Syrus: Providing Practice Problems in Discrete Mathematics With Instant Feedback

Diego Zaccai[1] Aditi Tagore[1] Dustin Hoffman[1] Jason Kirschenbaum[1]
Zakariya Bainazarov[1] Harvey M. Friedman[2] Dennis K. Pearl[3]
Bruce W. Weide[1]

[1]Dept. of Computer Science and Engineering [2]Dept. of Mathematics [3]Dept. of Statistics
The Ohio State University
Columbus OH 43210, USA
(zaccai.1, tagore.2, hoffman.373, kirschenbaum.9,
bainazarov.2, friedman.8, pearl.1, weide.1)@osu.edu

ABSTRACT

Syrus is courseware designed with the goal of helping students better understand logical sentences involving quantifiers. *Syrus* uses template-guided mutation of "seed" formulas to generate candidate practice problems, and third-party theorem-provers to automatically determine the truth value of each. It provides students with a virtually unlimited supply of unique and relevant practice problems and provides immediate feedback on each problem. Results of an empirical study of its efficacy are reported.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Learning—*Induction, Concept learning*; K.3.2 [**Computers and Education**]: Computer and Infomation Science Education—*Computer science education, Self-assessment*

General Terms

Human Factors, Theory

Keywords

Interactive Learning, Mathematical Theories in CS, On-line Education, Discrete Mathematics, Predicate Calculus

1. INTRODUCTION

For nearly every major topic in an elementary science or engineering course, student assignments include many practice problems, all of a similar nature. Evidently instructors believe that, as the saying goes, "practice makes perfect" because it helps students hone their skills in solving problems of a certain kind.

SIGCSE'14, March 5–8, 2014, Atlanta, GA, USA.

http://dx.doi.org/10.1145/2538862.2538929 .

With the advent of on-line courses and many universities' rush toward on-line education, the traditional constructs of education such as classrooms and homework are being challenged. Without the constraint of in-person instruction, classes become bigger and students more remote. In this context, traditional pedagogical tools such as graded homework assignments become either impossible or prohibitively expensive. Can software produce good practice problems? Can it solve them? Can students learn this way?

Syrus seeks to address such questions in one disciplinary area (formal logic, specifically predicate calculus) by adapting existing powerful research software (automated theorem-provers) to generate and solve many practice problems of pedagogical value.

2. EDUCATIONAL GOALS

The key learning outcome targeted by *Syrus* is that students should be able to understand and reason rigorously with logical statements in predicate calculus over various mathematical domains of interest in CS. The approach helps novices develop more expert-like skills through effortful practice [2]. The generic inspiration for building and evaluating such a tool is based on the following observations:

1. Introductory courses in many STEM areas involve students practicing on many homework problems.
2. Typical human grading of such assignments severely limits the number of such problems that can be attempted by students with useful and timely feedback.
3. State-of-the-art tools used by researchers in the field can sometimes solve student problems automatically.
4. Large numbers of problems with similar (pedagogically useful) structure can be generated automatically.

Before we describe *Syrus* and its evaluation, some history is in order. The original idea was that the student is given a sentence in predicate calculus over some rich collection of mathematical domains (integers, reals, sets, strings, etc.), is asked whether it is valid, and is then challenged to *prove* that claim. We discovered there are two basic dimensions to the "space" occupied by any tool that attempts to accomplish this. One is the depth of interaction between the software and the student: how much feedback is provided in real-time as the student works her way through the proof? A tool with deep interaction needs to check proof steps proposed by the

student and advise about possible next steps. The other dimension is the domain of discourse: over what mathematical domains are the sentences that constitute the practice problems? A tool should help the student learn to understand sentences over diverse mathematical domains, not simply (for example) statements about a very constrained domain such as integer arithmetic modulo 2.

One might initially prefer deep interaction on problems over multiple rich mathematical domains. Yet it turns out this is impractical at present because of both technological shortcomings in automated theorem-provers and open theoretical issues in formal logic (details of which are beyond the scope of this paper). In short, it is arguably feasible to move significantly along one dimension but not both. We first built a prototype tool supporting deep interaction on proof development for logical sentences over a very limited mathematical domain involving integers with addition of constants and relational operators. Some new theory to support this was developed by one of the authors (Friedman). There were two problems. First, more new theory is required to suitably constrain other interesting mathematical domains to support proof strategies for students and permit deep interaction with the tool. Second, and more urgently, it became evident that students had to improve their reading and understanding of logical sentences with quantifiers. Simply deciding whether a fairly short statement in predicate calculus is valid turns out to be a difficult enough initial task for learners.

