Experiences with Automated Compiler Exercises

Leena Salmela and Jorma Tarhio
Helsinki University of Technology

Abstract

We use home exercises in our Compiler course. We have implemented a graphical environment for doing compiler exercises related to finite state automata and parsers. The system also includes an automatic assessment system for the exercises. We introduce the main features of the system and review experiences of the first year of use.

1 Introduction

A course devoted to compiling techniques of programming languages belongs to the Computer Science curriculum of many universities. We use home exercises in our Compiler course. In our former system, a teaching assistant graded the written answers submitted by the students. Several problems have arisen with this approach. First of all, it has not been possible to give individual feedback to the students given the current resources and the feedback has arrived too late. Plagiarism has also been a growing concern in the course since all students have had the same assignments. Some students also found the practice of returning the home exercises as text files unintuitive.

To attack these problems we automated those home exercises that deal with finite state automata and parsers. Automatic assessment allows immediate feedback to the students and it is possible to give individual assignments to students thus alleviating the problem of plagiarism. Our system is called ACE which is short for Automated Compiler Exercises. The technical features of ACE were introduced in our work-in-the-progress paper [14]. In this paper we review experiences of using ACE and discuss its significance.

ACE contains a graphical environment for studying and completing the exercises. The visualizations are mostly adopted from JFLAP [3] which is a visualization tool for formal languages and automata theory. JFLAP is based on earlier work of Susan Rodger.

Automatic assessment has been successfully used in introductory courses at our university [11]. For example the Ceilidh system [1] and Scheme-Robo [13] have been used in the programming courses, the TRAKLA2 system [9], which has a graphical interface for doing algorithm simulation exercises, has been used in the Data Structures and Algorithms course and the Stratum framework [4] has been used in several other courses in our university. Automatic assessment has proved to be effective in these cases and the student response has also been generally positive.

There are a number of tools visualizing at least some parts of a compiler [2, 3, 6, 7, 12, 16, 17]. Several visualizations of finite automata and parsers have been developed. Some of these visualization tools, like JFLAP [3] and Exorciser [16], have taken a step towards automatic assessment. They allow the student to try building his own solution. Once the student thinks he has accomplished the task the tool will assess the solution and tell the student if it is right. These tools also allow the student to take a look at the model answer. However, these tools do not fully cover the exercises we have used and they are intended for self study so they do not keep track of students' points and solutions.

There are several studies on whether computer-aided assignments with visualizations would lead to a better learning outcome in Computer Science. Most conclusions are negative, see e.g. [5], but there are some positive cases, see e.g. [10]. Anyway, visualization is useful in practice by increasing study motivation of students [15], and computer-aided assignments are cheaper to organize than conventional approaches in large courses [8].

The rest of this paper is organized as follows. In Section 2 we give an overview of the exercises. In Section 3 we describe technical details of the ACE system. Section 4 describes the experiences with ACE we have had and in Section 5 we give some concluding remarks. Understanding of Sections 2 and 3 may require background knowledge in Computer Science.

2 Overview of the exercises

The home exercises of the course have been organized into six rounds. The first three rounds handle the front end of the compiler and the last three rounds the back end. The aim was to automate only the first three rounds, partly because the course will be split into two courses in the future, and the former part will get an even larger attendance while latter part will attract less students. The first three rounds deal with finite state automata (FSAs), LL parsing and LR parsing respectively. Each round has four exercises. From the constructive point of view, the exercises deal with building and traversal of graphs related to compiling either in a graphical form or as tables.

In the FSA round the student is given a regular expression and his task is to form a nondeterministic finite state automaton (NFA). Then the NFA is simulated with a given input. Next the NFA is converted to a deterministic finite state automaton (DFA) and this DFA is then simulated.

The second round deals with LL parsing. First the student should remove left recursion from the given grammar and do left factoring. Then the First and Follow sets are calculated and the LL parse table is constructed based on them. Finally, the parser is simulated with the given input.

