HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

GraFLAP – Automated Grading of Formal Languages and Automata Problems

Frauke Sprengel

University of Applied Sciences and Arts Hannover
Faculty IV, Department of Computer Science
frauke.sprengel@hs-hannover.de

Technical Report

November 2024

Abstract

Grammars, automata, and machines in theoretical computer science can be viewed as specialized types of programs. Automata and grammars can be graded automatically using sets of words that either belong or don't belong to the corresponding language. For machines, traditional unit testing with predefined input and output can be performed. Additionally, we can verify whether automata, grammars, or machines are of the requested type. Students use JFLAP for constructing and testing automata and machines. Our grading system, GraFLAP, builds upon JFLAP and includes additional testing capabilities. We provide tasks in ProFormA format and utilize Grappa to connect the grader to Moodle.

Keywords: Computer aided assessment, GraFLAP, JFLAP, Grappa, ProFormA, LMS, Moodle

DOI of this report: `https://doi.org/10.25968/opus-3392`

1 Introduction

Introducing computer-aided assessment (CAA) in theoretical computer science lectures presents similar challenges to those found in mathematics. Basic testing can be done through multiple choice, numerical inputs, and simple string responses. More advanced options are available through systems that use computer algebra in the background, such as LON-CAPA [Kor+01] or Moodle STACK [SG06]. These systems can test formulas, regular expressions, or verify specific mathematical properties. While some progress has been made in automated proof assessment in STACK (cf. e.g. [BS21]), there remains a significant role for traditional paper-and-pencil homework. For automata, machines, and grammars, we offer an integrated solution within learning management systems like Moodle.

Our students previously used JFLAP primarily for creating visual representations of automata. Most never explored JFLAP's full functionality as described in section 2, such as testing whether their automata accept the intended words. We developed GraFLAP, see section 4, as an extension of JFLAP to enable grading of automata, grammars, regular expressions, and machines. Since these exercises typically have multiple valid solutions, we test functionality by verifying whether a specific language is generated by the grammar or accepted by the automaton, or whether certain inputs produce the expected machine outputs. GraFLAP accepts JFLAP XML files containing automata or machines, or simple strings for grammars, regular expressions, or example words as an input. The system then tests the student's submission against the instructor's specifications.

We initially integrated the grader with LON-CAPA [SR17], allowing for randomized exercise versions to prevent cheating. After our university transitioned to Moodle as its sole learning management system, we needed to adapt GraFLAP accordingly, see section 6. Our university's "Automated Grading of Programming Exercises" group had already developed a specialized question type for programming tasks using an updated version of Grappa [GHW15] to connect external graders to Moodle. This led us to enhance GraFLAP for Grappa compatibility.

Now, we have GraFLAP questions seamlessly integrated into Moodle quizzes.

2 Interactive Program JFLAP

JFLAP (Java Formal Languages and Automata Package) is an interactive standalone Java program for teaching theoretical computer science. It was developed by Susan Rodgers [Rod06] and her team. Students can construct automata, experiment with

grammars, and explore many other features. The program is considered the gold standard in this field (see next section).

In detail, JFLAP covers the following topics:

- finite, push-down, and Turing automata
  - construction and testing
  - conversion to grammar and back
  - detection of non-determinism
- for finite automata
  - converting non-deterministic finite automata into deterministic ones
  - minimizing finite automata
  - conversion to regular expressions and back
  - complete finite automata
- grammars
  - construction and testing
  - verification of grammar type
- regular expressions

