

ACE: Automated Compiler Exercises

Leena Salmela and Jorma Tarhio
Helsinki University of Technology

lsalmela@cs.hut.fi, tarhio@cs.hut.fi

Abstract

Automatic assessment has become popular in large introductory courses in computer science. Although the main issue has been to reduce the work load of course personnel also other benefits like better and faster feedback have been attained. We have constructed an automatic assessment system for home assignments to be used in a Compilers course. The system also features a graphical environment for doing the assignments.

1 Introduction

Home assignments have been a part of the Compilers course in Helsinki University of Technology for several years. The home assignments have covered finite state automata, parsers, generation of intermediate code, flow analysis and register allocation. The purpose of these assignments has been to activate the students during the term.

However, several problems have arisen with the assignments. First of all, given the current resources it has not been possible to give individual feedback to one hundred students 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 have also found the current practice of returning the home assignments as text files in a specified form unintuitive.

To attack these problems we are automating those home assignments that deal with finite state automata and parsers. Automatic assessment would allow for immediate feedback to the students and it would be possible to give individual assignments to students thus alleviating the problem of plagiarism. The system also contains a graphical environment for studying and completing the assignments. The visualizations are mostly adopted from JFLAP (Cavalcante et al., 2004) 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 in the Helsinki University of Technology (Malmi et al., 2002). For example the Ceilidh system (Benford et al., 1993) and Scheme-Robo (Saikkonen et al., 2001) have been used in the programming courses, the TRAKLA2 system (Korhonen et al., 2003), which has a graphical interface for doing algorithm simulation exercises, has been used in the Data Structures and Algorithms course and the Stratum framework (Janhunen et al., 2004) has been used in several courses in the laboratory of Theoretical Computer Science. Automatic assessment has proved to be effective in these cases and the student response has also been generally positive.

Several visualizations of finite automata and parsers have been developed. Some of these visualization tools, like JFLAP (Cavalcante et al., 2004) and Exorciser (Tscherter et al., 2002), 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 assignments we have used and they are intended for self study so they do not keep track of students' points and solutions.

The rest of this paper is organized as follows. In Section 2 we give an overview of the assignments to be automated. In Section 3 we describe the ACE system and in section 4 we give some concluding remarks.

2 Overview of the Assignments

The assignments 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. We are now attempting 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 assignments.

In the first assignment of the FSA round the student is given a regular expression and his task is to form a nondeterministic finite state automaton (NFA) using Thompson's construction. Then in the second assignment the constructed NFA is simulated with a given input. In the third assignment the NFA is converted to a deterministic finite state automaton (DFA) and this DFA is then simulated in the last assignment of this round.

The second round deals with LL parsing. First the student should remove left recursion from the given grammar. In the second assignment this grammar is left factored. Then in the third assignment the First and Follow sets needed in the LL parse table construction are calculated. Also the filling of the parse table is part of the third assignment. Then in the last assignment the constructed parser is simulated with the given input.

In the third round an LR parser is constructed. In the first assignment the student forms the LR(0) item sets of a given grammar and figures out the transitions between them. Then in the second assignment the First and Follow sets are calculated. Based on these sets also the LR parse table is constructed. The grammar used in this exercise is ambiguous so the parse table now contains ambiguity. In the third assignment of this round the ambiguity is removed from the parse table so that the given precedence and associativity constraints are satisfied. In the last assignment the constructed parser is simulated with the given input.

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 Thompson's construction algorithm 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 assignment. Some of the assignments 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 assignment 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 assignments in order to artificially increase the number of different assignments. We are also studying ways to generate new grammars and regular expressions for assignments.

3 ACE

Given the various types of exercises that the automatic assessment system needs to support we decided to build a client for 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 and the verifiers Automated Compiler Exercises (ACE).

Here we give an overview of the client and the verifiers. We have embedded these in a framework called Stratum (Janhunen et al., 2004) which has been developed in the Laboratory of Theoretical Computer Science in Helsinki University of Technology.

3.1 ACE Client

The central part of the ACE client is the visualization of the data structures needed in the assignments. The client needs to support the following visualizations:

- Editing and simulating a FSA. When converting a NFA to a DFA or building the LR item set automaton special restrictions apply to editing the labels of states.
- Editing a grammar.
- Defining the First and Follow sets for a grammar.
- Editing LL and LR parse tables.
- Showing precedence and associativity information of operators.
- Simulating LL and LR parsers.

Almost all these features are present in the JFLAP software (Cavalcante et al., 2004). Thus the ACE client is built reusing the code from JFLAP. Some changes of course needed to be done. JFLAP does not support showing precedence and associativity information for operators so visualization for this was built. The simulation of FSAs and parsers in JFLAP are merely animations so we needed to add some interactivity here so that the students can show how the FSA or parser works. For example when simulating an LL parser using ACE 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 assignment rounds and assignments. Now ACE can lead the student through a assignment round one assignment at a time. Other changes included the design of a new file format which contains the information about assignments and assignment 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. In the end the possibility to submit an assignment was added to ACE when it was embedded to the Stratum framework.