So, we next built a second prototype in which that *was* the student's problem: decide whether a given sentence is valid. Now, the sentences are over the much broader and more interesting collection of mathematical domains mentioned above, but with no deep interaction. The student says "true" or "false" for each sentence and the tool reports whether she is correct. The *Syrus* tool described in this paper generates a large number of sentences in predicate calculus over some key mathematical theories of interest in a discrete math course for CS majors; it solves these problems (determines the validity of these sentences) automatically using automated theorem-proving tools, so that the answers are known to the system; it issues these sentences as true/false problems; and it collects data to be used to evaluate the extent to which learning outcomes are achieved (as related to various measures of student engagement with, and success in solving, these practice problems).

Syrus contributes to STEM education by demonstrating the potential for using research tools to create related pedagogical tools and practice problems, as well as the difficulties faced when attempting to do so. The techniques used in *Syrus* suggest that it is likely feasible to adapt research tools for education in areas other than mathematics. The reason for optimism is that problems that challenge novice learners remain basically static even as the capabilities of research tools to solve such problems (automatically) continues to improve.

3. INTERACTING WITH *SYRUS*

Once a student logs in to the *Syrus* system using an email address, she is presented with a random problem, i.e., a mathematical sentence, from the currently enabled problem set. When a problem is displayed, the student is asked whether the mathematical sentence is true. The student must determine this and answer by pressing either the **True** or **False** button below it. Figure 1 shows a sample question.

After selecting an answer, the student gets instant feedback from *Syrus* on whether the answer is correct, and may move to the next problem. Additionally, via the **My Account** button on the left, the website allows the student to track how many problems have been

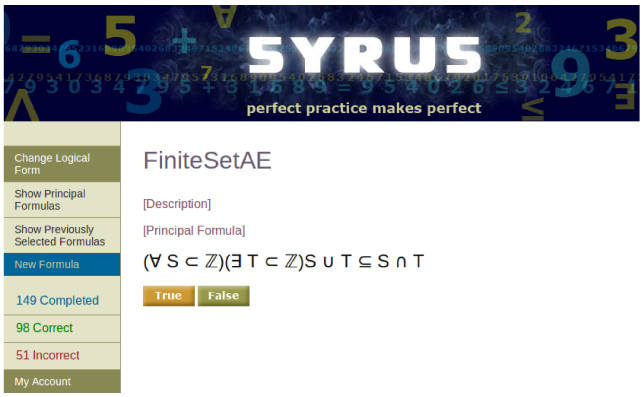

Figure 1: A sample problem in the *Syrus* webpage

completed, how many were answered correctly, and the average time spent on each problem.

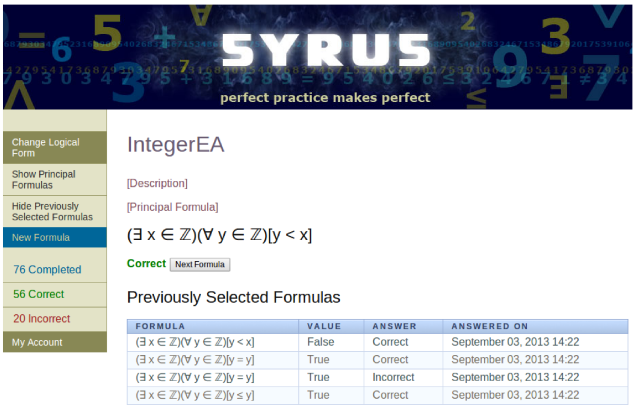

Figure 2: A sample list of previously solved problems in the *Syrus* web page

While working a problem set, a student can review past performance by selecting the **Show Previously Selected Formulas** button. This provides a list of all the problems completed since the start of the current problem set, as well as the selected answer, and whether the response was correct. Figure 2 shows a sample list of previously answered problems.

To view the problems that inspired the currently selected problem set, a student can select the **Show Principal Formulas** button on the left. This provides a listing of the seed formulas from which the current problem set was generated, as shown in Figure 3. Having access to the seed formulas allows a student to ask questions about a group of problems in a class discussion. This is particularly important, since the large number of sentences makes it unlikely that many students will encounter the same problem. Furthermore, since the discussion is based on the seed problem, the techniques discussed in class are likely to remain relevant for all the problems generated from it.