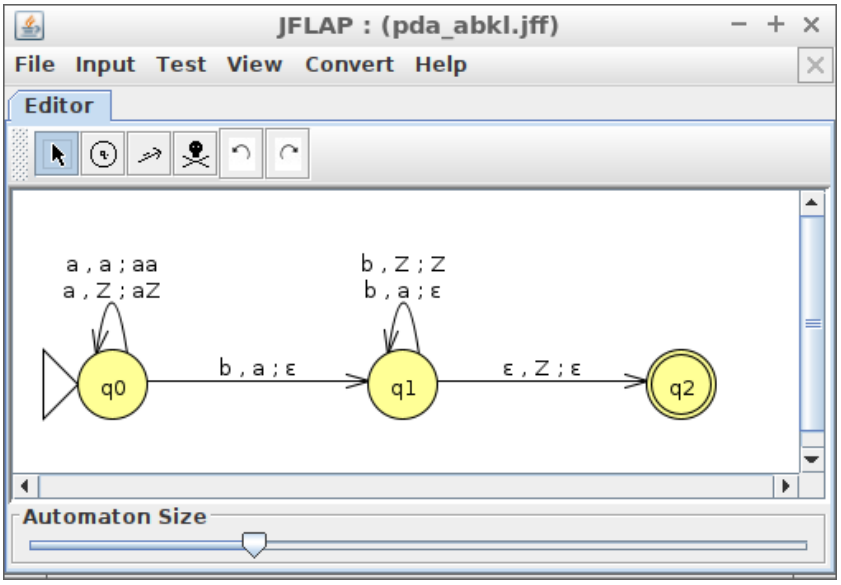


Figure 1: Push-down automaton in JFLAP.

- pumping lemma for regular and context-free languages
- Mealy and Moore machines, Turing machines (with one or more tapes)

The program is user-friendly and largely self-explanatory. Tutorials are available on their website[1]. Several years ago, JFLAP 7 was internationalized at our university, allowing us to use JFLAP 7 with either an English or German user interface.

3 Related Work

A comprehensive overview of automata simulation tools and their educational applications is provided in [CSK11]. The authors identify JFLAP as "the most sophisticated tool for simulating automata".

Building upon JFLAP's foundation, several tools for teaching and assessing theoretical computer science concepts have emerged, including [She+14b; She+14a] and [Bez+22]. These developments primarily focused on deterministic finite automata, utilizing JFLAP as their input mechanism. In [She+14b; She+14a], students receive immediate feedback within JFLAP itself. They can work on instructor-provided problems or create their own language definitions using simple Java routines or through a graphical interface. The approach in [Bez+22] allows students to submit their solutions through JFLAP to a feedback server. However, neither approach supports grading integration within learning management systems like Moodle.

The FSMbuilder tool [RRE24], developed for the PrairieLearn platform, specifically targets finite automata.

In [Moh20b; MSR21; Moh20a], the authors present an innovative e-textbook featuring simulations and auto-graded exercises. This comprehensive resource covers all aspects of an introductory theoretical computer science course, with auto-graded exercises forming one component rather than the primary focus. The grading system builds on a JavaScript reimplementation of JFLAP. Automated feedback relies on test string sets, similar to software development unit tests - an approach we also employ in GraFLAP. The e-textbook integrates with Moodle through LTI as part of an OpenDSA server. While this is a valuable resource, translating it to German would require significant effort. Our existing materials and exercises are already in German, and we specifically need automata, machine, and grammar exercises in Moodle rather than a complete e-textbook.

AutomataTutor [Dan+15; DAn+20] offers a web-based platform for exercises involving finite automata, push-down automata, Turing machines, regular expressions, and

[1] `www.JFLAP.org`

context-free grammars. It provides immediate, constructive feedback but lacks integration with learning management systems.

Another interesting approach is ACL2 [KWM23] for checking several types of automata. Students must describe their automata using a specialized notation (requiring additional learning effort). This tool integrates with GradeScope.

On the other hand, several Moodle plugins exist for grading programming tasks, such as Coderunner [LH16], which includes a graphical interface [Lob24] for inputting graphs and automata. Using it for automata grading requires libraries like [Sie24; ER23]. Similarly, [GS24a; GS24b] developed an unofficial Moodle plugin for grading graph-related exercises, including automata. While this represents an interesting approach, it remains unofficial.

4 Grading Engine GraFLAP

We developed GraFLAP (Grader for Formal Languages and Automata Problems) as an extension of JFLAP, enabling its use as a grading engine within learning management systems. This development occurred over several years through the author's work [Spr16; SR17] in conjunction with multiple student theses [Tos13; Hel16; Son22]. GraFLAP supports grading of the following problem types:

- Given a language (specified as a set, by a grammar, regular expression, text, etc.) – construct an automaton (finite / push-down / Turing)
- Given a language (specified as a set, by an automaton, regular expression, text, etc.) – create a grammar (of a specified type)
- Given a language (specified as a set, by an automaton, grammar, regular expression, text, etc.) – provide example words
- Given a regular language (specified as a set, by an automaton, grammar, text, etc.) – create a regular expression
- Given a type of automaton / grammar – provide an example
- Given a task (description of input and output) – design a machine

The language specified in the problem should be provided to the program in a straightforward format: as a regular expression, a grammar, or collections of words that belong and don't belong to the language. Testing primarily involves unit tests, verifying whether words in the language are accepted/generated and words outside it are rejected.

