BEATCS no 117

AUTOMATA TUTOR AND WHAT WE LEARNED FROM BUILDING AN ONLINE TEACHING TOOL

Loris D'Antoni†, Matthew Weaver†
Alexander Weinert‡, Rajeev Alur†
†: University of Pennsylvania ‡: RWTH Aachen University

Abstract

AUTOMATA TUTOR is an online tool that helps students learn basic concepts in theory of computation, such as finite automata and regular expressions. The tool provides personalized feedback when students submit incorrect solutions, and also helps teachers managing large classes by automatically grading homework assignments. AUTOMATA TUTOR has already been used by more than 2,000 students at 12 different Universities in 4 different continents.

In this paper, we summarize our experience in building such a system. We describe the algorithms that are used to produce personalized feedback, and then evaluate the tool and its features through extensive user studies involving hundreds of participants.

1 Introduction

Both online and offline, student enrollment in Computer Science courses is rapidly increasing. For example, enrollment in introductory CS courses has roughly tripled at UC Berkeley, Stanford and the University of Washington in the past decade [12]. In addition, Computer Science is the most frequently taken MOOC subject online [17, 10]. With roughly a thousand students in a lecture hall, or tens of thousands following a MOOC online, approaches like manual grading and individual tutoring do not scale, yet students still need appropriate guidance through specific feedback to progress and to overcome conceptual difficulties. Many tutoring systems have been proposed to assist students learning various aspects of computer science such as LISP programming [3] and SQL queries [11]; however, these tools are typically not able to grade the student solution or provide actionable feedback. Recently there has been work on applying computing power to

teaching tasks including grading programming problems [15] as well as generating and grading geometric constructions problems [8, 9]. The tool we describe in this paper falls in this line of work.

AUTOMATA TUTOR is an online tool (`http://www.automatatutor.com`) that helps students learn basic concepts in theory of computation, such as finite automata and regular expressions. The tool provides personalized feedback when students submit incorrect solutions, and also helps teachers managing large classes by automatically grading homework assignments. The techniques we use to produce these grades and feedback messages are based on algorithmic techniques that are grounded in program synthesis, logic, and the theory of formal languages [2, 6]. Real courses at 14 different universities with more than 3,000 students in 4 different continents have already used AUTOMATA TUTOR.

In this paper we summarize our experience in building such a system. We start by describing the features AUTOMATA TUTOR offers to help both students and instructors (Sec. 2), and provide a brief history of how the tool became what it is now (Sec. 3). We then describe the experiments, surveys, and user studies that we ran to assess the overall user satisfaction and the quality of grades, feedback messages, and user interfaces (Sec. 4). Finally, we discuss what we learned from the data we collected, and from three years of experience building the tool (Sec. 5).

Other automata theory education tools There are several strategies for teaching automata and other formalisms in computer science education. Our system is the first online tool that is able to grade students' solutions and can provide actionable feedback rather than just counterexamples.

The other notable tools for teaching DFA constructions are JFLAP and Gradiance. JFLAP [13] allows students to author and simulate automata and is widely used in classrooms. Instructors can test student models against a set of input strings and expected results. Recently JFLAP has been equipped with an interface that allows students to test their solution DFA on problems for which they only have an English description of the language [14]. To do this the student writes an imperative program that matches the given language description. If this program is not equivalent to the student's DFA JFLAP automatically produces a counterexample. Gradiance[1] is a learning environment for database, programming and automata concepts. It focuses on providing tests based on multiple choice questions. These tools either do not support a way for drawing DFAs, or do not have a high-level representation of a problem and can therefore not provide grades and feedback about the conceptual problems with a student's submission.