In the third round an LR parser is constructed. First the student forms the LR(0) item sets for a given grammar and figures out the transitions between them. Then the First and Follow sets are calculated. Based on these sets, the LR parse table is constructed. The grammar is ambiguous, and so the parse table now contains ambiguity. Next the ambiguity is removed from the parse table so that the given precedence and associativity constraints are satisfied. Finally, the constructed parser is simulated.

Some of these exercises are clearly algorithm simulation exercises. The simulation of a FSA or a parser clearly falls into this category. The solution to this kind of exercise is an ordered list of steps. Some of the other exercises include simulation of an algorithm but the algorithm is more loosely defined. For example the algorithm used in the NFA construction does not define a total order for the construction of the automaton parts. Thus it only defines a partial order for the steps that are needed to construct the whole automaton. Of course a total order may be enforced in such an algorithm but this would unnecessarily complicate the exercise. Some of the exercises are even more loosely defined like the removal of left recursion from a grammar. In this case some transformation rules are presented in the study material but the use of exactly these rules is not enforced. These exercises are conceptual in nature. They test the student's understanding of the concept rather than knowledge of a specific algorithm.

We have around ten exercise sets for each round. A set for each student is chosen randomly among those sets. Moreover we allow permuting and replacing of local strings and names in the exercises in order to artificially increase the number of different exercises. In order to improve the quality of the exercises, we are investigating new ways to generate grammars and regular expressions.

3 ACE

3.1 Architecture

Given the various types of exercises that the system needs to support we decided to build a client for computer-aided doing the exercises and verifiers for checking them. These components could then be embedded into a framework which takes care of submissions and the needed book keeping. We call the client with the verifiers ACE.

Overviews of the client and the verifiers are given in subsections 3.2 and 3.3. We have embedded the client and the verifiers in a framework called Stratum [4] which has been developed in our university. Stratum follows the client/server architecture. Records on exercises, submissions, and results are kept on the server.

The ACE client is a Java applet which is used on the Web. This applet is the graphical environment for doing exercises, but with it the student also uses the services of the server. On the server, each student has a personal Web page which shows his personal assignments and the submission status of each subtask. The verifiers module is embedded in the server.

3.2 ACE Client

The central part of the ACE client is the visualization of the data structures needed in the exercises. Almost all these features are present in the JFLAP software [3]. Thus the ACE client is built reusing the code from JFLAP. However, some changes also needed to be done. JFLAP does not support showing precedence and associativity information of operators, and so a visualization for this was built. The simulation of FSAs and parsers in JFLAP are merely animations, and thus we needed to add some interaction so that the students can show how a FSA or parser works. For example when simulating an LL parser, the student has two choices in each step. He can either choose to advance in the input or apply a rule from the parse table.

Another major change was adding the notion of exercise rounds and exercises. Now ACE can lead the student through an exercise round one assignment at a time. Other changes included the design of a new file format which contains the information about assignments and exercise rounds. Because of the new file format, it is also not so easy for the students to use JFLAP to generate the correct answers. The generation of correct answers was of course disabled from the user interface.

Fig. 1 shows a screenshot of the ACE client. Here the user is converting a NFA to a DFA. He has already defined the initial state of the DFA and the state which the DFA enters after reading the symbol 'x' in the initial state. The labels of the DFA states show their corresponding NFA state sets.

3.3 Verifiers

We were also able to reuse some parts of JFLAP when building the verifiers. Some of the exercises like removing left recursion from a grammar are not supported by JFLAP so we needed to implement whole checkers for these. The simulation of FSAs and parsers in JFLAP are only animations without the possibility of error so we needed to implement new verifiers for these too.