Once a student feels confident in her ability to solve problems of the current form, she can select the **Change Logical Form** button to move toward more complex problems. This button allows the student to increase the difficulty by selecting a problem set within

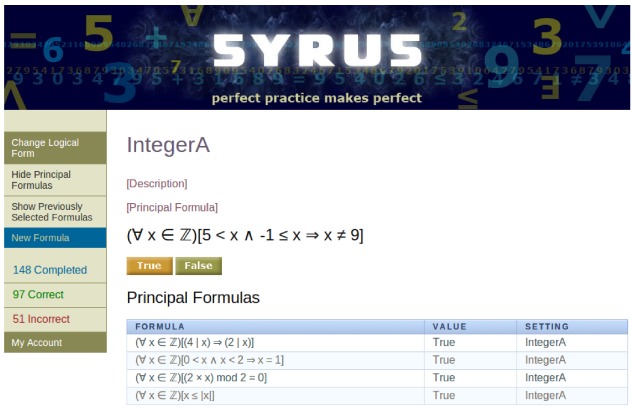

Figure 3: A display of the seed formulas used to generate a problem set

the same class of problems but with a higher number of quantifiers, or to move on to a completely different type of problem.

After deciding to change the logical form, the student is presented with a list that is categorized by mathematical theories with sub-categories represented by the quantifier structure of the formulas. For example, in Figure 1, `FiniteSetAE` shows that the current problem set involves finite sets and that the sentences contain variables that are universally quantified followed by variables that are existentially quantified. Figure 4 shows different problem sets involving integers. (Such details in the user interface are incidentally part of *Syrus* now, but are not essential to the idea).

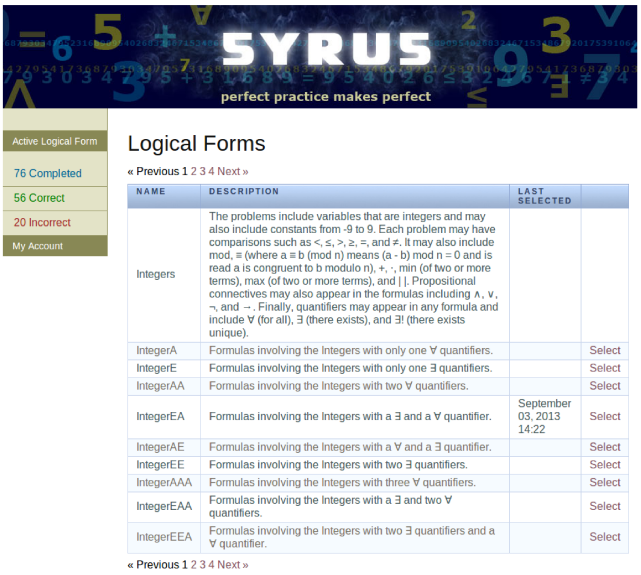

Figure 4: Various problem structures in the *Syrus* webpage

In order to provide a seemingly endless number of problems, *Syrus* has a large number of sentences ranging over different mathematical theories relevant to CS students including integers, reals, sets and strings. The problem sets are first organized by the type of problems. For example, one problem set includes problems about finite sets and their basic operators with a single universal quantifier, while another includes problems dealing with Presburger arithmetic and two alternating quantifiers. Inside each of those problem types, further sub-categories allow students to select an increasing number of quantifiers for each. Sentences (1)-(3) illustrate the different kinds of mathematical theories built into *Syrus*.

Sentence (1) is a sample problem involving integers. *Syrus* allows the comparison of numerical types. It can also handle addition, subtraction, multiplication, modulus, and divisibility of integers. The max function evaluates to the larger of its two integer arguments; the min function is similar. Sentence (1) is false.

$$(\forall x \in \mathbb{Z})(\exists y \in \mathbb{Z})[y = y - \max(7, x)] \quad (1)$$

Sentence (2) is a problem involving real numbers. Real numbers can be added, subtracted, and multiplied. The absolute value function is also available. Sentence (2) is true.