Additional tools are available for problems related to DFA constructions. In ProofChecker [16], students prove the correctness of a DFA by labelling the lan-

[1] `http://www.newgradiance.com/`

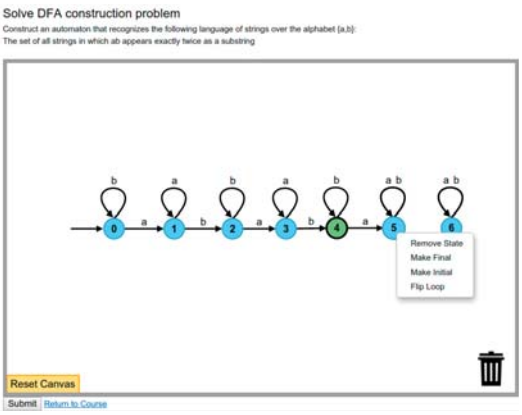

Figure 1: A student's view at an attempt at solving a DFA construction problem

guage described by each state: given a DFA the student enters "state conditions" (functions or regular expressions) describing the language of each individual state. The system then tests these conditions against a finite set of strings. In DeduceIt [7], students solve assignments on logical derivations. DeduceIt is able to then grade such assignments and provide incremental feedback. Visualizations such as animations of algorithms or depictions of transformations between automata and equivalent regular expressions exist [4]. These tools do not support course management, grading, and typically provide only counterexample based feedback. Moreover, the support for most of these tools has been discontinued and they are not available to the public anymore.

To the best of our knowledge, Automata Tutor is the first tool for teaching formal languages that includes course management, grading, and feedback generation. Moreover, the features of Automata Tutor have been thoroughly evaluated using multiple experiments and user studies [2, 6].

2 Automata Tutor in a nutshell

Automata Tutor is an online education tool created to help students learn basic concepts in theory of computation. In particular, it provides an interface for students to draw the corresponding DFA or NFA to a given description, and receive instantaneous feedback about their submission. The tool also supports regular expressions and NFA-to-DFA constructions. In this section we focus on how Automata Tutor is helpful to both teachers and their students.

2.1 A tool for students

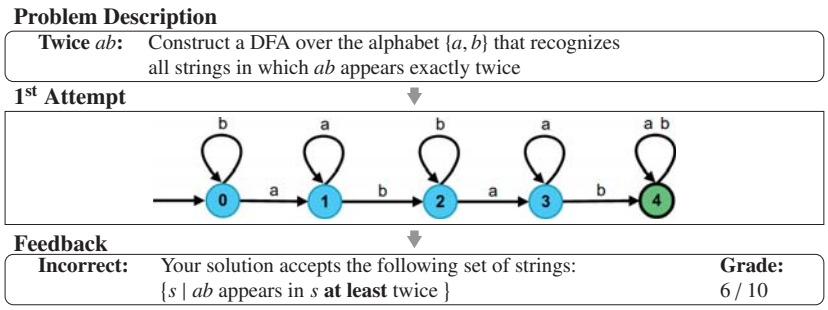

Figure 2: A student's first attempt at solving a given problem. The user receives personalized feedback.

Automata Tutor offers a structured, easy to use interface for students to practice drawing automata matching a particular description, as shown in Figure 1. While allowing students to quickly draw any automaton, the tool also enforces that all automata are legal, helping students better understand the concepts. For example, when a student adds a new node to a DFA, edges from the node for each symbol in the alphabet are added automatically and cannot be deleted.

Upon submitting an automaton, Automata Tutor provides students with instantaneous feedback to help them understand and fix their mistakes. Consider a student attempting to draw a DFA accepting the language of all strings in which "ab" appears exactly twice. If the student draws and submits an automaton that accepts a closely related language, such as the language of all strings in which

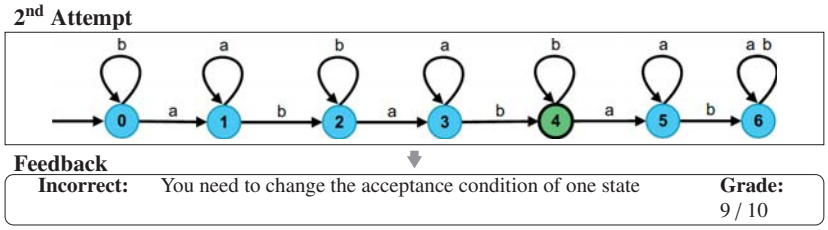

Figure 3: The student's second attempt at solving the problem. They are given a hint at how to change their automaton to the correct one.