The verifiers have the following general structure. First they check if the input the student used was the one given

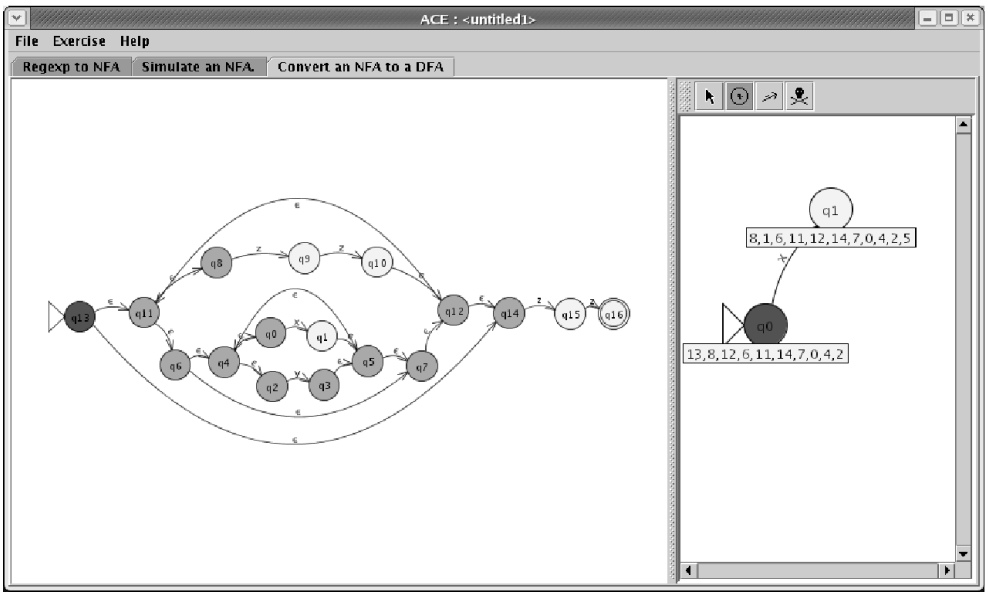

Figure 1: **A screenshot from the ACE client. Here the user is converting a NFA to a DFA.**

in his exercise. Then they generate the model answer and compare that to the student's solution. As a last step the verifiers generate feedback to the student.

Checking involves some computational difficulties. For example, one might like to check the removal of left recursion as follows. First one would check that there is no left recursion left in the grammar produced by the student. Then one would check that the grammar still produces the same language. This approach is unfortunately not possible because it is undecidable to determine if two context-free grammars produce the same language. So in this case we have to enforce the use of a set of transformation rules to be able to check the exercise.

4 Experiences with ACE

4.1 Activity of students and exam results

There are in total six exercise rounds in the course. In order to pass the course students have to pass three of these rounds. One of the passed rounds has to be among the first three exercise rounds associated with the front-end of a compiler and one has to be among the last three rounds associated with the back-end of a compiler. Doing more than the required number of exercises does not give any extra credit to a student. Because of this the first round is very popular, the second round somewhat less popular and the third round is done by less that half of the students since doing the third round does not benefit most students in any way.

Table 1 shows the statistics of the first three exercise rounds in years 2001–2004. In 2004 the first three rounds were solved using ACE while in 2001–2003 all the exercises were returned via email and checked by the assistant later. In each year of 2001–2003 the exercises were the same for all the students while in 2004 there were ten different versions of each exercise. Therefore in the previous years students could get more easily help from other students. In each year about a hundred students participated in the course.

Exercise round	*Submitted %*	*Passed %*
2004:		
1. Finite automata	92	88
2. LL-parsing	81	73
3. LR-parsing	72	53
2003:		
1. Finite automata	92	90
2. LL-parsing	85	84
3. LR-parsing	35	35
2002:		
1. Finite automata	87	81
2. LL-parsing	73	71
3. LR-parsing	44	43
2001:		
1. Finite automata	91	89
2. LL-parsing	87	84
3. LR-parsing	38	33

Table 1: **Exercise statistics of the Compiler course in years 2001–2004.**

In the previous years the students submitted a whole round at a time while in 2004 they submitted one exercise at a time. Thus a student who did not manage to do

the round completely might not have submitted anything in the previous years whereas in 2004 he might have submitted only the first exercise of a round (in order to be included in the number of Table 1). Therefore the numbers of active students are not quite comparable. The grading policy of an exercise round has also changed. In the previous years small mistakes did not result in failing the round because the students could submit their solution only once. With ACE the students were allowed to resubmit as many times as they wanted but to pass a round they needed to get all the exercises correct.