$$(\exists x, y \in \mathbb{R})[|x - y| = |x| + |y|] \quad (2)$$

Sentence (3) involves strings (represented by the Kleene star) over the alphabet of integers. In terms of comparison of strings, only equality and non-equality are allowed. String functions include concatenation, represented by the $\circ$ operator, and length, represented by $|\bullet|$. The reverse function evaluates to the reverse of a string, i.e., its entries in the opposite order. A substring function is also provided. Sentence (3) is an example of a problem involving both string concatenations and their reverse. Sentence (3) is true.

$$(\forall s, t \in \mathbb{Z}^*)[\text{reverse}(s \circ t) = \text{reverse}(t) \circ \text{reverse}(s)] \quad (3)$$

Problems involving finite sets are also available in *Syrus*. These kinds of sentences and the possible operators on sets are described in detail in Section 4.

4. DESIGN

The *Syrus* courseware is delivered to students via the web. Behind the scenes is an integrated tool-chain shown in Figure 5.

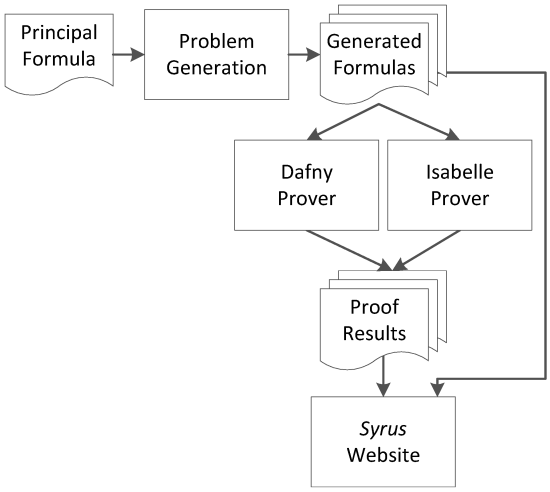

Figure 5: *Syrus* toolchain.

The first component is the problem generation tool. Ideally, every problem presented to a student would be of intellectual and pedagogical value. The easiest way to ensure this would be to present students only with problems carefully designed by an instructor or other expert. However, that process would be prohibitively expensive for a domain as rich as predicate calculus over multiple theories relevant to CS students. We would like thousands of problems

in each problem set to ensure that students have the necessary volume of practice problems. There should ideally be a substantially large number of problems in each problem set to allow students to practice without running into the same problem multiple times. To achieve this, a problem generator takes a *principal formula* as input and generates multiple variant problems from it. A principal formula may be taken from a textbook or otherwise created manually by an instructor.

After the creation of a candidate problem set, each of the problems in the set must be answered. Given that a problem set typically contains a large number of problems, we turn to off-the-shelf theorem provers to perform the task of determining the validity of each candidate problem sentence. Thus, the second component consists of third-party theorem provers that are used to automatically determine the correct answer for each generated problem.

The last major piece is the website that serves as an interface for the students. The website takes the problems generated by the first component, along with the results of the automated answering process, and presents them to students. The following subsections describe the design and development of each of these components.

4.1 Problem Generation and Proof

Randomly generating arbitrary mathematical sentences would provide us with a very large problem set, but at the cost of quality. While randomly generated (but syntactically correct) sentences might be proven true or false, their intellectual and pedagogical value would be questionable. Sentences that are either too short-/easy or too long/difficult or just nonsensical semantically are bound to be created by a random process, and the labor-intensive job of filtering uninteresting problems would again fall to an instructor.

As a result, our problem generator uses a hybrid approach. Instructors provide *Syrus* with mathematical sentences that are considered to be interesting for the subject matter being taught, be it because they are important lemmas or because students in previous courses have had difficulties understanding sentences of a similar form. These sentences become principal formulas. The tool-set generates mutations of them using a syntactic approach. For example, students might be expected to understand that a non-empty set of integers contains an integer. This principal formula is sentence (4).

$$(\forall S \subset \mathbb{Z})[\phi \subset S \implies (\exists x \in \mathbb{Z})[x \in S]] \quad (4)$$

Similarly, instructors using the tools have noticed that students have special difficulty reasoning about long chains of operations. We therefore introduced principal formulas that contain two relatively long sequences of operations, requiring students to simplify such chains of operations to determine how the resulting sets are related to one another. An example of such a principal formula is (5).

$$(\forall S, T \subset \mathbb{Z})[(\exists R \subset \mathbb{Z})[((S-T) \cup R) \cap S = (S \cup T) \cap R]] \quad (5)$$

This formula is not true. However, this does not pose a problem because the truth values of the formulas generated from it are determined by theorem provers independently of the principal formula.

