accepted to several PhD programs, and is currently attending one of these schools.

5 LESSONS LEARNED

Reflecting back on our experiences with using formal methods research as an entry point to computer science research and a way to build a computing identity, we explore here some key lessons we have learned.

5.1 Recruitment

Recruiting students for collaboration has been one of the most important steps of our process. At Yale University, we are privileged

to have a large pool of talented undergraduates to draw from - however the main challenge is awareness. There is some existing infrastructure in place - as in many universities - to encourage Computer Science majors to complete a senior thesis as a research collaboration with a lab. While this is effective, as such a thesis is completed in the students' final year, the student then leaves just as they begin to be particularly productive from a research perspective. Involving students at early stages in their academic career not only has helped the students themselves, but also has increased the long term quality of our collaboration.

Unfortunately, due to the time constraints on faculty members, proactive recruiting dedicated to high school students was not a practical strategy in our situation. The cold email is an increasingly common strategy for high school students to get involved in research. Students reach out to a large number of professors (in the case of Sec. 3.3, as many as 100) in hopes that one will respond and allow them participate in an unpaid internship in their lab. While the volume of these emails can be overwhelming, we have had success by forwarding these students to graduate students, following the cascading mentorship model. In this way, the faculty member provides mentorship training opportunities to the graduate student, and the graduate student can utilize the students' assistance.

In terms of selecting high school students, while the high school students' resumes provided some clues as to their prior experience, we found the students to be too young for their resume to be a useful predictor of their success in research. Generally, demonstrating some prior experience and interest in programming was sufficient. Beyond this, students largely self-selected when presented with concrete research tasks. However, from another perspective, this is a potentially negative result. While we have had a number of successful high school interns, as listed in Sec. 3, a number of students have also dropped contact with us after a short time. Investigating effort-effective strategies to stay engaged with more students who initially reach out for collaboration is a space for further research.

5.2 Ideation

We learned to ensure that student projects are *noncritical paths* along our larger research vision, but still contribute significant value, which encourages the students' sense of computing identity. Additionally, granting students the latitude to guide their project scope and how their project integrates into the larger research vision encourages the development of a stronger computing identity, with the student as the leader, or "local expert", of their topic.

The benefits of mentorship go beyond the mentees, especially with the cascading mentorship model. For graduate students, the process of advising students and defining scope of projects is valuable experience. In our experience, undergraduate research is generally fairly regulated, such as being formalized as a course, a summer internship, or a campus job. In contrast, working with high school students allows graduate students to take more risks with mentorship, which in turn creates more learning opportunities. Especially for graduate students who are planning to go on the academic job market, the experience of running a 'micro-lab' environment has been particularly useful.

5.3 Research

The first and most important guideline for our research programs has been to provide academic support to early stage students - especially in stressing the importance of asking the right questions. Students have tended to ask too few questions, and waste time on technical challenges that can be answered quickly by the mentor. This is dangerous to the success of the collaboration, as it can cause students to lose interest in the project.

Consequentially, we have found it to be important that the mentor makes sure the student feels comfortable asking questions. However, sometimes a question will be more appropriate for the student to discover on their own. This is especially important for developing computing identity, as we have seen this provide students the confidence needed to solve problems on their own.

In working to minimize the interruptions to the workflow of graduate students (as mentoring students was *not* the graduate students' primary responsibility), we found that having multiple students working on projects at the same time and *in the same space* encouraged peer mentoring. By connecting the students, they can lend their expertise to each other and progress more quickly. This worked particularly well with the high school students, but had not worked well with the undergraduates. We suspect that this was due to the great flexibility the undergraduates had in the physical workspace. We plan to further investigate strategies to increase informal peer mentoring among undergraduate researchers.

6 PUBLISHING

In our experience, early stage students have had limited scientific writing experience, so the majority of the initial drafts were written by graduate students and faculty. However, we have been able to keep students engaged during the publication process by assigning other critical tasks, for example, in the analysis of data.

When possible, ensuring that publication happens before January of the students' senior year of high school will deliver the most value to the student, as the publication can be included in their college application. This acts as a good motivator for them to complete their projects by a hard deadline.

One challenge we have found with the computer science conference model is the challenge of travel. Publishing in conferences that are geographically far from the students has presented challenges in allowing the student to fully participate in the research experience. While we have sometimes been able to fund undergraduate researchers during their time at the undergraduate institution, if a student graduates before the conference takes places, it becomes more difficult to find the resources to allow the student to attend. In the case of one of the student co-authors in Sec. 4.2, the student was able to use funding from their company to attend the conference where we had published the work, as the work was also aligned with the goals of their post-graduation employer. However, this problem is even exasperated in the case of high school students. Our current solution is to intentionally target conferences that are relatively local to allow the student to attend as well.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1553168 and No. CCF-1715387.

REFERENCES

[1] Domagoj Babic, Stefan Bucur, Yaohui Chen, Franjo Ivancic, Tim King, Markus Kusano, Caroline Lemieux, László Szekeres, and Wei Wang. 2019. FUDGE: fuzz driver generation at scale. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019.*

[2] John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. 2018. Semantic-based Automated Reasoning for AWS Access Policies using SMT. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018.*

[3] Kristy Elizabeth Boyer, E. Nathan Thomas, Audrey S. Rorrer, Deonte Cooper, and Mladen A. Vouk. 2010. Increasing Technical Excellence, Leadership and Commitment of Computing Students Through Identity-based Mentoring. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10).* ACM, New York, NY, USA.