Table 1 shows that the percentage of students passing the first and the second round has not changed very much over the years. However, the third round was much more popular in 2004. Moreover, 40% of the students actually managed to pass all three rounds in 2004 while this number was considerably lower in the previous years. Therefore it seems that the system encouraged the students to do more exercises even if they are not given any extra points for this.

We compared the results of the mid-term exam of Fall 2004 covering the exercise rounds of ACE with those of the corresponding exams of the previous years. We did not notice any significant change. So it seems that ACE does not hinder learning. However, because the problems of these exams are not fully comparable, this is only a preliminary observation.

4.2 Questionnaire 1

We asked the students of the 2004 course answer to a questionnaire, and 60% of the 91 students replied. Concerning the possible advantages of ACE, the most common answer dealt with immediate feedback, which helped to correct errors. Also ability to make a round incrementally in several sessions and possibility to iterate answers were also important benefits.

Several students said that ACE saves time when compared with the traditional pen and paper way. However, the students who already mastered the topics of the exercises before using ACE, found the computer-aided approach slower than pen and paper, because they had to learn to use ACE. In the specific question, we asked whether the students felt wasting time with ACE. Only 13% of them answered yes, and 2% were uncertain. The rest 85% did not waste time. The average use time per round was 4 h 8 min.

We also asked the students about problems they encountered when using ACE, and 76% of them reported those, mostly minor technical problems. A part of the minor problems has already been corrected, and most of them will be taken care of before the next course will begin. A typical complaint dealt with vague error messages and unclear and limited instructions. Several students wished for demonstrations and examples in order to get easier acquainted with the system and the input formats.

Classification	*%*
Flop	2
Poorer	2
Similar	48
Better	39
Superior	9
	100

Table 2: **Answers to the question: "How did you find ACE when compared with the other tools (flop, poorer, similar, better, superior)?"**

Then we asked whether ACE supported learning. The answer was clear: 96% of the students answered "yes".

Finally, we requested the students to compare ACE with similar computer-aided tools for homework assignments of other courses. There are at least three other courses using such technology at our department. So 80% of the students had used similar systems earlier. Table 2 shows the result of the comparison. ACE was well received, and almost half of students regarded it at least better than the other systems. One student explained his opinion as follows pointing out some problems of computer-aided instruction: "ACE is better because it is not as mechanical as some other tools, which do not require pondering at all; yet even ACE provides so much help (in guiding towards the solution) that if only using ACE, one does not know what to do without it. So ACE is good, but pen-and-paper is still needed."

4.3 Questionnaire 2

Among the same student group as the first questionnaire, we made another questionnaire in connection with the course evaluation in which 54% of the students participated. A course evaluation is organized in the end of every course at our department. Besides fixed questions, the teacher is allowed to present additional questions. We asked the students to compare ACE with the traditional approach. The result is shown in Table 3. Note that the first questionnaire took place after the fourth exercise round, whereas the students had completed all the exercises while the second questionnaire was made.

The first question dealt with meaningfulness. Only few of the students had experienced the traditional way more meaningful than ACE. One third did not see difference between the alternatives, and more than half found the computer-aided way more meaningful.

Next the students were asked to compare the laboriousness of the approaches. Based on the first questionnaire we already knew that those students, who already mastered the subject, wasted time whereas those students, who did not know the subject beforehand, saved time. Now in the end of the course, half of the students found the traditional

Which way was...	*ACE*	*No diff.*	*Trad.*	
More meaningful?	59	33	8	100
More laborious?	18	31	51	100
Better from the point of view of learning?	51	37	12	100

Table 3: **Answers to the question: "Which way was more meaningful, more laborious, and better from the point of view of learning? ACE or the traditional approach?" The numbers are percentages.**

way more laborious. About one third experienced both approaches similar, and the rest of the students considered ACE more laborious.