New candidate problems are generated by replacing some operators and variables in the principal formula in a manner that preserves its overall structure. The mutation of the sentence allows variables and constants to be interchanged with other variables that have already been bound by a quantifier and are of the same type, as well as constant values. These variations can be limited in various ways, e.g., to a fixed number of occurrences per generated formula, to ensure that the problem set does not deviate too significantly from its seed principal formula.

$$(\exists S \subset \mathbb{Z})[\phi \neq S \iff (\forall x \in \mathbb{Z})[x \notin S]]$$

$$(\exists S \subset \mathbb{Z})[\neg((\exists x \in \mathbb{Z})[x \notin S] \vee \phi \neq S)]$$

$$(\forall S \subset \mathbb{Z})[(\forall x \in \mathbb{Z})[x \in S] \vee \phi \subseteq S]$$

Figure 6: Some of the resulting variations of sentence (4)

Similarly, logical connectives such as $\wedge$, $\vee$, $\implies$ and $\iff$ can be interchanged with one another. When these exchanges occur, the mutation ensures that the operation types do not change (thus the sentences remain syntactically correct). For example, for numeric types the operators $+$, $-$, $*$, $\max$, $\min$ may be interchanged within the formula. Similarly for sets, $\cup$, $\cap$, and $-$ (set difference) may be replaced by one another. Finally, mutations on the quantifiers are also permitted, with $\forall$, $\exists$ and $\exists!$ as possible choices. Some of the variants of the principal formula in sentence (4) are shown in Figure 6.

The Isabelle [7] or Z3 [1] theorem prover is used to check the validity of each candidate problem. We use Isabelle to check the validity of problems involving integers and real numbers, and Z3 to check problems involving sets and strings. When interfacing with Z3, the problems, along with the theories and mathematical definitions used in them, are expressed as assumptions and assertions in the Dafny [5, 3] programming language. Dafny is a programming language for software verification that uses Z3 as its back-end prover. We chose to use Dafny as the front-end to Z3 because it provides an easier route for translating statements about strings and sets than interacting with Z3 directly. Dafny translates these statements into the Boogie [4] intermediate language, which in turn generates verification conditions to be proved by Z3.

When adapting a problem for processing by Isabelle, the formula statement is translated in a manner similar to the Dafny translation previously discussed. The primary difference is the translation target language. Specifically, the Isabelle proof environment is built on a meta-logic framework (called Pure) with object logics (such as first-order logic) layered on top. Pure is a natural deduction system that includes rules for handling quantifiers, propositional connectives (negation, and, or, and implies), as well as unification. An object logic includes the syntax to express formulas in it, as well as the axioms and proof rules/theorems required to reason in it.

Each generated problem is initially represented in an XML format. We have built translation tools for Isabelle and Dafny/Z3 that convert from our XML format into their respective input formats.

4.2 Website

The *Syrus* website is implemented using Ruby on Rails. It serves as the only client-facing component of the courseware (the generation and proof process previously described are performed off-line). In addition to allowing students to view and answer questions, it also tracks statistics on the progress of each student. Problem sets containing sentences and their truth values are loaded into the website in bulk, formatted according to the notation used in the course, and then stored in a database.

The statistics tracked by the website are not only useful to the students, but also to instructors and researchers. For example, the detailed statistics allow us, as researchers, to gather data on the effectiveness of the product. Those same statistics, when viewed in the scope of particular problems, allow the course instructors to determine what types of problems students need additional help on. Also, the more limited information such as success rate and average

response time serve to guide the students and motivate them in a similar way to points in a game.

5. IMPACT AND EVALUATION

During the 2010-2011 academic year the *Syrus* software was developed and tested by students taking a discrete mathematics course, which has computer science students as its primary audience. During this period, data collection methods were also tested, and a short assessment of the skills that *Syrus* focused on was created by the course coordinator. This assessment was embedded in the course's final exam to allow for one method of project evaluation.

Complicating the study was the Summer 2012 change of the university calendar from quarters to semesters, and a change of the former math course to a new computer science course. Fortunately, we were able to work around these problems, but further studies would be complicated by the need for new baseline data. *Syrus* is currently being used in some sections of the new computer science foundations course.