For machine-related tasks, the problem specifications should be provided either as pairs of input and corresponding output words, or as an example machine with a set of input

words. Testing involves black-box verification of correct output generation from given inputs.

Here are some example problems that GraFLAP can handle.

Example 1 (Example Words) *Given the following language*

$$L = \{uv^T \mid u, v \in \{0,1\}^* \text{ with } |u| = |v| \text{ and } \forall i : u_i \neq v_i\}$$

Input 6 words (excluding the empty word) from the language, separated by commas.

Example 2 (Automaton) *Design an automaton that accepts the following language*

$$L = \{a^n b^n \mid n \in \mathbb{N}\}.$$

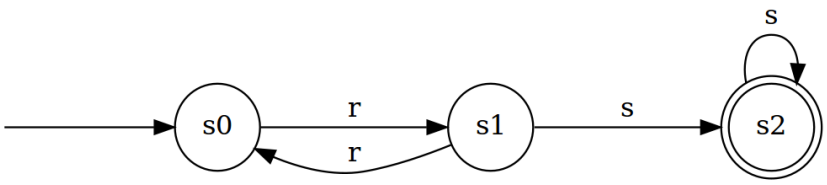
Use JFLAP to solve this problem and upload your solution file here.

Example 3 (Grammar of specified type) *Create an example of a context-free grammar that is not right-linear. The generated language should use the alphabet* $\{a, b, c\}$.

Format your grammar as follows:

Separate rules with commas (,), use -> between left- and right-hand sides, separate multiple right-hand sides with |, use S as the start symbol, represent the empty word with E, use upper case letters for variables (non-terminals), and lower case letters for terminal symbols. Example grammar format: S->E, S->A|aBa, aBa->abba, A->b|E.

Example 4 (Grammar for a given language) *Given a finite automaton represented by this state diagram:*



Create a right-linear grammar for the language this automaton accepts. Place any ε*-rules at the end. If they cause problems, eliminate them.*

Example 5 (Turing machine) *Design a Turing machine that converts binary numbers into unary form. Consider only positive binary numbers in your solution.*

Use JFLAP to solve this problem and upload your solution file here.

Students receive feedback showing the state diagram of their submitted automaton or machine (when applicable) and testing results. We verify whether the required language is accepted by the automaton or generated by the grammar, and check if the grammar or automaton is of the specified type. For machines, we test whether given inputs produce the expected outputs through unit testing.

5 Modes and Types

GraFLAP requires specific parameters to define each problem. It supports the following operational modes:

- **a(r|g)[t][w][p]** (Automaton, Regex or Grammar, [Type], [Words], [Parts]),

  Students must create an automaton for a language described by the instructor's regular expression or grammar. Optional parameters include type specification, test words, and requests for formal definitions of automaton components (states, initial states, final states, alphabet, [stack alphabet], transition relation).

- **gg[t][w]** (Grammar, Grammar, [Type], [Words]),

  Students must create a grammar for a language described by the instructor's grammar. Type specification and test words are optional parameters.

- **e(a|g)t** (Example, Automaton or Grammar, Type),

  Students must provide an example of either:

  - An automaton of a specified type (deterministic/non-deterministic, finite state, push-down, or Turing)
  - A grammar of a specified type (right-linear, context-free, non-context-free)

- **m(p|mw)** (Machine, Pair) or (Machine, Machine, Words)

  Students must design a machine according to the instructor's specifications. Testing uses either:

  - Instructor-provided pairs of input and corresponding output words, separated by % between pairs and semicolons (;) within pairs. Only valid input-output pairs are tested, so there is no need for the separator !.
  - Instructor-provided machine and input words, where output words are generated by the instructor's machine.

- **ww** (Words)

  Students must provide example words for a language described by the instructor's regular expression or grammar.

- **rr[w]** (Regex, Regex, [Words])

  Students must create a regular expression for a language described by the instructor's regular expression. Test words are optional.