The last question was about learning. Half of the students thought that the computer-aided approach was better from the point of view of learning. Only 12% of them found the traditional approach better.

5 Concluding remarks

We have described the ACE system for automatically assessing exercises related to finite state automata and parsers. The system supports individual exercises for students and it has a visual interface for studying and completing the exercises. The system has been used once in our Compiler course, and it was well received by the students.

ACE is an advanced learning environment. It makes the concepts of compiler construction concrete by visualizing them. ACE supports learning by doing. The student can submit a trial solution, and the system gives feedback about the possible errors. Because only correct submissions are accepted, the student is allowed to try each exercise of a round several times. Most of the exercises are constructed in such a way that it is almost impossible to reach an acceptable solution only by guessing.

ACE shares a part of the visualizations of JFLAP which is an excellent tool for demonstration and self-study purposes. However, the integrated assessment and bookkeeping make ACE more useful than JFLAP because the assignments of a large course can be delivered and graded automatically with ACE.

ACE works on the Web so that students can do the exercises anywhere at the time suitable for them. Thus ACE suits well to distance learning e.g. in a virtual university. The general principles of ACE can be adapted to other subject with constructive assignments.

Acknowledgements

We thank Susan Rodger and Tomi Janhunen for letting us use their codes.

References

[1] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Ceilidh: A course administration and marking system. In *Proceedings of the 1st International Conference of Computer Based Learning*, 1993.

[2] C. Boroni, F. Goosey, M. Grinder, and R. Ross. Engaging students with active learning resources: Hypertextbooks for the web. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, pages 65–69. ACM, 2001.

[3] R. Cavalcante, T. Finley, and S. H. Rodger. A visual and interactive automata theory course with JFLAP 4.0. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 140–144. ACM, 2004.

[4] T. Janhunen, T. Jussila, M. Järvisalo, and E. Oikarinen. Teaching Smullyan's analytic tableaux in a scalable learning environment. In *Proceedings of the 4th Finnish/Baltic Sea Conference on Computer Science Education*, 2004.

[5] C. Kehoe, J. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[6] A. Kerren. Animation of the semantical analysis. In *Proceedings of 8. GI-Fachtagung Informatik und Schule, INFOS99*. Informatik aktuell, Springer, 1999. (in German).

[7] J. Khuri and Y. Sugano. Animating parsing algorithms. In *Proceedings of the 29th SIGCSE Technical Symposium*, pages 232–236. ACM, 1998.

[8] A. Korhonen. Visual Algorithm Simulation. Dissertation. Helsinki University of Technology, 2003.

[9] A. Korhonen, L. Malmi, and P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of the 3rd Finnish/Baltic Sea Conference on Computer Science Education*, pages 48–56, 2003.

[10] R. Levy, M. Ben-Ari, and P. Uronen. The Jeliot 2000 program animation system. *Computers and Education*, 40(1):1–15, 2003.

[11] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of the 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 55–59. ACM, 2002.

[12] R. D. Resler and D. M. Deaver. VCOCO: A visualisation tool for teaching compilers. *ACM SIGCSE Bulletin*, 30(3):199–202, 1998.

[13] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. In *Proceedings of the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 133–136. ACM, 2001.

[14] L. Salmela and J. Tarhio. ACE: Automated compiler exercises. In *Proceedings of the 4th Finnish/Baltic Sea Conference on Computer Science Education*, 2004.

[15] J. Stasko. Using student-built algorithm animations as learning aids. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*, pages 25–29. ACM, 1997.

[16] V. Tscherter, R. Lamprecht, and J. Nievergelt. Exorciser: Automatic generation and interactive grading of exercises in the theory of computation. In *4th International Conference on New Educational Environments*, pages 47–50, 2002.

[17] S. R. Vegdahl. Using visualization tools to teach compiler design. *Journal of Computing Sciences in Colleges*, 16(2):72–83, 2001.