In the Winter and Spring 2012 terms, the assessment was embedded in the course's final exam and scored consistently using a rubric designed by the course coordinator. There were two sections in each of these two terms. In Winter, neither section used *Syrus*. In Spring, the instructor of one section strongly suggested that students study using *Syrus* and one did not mention it. This allowed us to judge the effect of *Syrus* (Spring comparison) versus the typical degree of section-to-section differences without *Syrus* (Winter comparison).

The data collected was merged from three sources. The university's Student Information System was queried to get student age, ethnicity, gender, grade in the course, GPA in other Math courses, overall GPA, and Math Placement scores. Secondly, the *Syrus* system itself was queried to get information on the number of questions attempted, correctly answered, incorrectly answered, and skipped, together with the number of seconds taken in each of these activities. Finally, the course professors provided the scores on the embedded final exam question (20 points maximum) for each student in the four sections taking the final.

Section	Students	Used *Syrus*	Embedded Question	Course Grade - Math GPA
WI 1	65	No	15 (13 to 16)	-0.04 (-0.4 to 0.4)
WI 2	57	No	15 (13 to 16)	0.10 (-0.50 to 0.73)
SP 1	51	Yes	16 (12 to 18)	0.05 (-0.40 to 0.70)
SP 2	57	No	15 (11 to 19.25)	-0.17 (-1.20 to 0.45)

Table 1: Comparison of courses using *Syrus* with those that don't. The numerical results are presented with the median on top and the 25th and 75th percentile in parentheses.

Two principal response variables were studied: the grade on the embedded final exam question, and the difference between the student's grade in the course and their GPA in previous Math courses. This second response variable thus measured student performance in the logic course normalized by their expected grade given their performance in previous courses. A general overview of the results is presented in Table 1.

The students in the section asked to use *Syrus* as a study aid had a median score one point higher than the other three sections studied in the embedded final exam question. However the distribution of scores overlapped, and this difference may easily have been explained by random chance. The students in the Spring section that used *Syrus* also did better in the Math course relative to their previous math grades when compared to students in the other sections. Due to some skewness in the distributions, the change in the means was greater than the change in the medians, moving from -0.33 to $+0.11$. This difference was not easily explained by random chance ($p \approx 0.05$).

The fact that only small differences are seen in student learning as a result of using *Syrus* is probably to be expected since its use was not required by the course: even in the section where it was highly recommended for study, no grade was directly given for its use. As a result the median usage was a little under 10 minutes total, though some students did use it more. Only six students used *Syrus* for more than an hour total (spread over multiple sessions). This small group had a median score of 17.5 on the embedded question and had a median of 0.43 on the course grade minus previous math GPA.

The improved grades in the course compared to previous math courses for the section that used *Syrus* was seen across both sexes consistently. For males the median went from -0.51 in the *Syrus* section to -0.76 in the non-*Syrus* section; while for females the median went from -0.31 to -0.59. The improved grades in the discrete logic course compared to previous math courses for the section that used *Syrus* was also seen in both minority and white students. For the white students the median went from -0.51 in the *Syrus* section to -1.02 in the non-*Syrus* section; while for the minority students the median went from -0.46 to -0.60.

A statistical model controlling for gender, minority status, math placement scores and age did not change the significance of the difference between the Spring *Syrus* section and the non-*Syrus* section for the grades-based measure (neither did it change the lack of a significant difference in the scoring of the embedded question).

In the section using *Syrus*, the percent of the total time using *Syrus* that resulted in a correct answer to a *Syrus* question was correlated with both response variables. The rank correlation with the grade-based response was 0.40, while the rank correlation with the embedded question was 0.52. This is to be expected since all three measures gauge the degree to which the students learned core material in the class, but it does reinforce the underlying assumption that success in using *Syrus* is correlated with success in the class and in learning the material at hand.

The *Syrus* project showed some promise in improved outcomes in the discrete mathematics course, when adjusted for expectations based on grades in previous math courses. This result was consistent among different subpopulations. However, no effect of the use of *Syrus* was seen in a question embedded in the final exam that was specifically designed to gauge the knowledge that *Syrus* most directly covered. Further study in an experimental setting with heavy use of *Syrus* would seem to be needed to better understand its value.