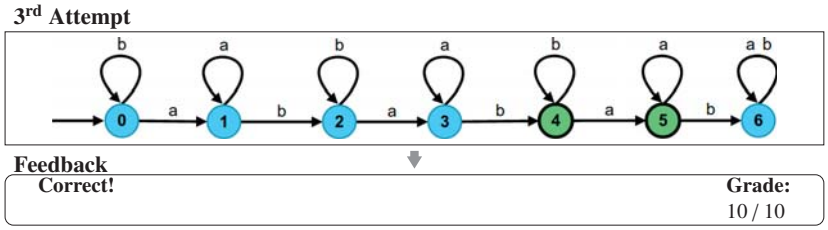

Figure 4: The student's final attempt at solving the problem. This attempt produces the correct automaton.

"ab" appears at least twice, the tool provides a hint describing the difference, as is done in Figure 2. After receiving the feedback message, the student can submit a new attempt (Figure 3). In this case, the student's submissions is structurally very similar to a correct automaton, and the tool suggests how the student might modify the automaton. If the student submits a correct automaton, as shown in Figure 4, the tool assigns full score. Lastly, the tool can also provide a counterexample string on which the student's automaton behaves incorrectly. [6] provides a thorough explanation about feedback for DFA construction problems.

For NFA construction problems, the tool provides counterexample feedback on incorrect automata. In addition, it gives students feedback if they did not take advantage of nondeterminism or if they have not given the minimal automaton, an example being "There exists an equivalent NFA with 1 fewer states and 2 fewer transitions."

2.2 A tool for teachers

Automata Tutor automates grading in a fair way that is comparable to grades assigned by a human grader, saving teachers time and energy. To calculate grades for a student's automaton, the tool estimates the percentage of strings on which the automaton fails and calculates the minimum number of syntactic edits the submission needs to accept the correct language. For DFA submissions, a distance is also calculated between the language of the student's automaton and the correct language. [2] describes the grading algorithm for DFAs in more detail. For NFA submissions accepting the correct language, the tool also takes into account how much larger the student's submission is than the solution.

The tool is also flexible, allowing teachers to specify their own assignments and assign them to their students. Instructors can create a course, and have their students register for it by distributing its course ID and password. They can then

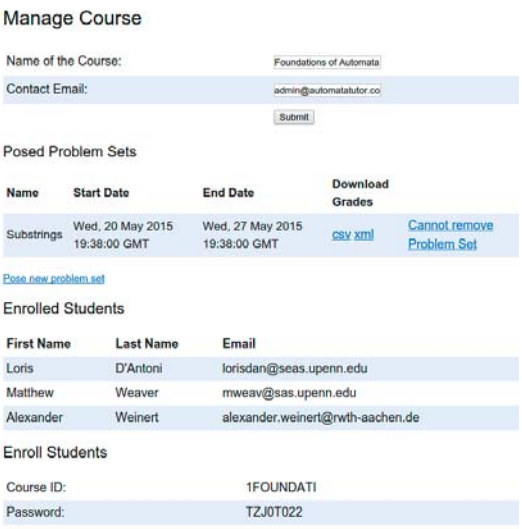

Figure 5: An instructor's view at one of their courses

create problem sets for the course with their own descriptions and solutions, and can choose to specify a limit to the number of attempts each student has for each problem. Ultimately, teachers can download the grade information for their students' submissions. The course management interface is shown in Figure 5.

3 The evolution of Automata Tutor

In its three years of existence, Automata Tutor has gone from a prototype, which was never used in real classrooms, to a tool that is widely accepted and used in multiple universities all over the world. We give an overview over the development history of the tool in this section and show the driving forces and ideas behind its development.

3.1 The precursor

A basic version of Automata Tutor was built to test the algorithms presented in [5]. This paper presented a logic for describing languages that could also be nonregular, and the algorithms presented in [5] could prove or disprove regularity for many such languages. In this version of Automata Tutor, an administrator

could use such a logic to pose automata constructions problems. Students could then solve the problems and receive a counterexample when the solutions they submitted was incorrect.