Grammars should be written as strings following this format:

```
S -> aB , B -> bS | E | a , S -> aSa
```

The grammar notation conventions are:

- Comma (,) separates rules
- Arrow (->) connects left- and right-hand sides
- Pipe (|) separates alternative right-hand sides
- Capital E represents the empty word (instead of ε)
- Capital S denotes the start symbol
- Other capital letters represent non-terminal symbols
- Lower case letters (a-z) and 0,1 serve as terminal symbols
- Spaces are optional and can be used freely

Grammar types include:

- **rl**: right-linear
- **rlcfg**: right-linear or context-free
- **cfg**: context-free (excluding right-linear)
- **ncfg**: non-context-free
- **non**: type not specified

Automata types include:

- **fa**: finite state automata
- **pda**: push-down automata
- **tm**: Turing automata
- **non**: type not specified

Prefix options include **d** for deterministic or **n** for non-deterministic.

Machine types include:

- **mealy**: Mealy machines
- **moore**: Moore machines
- **tm**: Turing machines

Prefix options include **d** for deterministic or **n** for non-deterministic Turing machines.

6 Integration into Moodle

The "Automated Grading of Programming Exercises" group at our university developed the middleware Grappa [Bec+13; GHW15] to connect various programming task graders with different learning management systems.

We utilize a new version of Grappa[2], which implements the ProFormA[3] format [Str+15] for data exchange.

For Moodle integration, we developed a new question type plugin called MooPT[4], which interfaces with the Grappa web service. Instructors provide programming tasks in ProFormA format. The plugin displays the task description from the ProFormA file as the question text and accepts either file uploads or text input from students.

[2] `https://github.com/hsh-elc/grappa-webservice`
[3] `https://github.com/ProFormA`
[4] `https://github.com/hsh-elc/moodle-qtype_moopt`