6. DISCUSSIONS AND RELATED WORK

The disparity in the number of sentences that are determined either true or false for a given quantifier structure makes it impractical to randomly select a problem from the current logical form. For example, a student might realize that the percentage of true sentences in a setting that is universally quantified is larger that that of the false ones. This would improve the student's odds of getting a

correct answer by simply guessing true all the time. To avoid this problem, our website first decides whether it will provide a student with a true or false sentence, and then selects a sentence from the setting that matches the intended answer. This way the distribution of results seen by the student is independent of that of the proving mechanisms and the chances of a student getting a correct answer through guesswork is always 50%.

In terms of related educational software, the works of Suppes, *et al.*[9], at Stanford, and Sieg, *et al.* [8], at CMU, stand out as the best developed and the most closely related to *Syrus*. They are also among the very few such tools to have been evaluated with student subjects.

The CMU group observed [8] that resolution theorem provers gave little help to students in thinking about how to prove something, and hence set out to develop a proof search system based on natural deduction: the CMU Proof Tutor. An important difference from the (original) CMU Proof Tutor, however, is that *Syrus* focuses on quantified formulas over mathematical theories of central importance in CS education. *Syrus* also automatically generates targeted practice problems; it is not clear whether the new Proof Tutor will do that.

The EPGY Theorem Proving Environment is used by gifted students in Stanford's Education Program for Gifted Youth. Its published evaluation [9] involved 170 students doing geometry proofs. This is a sophisticated and rather heavyweight proof environment that can be used across a variety of mathematical domains, with a pedagogical focus on proving interesting results in a variety of underlying mathematical theories (including geometry, linear algebra, multivariate calculus, and differential equations). The tool's purpose is to fill in gaps in student proofs – anything that can be completed by the proof assistant Otter [6] within 4-5 seconds. It attempts to supply automatically the below-the-radar details that normally would not appear in a proof in the mathematical literature, and hence to allow students to prove theorems as would be done in "standard mathematical practice". Besides being far less interactive (being intended for a completely different purpose), *Syrus* differs in several important respects. It provides students with a platform for repeated practice with logic problems that involve underlying mathematics in theories of interest to CS students, so it can focus specifically on student understanding of quantified sentences and reasoning involving them.

7. CONCLUSION

Syrus provides a unique and novel approach to improving access to practice problems while providing instant feedback. The technique of using research tools to solve large numbers of intelligently generated candidate problems is one that could certainly be adapted to other topics in mathematics. Additionally, disciplines outside of mathematics are also candidates for using this method. For example, further research could determine whether software tools used by researchers could be adapted to generate and solve appropriate practice problems for students in an introductory chemistry or physics course. Unfortunately, the jury is still out on how effective such tools might be. Our study of *Syrus* does not establish efficacy but rather promise.

It has become especially clear that the key ideas behind *Syrus* will become increasingly important as higher education struggles with the impact of massive open on-line courses (MOOCs). MOOC students in STEM courses in particular will need to practice solving problems, some of which are of a sort that can be solved automatically with sophisticated research tools used by researchers in the same discipline. They must receive automated and, preferably, instantaneous feedback; having instructors grade homework for such a large number of students remains unfeasible and hiring armies of other human graders could prove expensive. It is likely that all results, positive and negative, regarding the use of technologies such as those developed and evaluated in this project, will prove relevant to the future of on-line education in general and MOOCs in particular.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grants DUE-0942542 and CCF-1162331. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation.

9. REFERENCES

[1] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340. Springer, 2008.

[2] K. A. Ericsson, R. T. Krampe, and C. Tesch-romer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, pages 363–406, 1993.

[3] K. R. M. Leino. Specification and verification of object-oriented software. Marktoberdorf International Summer School 2008, lecture notes.

[4] K. R. M. Leino. This is Boogie 2. Manuscript KRML 178, 2008. Available at http://research.microsoft.com/en-us/um/people/leino/papers.html.

[5] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proc. Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, pages 348–370. Springer, 2010.

[6] W. McCune. Otter and Mace2. http://www.cs.unm.edu/∼mccune/otter/, Aug. 2012.

[7] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A proof assistant for higher-order logic*. Springer-Verlag, 2002.

[8] W. Sieg and R. Scheines. *Philosophy and the Computer*. Westview Press, 1992.

[9] R. Sommer and G. Nuckols. A proof environment for teaching mathematics. *J. Autom. Reason.*, 32(3):227–258, Apr. 2004.