Limitations of the precursor Since the goal of the tool at this point was that of evaluating algorithms, the interface was still immature and drawing automata was cumbersome. Moreover, the feedback was limited to counterexamples. Due to these factors, this version of the tool was not deployed in real classes.

3.2 Automata Tutor 1.0

The precursor of Automata Tutor just provided students with a counterexample if their attempt at solving a problem was incorrect. Our goal in developing Automata Tutor 1.0 was to also provide students with a grade for their attempt as well as with feedback about how to improve and correct their automaton. In order to do so, the tool compared the student's attempt with the solution using three different metrics. These metrics are outlined in [2] and further discussed in Section 4.1. In addition to a numerical grade the tool provided actionable feedback expressed in plain English. We briefly described techniques used to produce the feedback in Section 2, and a study of the effectiveness of feedback messages is presented in [6].

Automata Tutor 1.0 only supported DFA construction problems. In this type of problem, the students are given a problem statement of the form "Construct a DFA that accepts all the strings in the language ..." The students could construct a DFA graphically in their browser and submit their attempt to the system. The system would then provide a grade and a feedback message.

In order to split the load as well as separate the concerns of displaying problems to the user and the actual computation, we split Automata Tutor 1.0 into two components: a web-facing frontend built in Scala and Lift, and a backend built using C# and ASP.NET. Another benefit of this architecture was that it would be possible to support other kinds of problems using the same frontend. Even though this did not happen in Automata Tutor 1.0, this architecture would prove very useful when building the successor, Automata Tutor 2.0. A typical communication between the frontend and the backend is shown in Fig. 6.

Limitations of Automata Tutor 1.0 Although Automata Tutor 1.0 served its purpose of evaluating the techniques from [2] very well, there were still some limitations, some of which were technical, while others were conceptual. The technical problems included the fact that the tool relied on features that were only supported by the Google Chrome browser. Additionally, although the interface

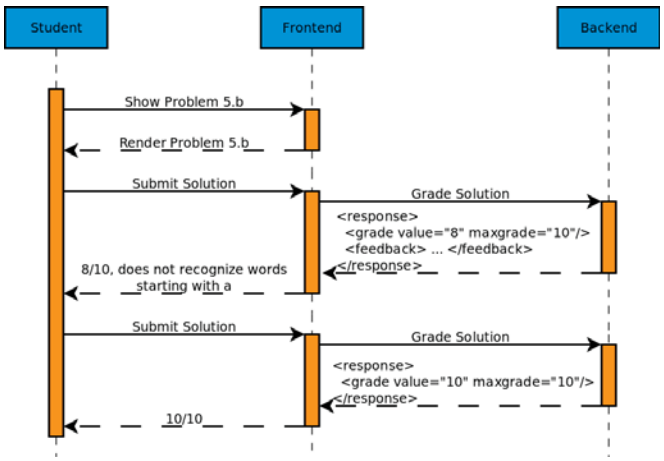

Figure 6: Typical workflow of a student solving a problem

for students to attempt the posed problems was improved from the one present in the previous version of the tool, it was still inconvenient to use.

Chief among the conceptual shortcomings was the fact that DFA construction problems were the only kind of supported problems. Whereas the separation of front- and backend already laid the foundation for the extension to other kinds of problems, it would have been non-trivial to extend the frontend to handle these new problem types. Also, using the tool required strong collaboration between instructors and administrators, as there was no way to create courses, and assign problems only to certain groups of students. These limitations were adressed during the development of Automata Tutor 2.0.

3.3 Automata Tutor 2.0

After we deployed and successfully used Automata Tutor 1.0 in 3 courses at 3 universities with about 400 students, the tool's limitations became apparent. We decided to reimplement the frontend with a heavier focus on scalability in terms of involvement of the administrators, and flexibility in terms of creating different types of problems.