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.

3.2 Verifiers

We were also able to reuse some parts of JFLAP when building the verifiers. Some of the assignments like removing left recursion from a grammar are not supported by JFLAP so we needed to implement whole verifiers 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 in his assignment. 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.

It would be nice to check some of the exercises in a bit different way. For example we would like to check the removal of left recursion as follows. First we would check that there is no left recursion left in the grammar produced by the student. Then we would need to 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.

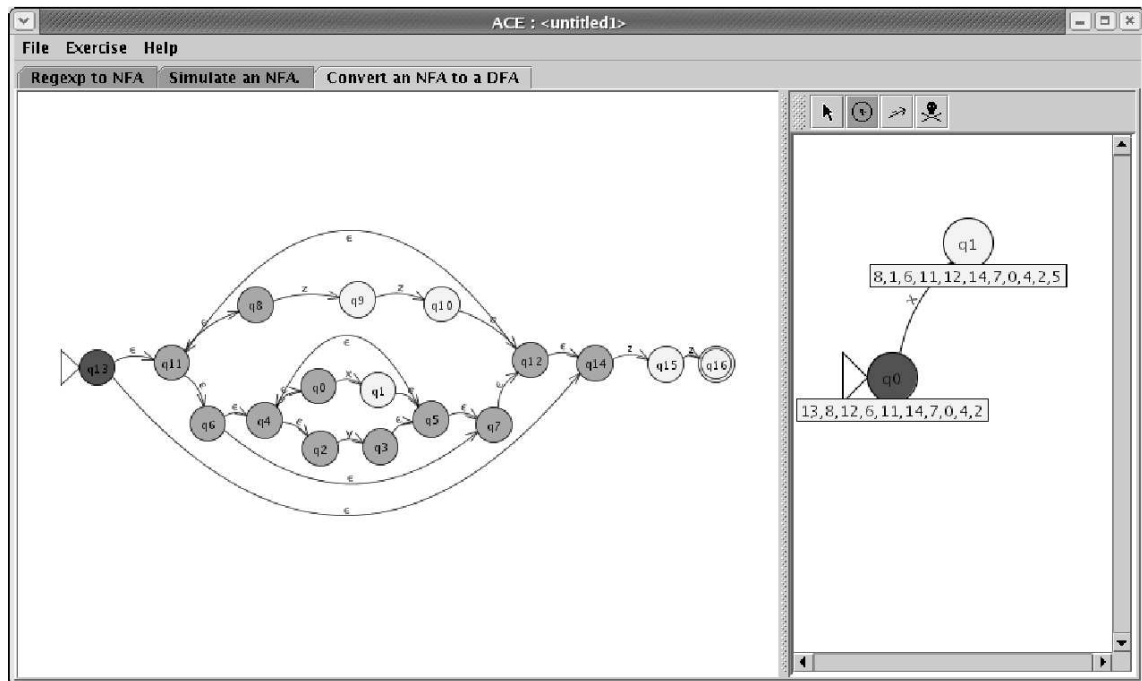


Figure 1: A screenshot from the ACE client. Here the user is converting a NFA to a DFA.

4 Concluding Remarks

We have described the ACE system for automatically assessing assignments related to finite state automata and parsers. The system supports individual assignments for students and it has a visual interface for doing the assignments.

The system has been used this fall in our Compilers course. We will report detailed experiences and results later on. The first impression is that our students found the system appealing. When considering the activity of students in our course in 2002–04, this fall we noticed an increase in the number of students working on the third assignment which is a voluntary exercise for most students.

5 Acknowledgements

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

References

- Benford, S., Burke, E., Foxley, E., Gutteridge, N., Zin, A. M., 1993. Ceilidh: A course administration and marking system. In: Proceedings of the 1st International Conference of Computer Based Learning.
- Cavalcante, R., Finley, T., Rodger, S. H., 2004. A visual and interactive automata theory course with JFLAP 4.0. In: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. ACM, pp. 140–144.
- Janhunnen, T., Jussila, T., Järvisalo, M., Oikarinen, E., 2004. Teaching Smullyan’s analytic tableaux in a scalable learning environment. In: Proceedings of the 4th Finnish/Baltic Sea Conference on Computer Science Education.
- Korhonen, A., Malmi, L., Silvasti, P., 2003. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In: Proceedings of the 3rd Finnish/Baltic Sea Conference on Computer Science Education. pp. 48–56.

- Malmi, L., Korhonen, A., Saikkonen, R., 2002. 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. ACM, pp. 55–59.
- Saikkonen, R., Malmi, L., Korhonen, A., 2001. Fully automatic assessment of programming exercises. In: Proceedings of the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education. ACM, pp. 133–136.
- Tscherter, V., Lamprecht, R., Nievergelt, J., 2002. Exorciser: Automatic generation and interactive grading of exercises in the theory of computation. In: 4th International Conference on New Educational Environments. pp. 47–50.