[4] Jennifer Burg, V. Paúl Pauca, William Turkett, Errin Fulp, Samuel S. Cho, Peter Santago, Daniel Cañas, and H. Donald Gage. 2015. Engaging Non-Traditional Students in Computer Science Through Socially-Inspired Learning and Sustained Mentoring. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15).* ACM, New York, NY, USA.

[5] Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky (Eds.). 1993. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA, USA.

[6] Loris D'Antoni, Dileep Kini, Rajeev Alur, Sumit Gulwani, Mahesh Viswanathan, and Björn Hartmann. 2015. How Can Automatic Feedback Help Students Construct Automata? *ACM Trans. Comput.-Hum. Interact.* (2015).

[7] Jennifer A. Davis, Matthew A. Clark, Darren D. Cofer, Aaron Fifarek, Jacob Hinchman, Jonathan A. Hoffman, Brian W. Hulbert, Steven P. Miller, and Lucas G. Wagner. 2013. Study on the Barriers to the Industrial Adoption of Formal Methods. In *Formal Methods for Industrial Critical Systems - 18th International Workshop, FMICS 2013, Madrid, Spain, September 23-24, 2013. Proceedings.*

[8] Betsy DiSalvo, Mark Guzdial, Amy Bruckman, and Tom McKlin. 2014. Saving face while geeking out: Video game testing as a justification for learning computer science. *Journal of the Learning Sciences* (2014).

[9] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling static analyses at Facebook. *Commun. ACM* (2019).

[10] Constantin Enea, Vasco Manquinho, and Ruzica Piskac. [n.d.]. VMCAI Winter School 2019. http://vmcaischool19.tecnico.ulisboa.pt/. Accessed: 2019-08-28.

[11] Jahde Eve. 2019. http://www.code441.com/. Accessed: 2019-11-30.

[12] Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. 2019. Temporal Stream Logic: Synthesis Beyond the Bools. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I.*

[13] Meg Fryling, MaryAnne Egan, Robin Y. Flatland, Scott Vandenberg, and Sharon Small. 2018. Catch 'Em Early: Internship and Assistantship CS Mentoring Programs for Underclassmen. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18).* ACM, New York, NY, USA.

[14] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets using Input-Output Examples. In *POPL'11.*

[15] Denise Gürer and Tracy Camp. 2002. An ACM-W literature review on women in computing. *ACM SIGCSE Bulletin* (2002).

[16] William T. Hallahan, Anton Xue, Maxwell Troy Bland, Ranjit Jhala, and Ruzica Piskac. 2019. Lazy counterfactual symbolic execution. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019.*

[17] William T. Hallahan, Anton Xue, and Ruzica Piskac. 2019. G2Q: Haskell constraint solving. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell, Haskell@ICFP 2019, Berlin, Germany, August 18-23, 2019.*

[18] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. 2017. IronFleet: proving safety and liveness of practical distributed systems. *Commun. ACM* (2017).

[19] Alexey Ignatiev, Antonio Morgado, Nina Narodytska, and Vasco Manquinho. [n.d.]. SAT/SMT/AR Summer School 2019. https://reason.di.fc.ul.pt/ssa-school-2019/. Accessed: 2019-08-28.

[20] Yasmin Kafai, Jean Griffin, Quinn Burke, Michelle Slattery, Deborah Fields, Rita Powell, Michele Grab, Susan Davidson, and Joseph Sun. 2013. A Cascading Mentoring Pedagogy in a CS Service Learning Course to Broaden Participation and Perceptions. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13).* ACM, New York, NY, USA.

[21] Andrew J. Ko and Katie Davis. 2017. Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17).* ACM, New York, NY, USA.

[22] Kairo Morton, Bill Hallahan, Elven Shum, Ruzica Piskac, and Mark Santolucito. 2020. Grammar Filtering For Syntax-Guided Synthesis. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI).*

[23] Lawrence C. Paulson. 2019. Inductive Analysis of the Internet Protocol TLS. *CoRR* (2019). http://arxiv.org/abs/1907.07559

[24] Alex Reinking and Ruzica Piskac. 2015. A Type-Directed Approach to Program Repair. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA.*

[25] Mark Santolucito, Drew Goldman, Allyson Weseley, and Ruzica Piskac. 2018. Programming by example: efficient, but not "helpful". In *Evaluation and Usability of Programming Languages and Tools (PLATEAU) at OOPSLA.* Also presented at SYNT 2018.

[26] Mark Santolucito, Kate Rogers, Aedan Lombardo, and Ruzica Piskac. 2018. Programming-by-example for Audio: Synthesizing Digital Signal Processing Programs. In *Function Art and Music (FARM) at ICFP.* http://marksantolucito.com/dsp-pbe.pdf

[27] Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, and Ruzica Piskac. 2017. Synthesizing Configuration File Specifications with Association Rule Learning. *Proc. ACM Program. Lang.* OOPSLA (Oct. 2017).

[28] Rahman Tashakkori, James T. Wilkes, and Edward G. Pekarek. 2005. A Systemic Mentoring Model in Computer Science. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1 (ACM-SE 43).* ACM, New York, NY, USA.

[29] Nathan Thomas. [n.d.]. Mentoring | STARS Computing Corps. https://www.starscomputingcorps.org/mentoring. Accessed: 2019-08-28.

[30] Etienne Wenger, Richard Arnold McDermott, and William Snyder. 2002. *Cultivating communities of practice: A guide to managing knowledge.* Harvard Business Press.

[31] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal Methods: Practice and Experience. *ACM Comput. Surv.* (Oct. 2009).

[32] Ennan Zhai, Ruzica Piskac, Ronghui Gu, Xun Lao, and Xi Wang. 2017. An auditing language for preventing correlated failures in the cloud. *PACMPL* OOPSLA (2017).