We addressed the former problem by implementing course management and assigning users one of two roles: students, instructors. The frontend now has a concept of courses, problems and problem sets. While students can only enroll in courses and solve the problems posed in these courses, instructors are allowed

to create courses and problems, and collect grades at the end of a course. A screenshot of the interface for course management can be seen in Fig. 5.

AUTOMATA TUTOR 2.0 also allows for easy extensibility to handle new problem types. A developer simply has to provide views that allow instructors to create, edit and delete a problem as well as a view that allows a student to solve a problem. Furthermore, the developer will have to implement a web-service offering a grading engine.

We implemented three new problem types in addition to the already existing DFA construction problems using this abstraction. Version 2.0 of AUTOMATA TUTOR thus supports the construction of DFAs, NFAs, and regular expressions from a description of a regular language in plain English, as well as the construction of a DFA that is equivalent to a given NFA. Since all these formalisms describe the set of regular languages, we could reuse parts of the existing grading and the feedback engines.

These changes were widely accepted and are now in constant use. Instructors deeply appreciated the possibility to manage their courses themselves instead of having to work with the administrators. We rolled the new frontend out at 14 universities in four continents. It is used by 3,000 students and has already graded more than 40,000 solutions.

Limitations of AUTOMATA TUTOR 2.0 Although the tool is being appreciated by the community, there is still room for many improvements. In particular, in its current version, the tool only supports DFA, NFA, and regular expressions problems. We discuss some future directions in Sec. 5.

4 Experience report

Throughout the process of developing AUTOMATA TUTOR, we have conducted a number of a studies to test the tool's success at helping students learn to construct automata. We summarize what learned from them in this section.

4.1 Results about automatic grading

Through a number of user studies involving over 500 students at three universities[2], we used students' responses on a 5 point Likert scale to measure

- how fair students feel the grades assigned by the tool are;
- how meaningful the grades assigned by the tool are.

[2] University of Illinois Urbana-Champaign, University of Pennsylvania, Reykjavik University

The general consensus is that students find that the partial grades assigned by the tool are both fair and meaningful. For DFA submissions, we additionally compared grades assigned by the tool to those given by human graders, and found they are comparable, although the tool is more consistent at assigning the same grade to the same solution. [2] contains a more detailed discussion on automatic grading in Automata Tutor.

4.2 Results about feedback

The user studies all agreed that, when it comes to feedback, simpler is better. We measured this by splitting students into three groups each receiving a different type of feedback: binary feedback (yes/no), counterexamples, and plain English hints. Afterwards, we had the students fill out a survey with a 5 point Likert scale asking about

- how useful is the feedback;
- how helpful is the feedback for understanding mistakes;
- how helpful is the feedback for getting the correct solution;
- how confusing is the feedback.

While too much feedback may be detrimental to students, having no feedback is worse; students who were only told if their submission was correct or not were slower at solving problems and did fewer practice problems than those receiving feedback. [6] provides a comprehensive report on this user study.

For feedback pertaining the size of a student's NFA, there was no improvement in performance from sharing how many fewer states and transitions were in the solution than just showing "A smaller NFA exists." Interestingly, in both cases, about half of students' subsequent submission no longer accepted the correct language.

4.3 Results about usability

We compared the user surveys with the old (1.0) and new (2.0) drawing interfaces and measured

- how easy students thought it was to draw using the interface;
- how predictable the behavior of the interface was.

We used a 5 point Likert scale for each metric, concluding that the new interface is significantly easier to use, and is significantly better at behaving as users expected.To both questions, the median student response for the new interface was a 5: the highest score. This is particularly meaningful, as a tutorial accompanied the old interface while no instructions are provided for the new interface.

4.4 What we learned from the instructors

We summarize the key (subjective) observations by the three instructors who have taught theory of computation courses multiple times before and have used Automata Tutor during our experiments. First, the requirement that the homework had to be submitted using the tutoring tool ensured students' participation. Once students started interacting with the software, they were very much engaged with the course material. Second, the average grade on the homework assignments offered in the course increased when using Automata Tutor. Third, the teaching assistants were very happy that the tool did the grading for them. Lastly, while not profound, we learned that instructors enjoy the tool and want more of this type of work: "This is how the construction of finite automata that recognize regular languages should be taught in a modern way! I wish I had similar tools for all the topics I need to cover" [1].

5 Discussion and future work

In its three years of life Automata Tutor has seen many updates and was adopted by more than 3,000 users. In this section we discuss the lessons we learned from our experiments and show how these lessons can be applied to further improving the tool. We first present what features were well-received by instructors and students before we discuss features that were not well-received and therefore removed from the tool. Finally, we show how we plan to extend Automata Tutor to support new problems and attract more users in the future.

5.1 What worked

Based on our surveys and experiments we observed the following:

Interfaces are important: Although this is known for many other domains, it is particularly important in the context of education. When students are already struggling to find the right way to approach an homework problem, a non-intuitive drawing interface can cause harm.

Simple feedback is good enough: Based on the experiments discussed in Section 4 and in [6], it is clear that almost any type of feedback is effective and improves the students' learning experience. The feedback messages have to be clear and concise: counterexamples, simple edits, etc.

Instructors like independence: In the earlier versions we allowed instructors to contact us to set up courses, which was too complicated for them. Enabling course management in Automata Tutor 2.0 allowed us to gain a large user base. Since the introduction of course management, 10 new universities started using the tool in real classes.

Instructors love automated grading: This is not surprising, but it is probably the feature that is most responsible for the success of the tool. Automata Tutor has been deployed in classes with more than 200 students for which manually grading an homework would take more than fifty man-hours. In Luca Aceto's words: *"From my perspective (and from that of my TAs), automatic grading is a real bonus. I love to teach, but I really hate to grade a large number of student assignments."* Up to today Automata Tutor has graded more than 50,000 student submission;

End of course surveys: These are really helpful in assessing the features that cause confusion and those that actually help the students. Quantitative questions which ask for overall user satisfaction are helpful in assessing the value of the tool. Open ended questions which ask for suggestions and opinions can guide on what features should be added or removed.

5.2 What did not work

Based on our surveys and experiments we observed that:

Verbose feedback is confusing: During the user study presented in [6] we found that longer feedback messages were confusing and were actually causing frustration among the students. We therefore removed verbose feedback messages and replaced them with counterexamples.

Long solution-oriented feedback: If the solution is far from a correct one it is better to simply tell the student that the solution is incorrect rather than providing a hint on how to fix it. In particular we observed that long edit scripts are confusing.

A single crash can cost many users: Especially in the domain of education, where student rarely do homework assignments, it is important to provide a robust tool on a robust server.

5.3 The final goal

In the next years we want to add more features to Automata Tutor and be able to fully support students learning basic theory of computation concepts. Concretely we plan to add:

Regular expressions: Although the tool supports them, the grading features are currently very basic. Defining new metrics and feedback that are tailored for regular expressions is part of our agenda.

Automata meta-constructions: An example problem would be: *Given two DFAs* $(Q_1, q_0^1, \delta_1, F_1)$ *and* $(Q_2, q_0^2, \delta_2, F_2)$ *define their intersection.* Such a feature requires a language that is able to symbolically manipulate the objects in the two automata signatures. Integrating such a language with a theorem prover, like Coq, might allow us to effectively grade these complex assignments.

Proofs of non-regularity: These are an important concept on which students often struggle. Some initial attempts at this problem can be found in [5], but it is still not clear how to build a user-interaction model that can produce feedback and grades, in particular for proofs based on the pumping lemma.

Proof of DFA correctness: Students need to know how to characterize the languages described by each state of an automaton in order to prove by induction that the DFA correctly accepts a target language. The logic presented in [2] could be used to allow the students to enter such descriptions. A informal attempt to solve this problem is presented in [16].

Context-free languages: It is not clear how to adapt our current methods for grading and feedback to non-regular languages. Even though equivalence of these languages is undecidable, there may be algorithms that work for small solutions.

Turing machines: Adding grading for Turing machines would be the next sttep after the grading of context-free languages. This poses the same questions as the previous case.

MOOC deployment: We would like to deploy Automata Tutor in a real MOOC. This would allow us to leverage a large user base and learn more about the tool capabilities from the MOOC's forum.

6 Conclusions

We presented Automata Tutor, an online tool that is already being used by 13 universities around the world to teach basic concepts in theory of computation.

AUTOMATA TUTOR is available at `http://www.automatatutor.com`. It allows instructors to manage courses, and it can currently provide the students with grades and personalized feedback for DFA, NFA, NFA-to-DFA, and regular expressions constructions. We discussed how the tool evolved and pointed out the driving features behind its success: simple and clear feedback messages, consistent grades, intuitive drawing interface, and course management for instructors. Our ultimate goal is to extend AUTOMATA TUTOR to support most undergraduate-level concepts in theory of computation such as proofs of non-regularity, automata meta-constructions, and context-free grammars.

Acknowledgements We would like to thank Luca Aceto for his invaluable support, feedback, and availability. Not only Luca helped us improving the tool, but he also advertised it to the community. This research is partially supported by NSF Expeditions in Computing awards CCF 1138996.

References

[1] Luca Aceto. Ode to the automata tutor, 2015.

[2] Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of dfa constructions. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1976–1982. AAAI Press, 2013.

[3] John R. Anderson and Brian J. Reiser. The LISP tutor: it approaches the effectiveness of a human tutor. *BYTE*, 10(4):159–175, April 1985.

[4] Beatrix Braune, Stephan Diehl, Andreas Kerren, and Reinhard Wilhelm. Animation of the generation and computation of finite automata for learning software. In Oliver Boldt and Helmut JÃijrgensen, editors, *Automata Implementation*, number 2214 in Lecture Notes in Computer Science, pages 39–47. Springer Berlin Heidelberg, January 2001.

[5] Pavol Cernỳ, Sumit Gulwani, Thomas A Henzinger, Arjun Radhakrishna, and Damien Zufferey. Specification, verification and synthesis for automata problems. 2012.

[6] Loris D'antoni, Dileep Kini, Rajeev Alur, Sumit Gulwani, Mahesh Viswanathan, and Björn Hartmann. How can automatic feedback help students construct automata? *ACM Trans. Comput.-Hum. Interact.*, 22(2):9:1–9:24, March 2015.

[7] Ethan Fast, Colleen Lee, Alex Aiken, Michael Bernstein, Daphne Koller, and Eric Smith. Crowd-scale interactive formal reasoning and analytics. In *Proceedings of UIST'13*, 2013.

[8] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari. Synthesizing geometry constructions. *SIGPLAN Not.*, 46(6):50–61, June 2011.

[9] Shachar Itzhaky, Sumit Gulwani, Neil Immerman, and Mooly Sagiv. Solving geometry problems using a combination of symbolic and numerical reasoning. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *Lecture Notes in Computer Science*, pages 457–472. Springer Berlin Heidelberg, 2013.

[10] Katy Jordan. Mooc completion rates: The data. 2014.

[11] Antonija Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of SIGCSE'98*, pages 307–311, New York, NY, USA, 1998. ACM.

[12] David Patterson. Why are english majors studying computer science? 2013.

[13] Susan H. Rodger and Thomas Finley. *JFLAP - An Interactive Formal Languages and Automata Package*. Jones and Bartlett, 2006.

[14] V.S. Shekhar, A. Agarwalla, A. Agarwal, B. Nitish, and V. Kumar. Enhancing JFLAP with automata construction problems and automated feedback. In *Contemporary Computing (IC3), 2014 Seventh International Conference on*, pages 19–23, Aug 2014.

[15] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of PLDI'13*, pages 15–26, New York, NY, USA, 2013. ACM.

[16] Matthias F. Stallmann, Suzanne P. Balik, Robert D. Rodman, Sina Bahram, Michael C. Grace, and Susan D. High. Proofchecker: an accessible environment for automata theory correctness proofs. *SIGCSE Bull.*, 39(3):48–52, June 2007.

[17] New York Times. Instruction for masses knocks down campus walls. 2012.