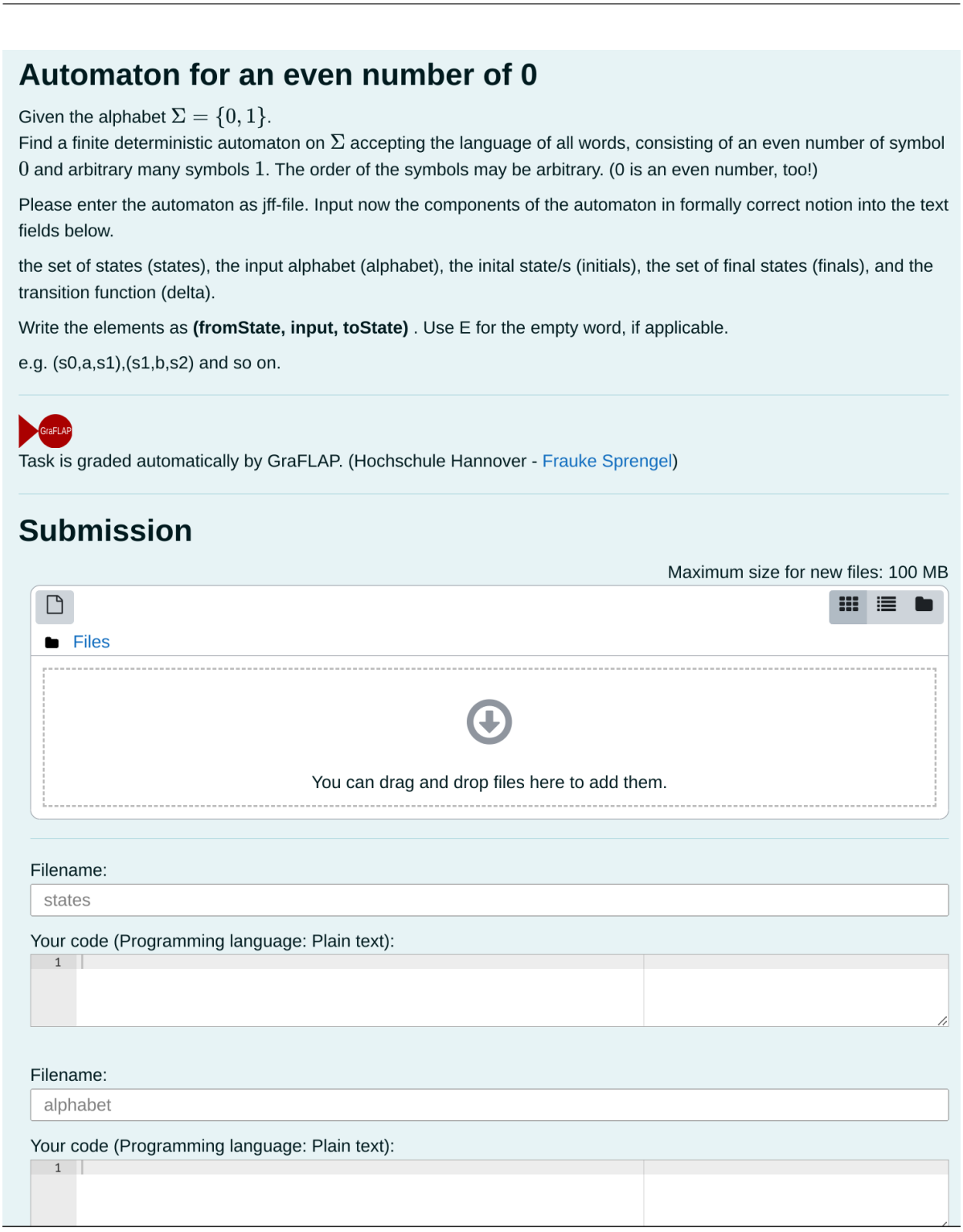


Figure 2: MooPT question for a finite automaton to be graded by GraFLAP.

Instructors can define grading schemes within the ProFormA task, with the option to show these schemes to students in advance.

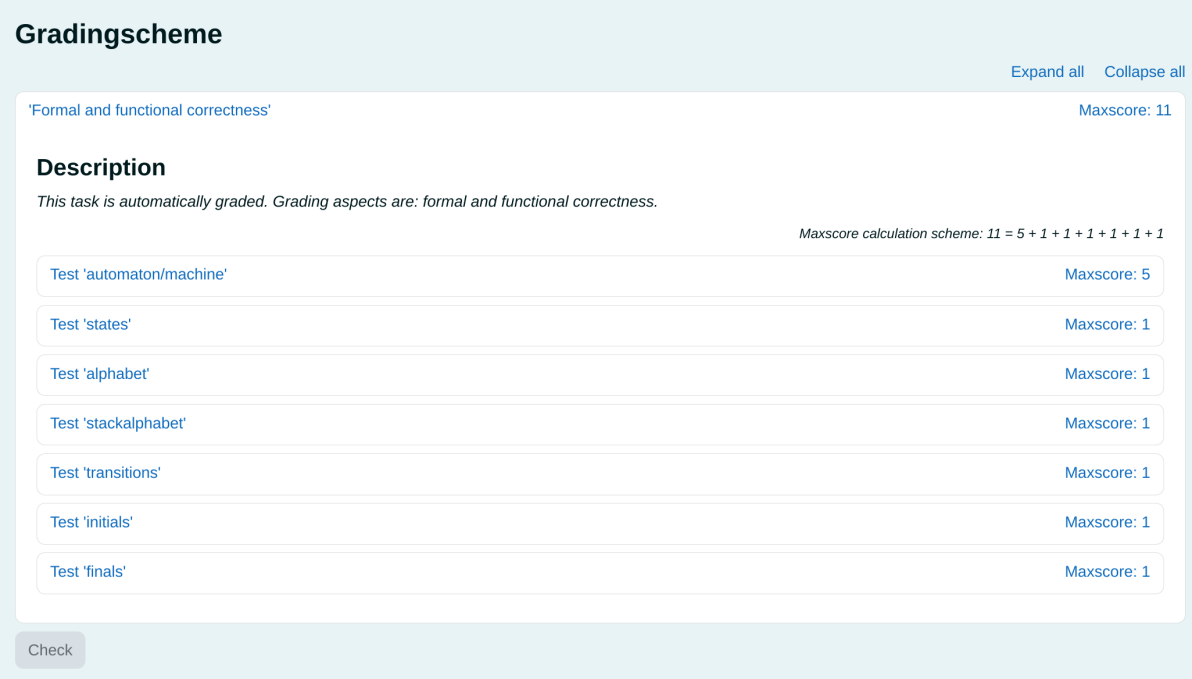


Figure 3: Grading scheme in MooPT question for a push-down automaton to be graded by GraFLAP.

Upon submission, the plugin transmits all data through Grappa to a grader like GraFLAP. The grading results are then displayed, including a state diagram of the student's submission for automata and machines. Students also receive feedback on the number of failed word tests. Teacher-specific feedback remains visible only to instructors.

The system supports various assessment types [Rol22], including immediate and adaptive feedback (with or without penalties) for formative assessment, and deferred feedback for summative assessment. All Moodle quiz functionalities are available, and the system can integrate with other platforms using ProFormA.

Of course, connecting GraFLAP to other systems using ProFormA is possible.

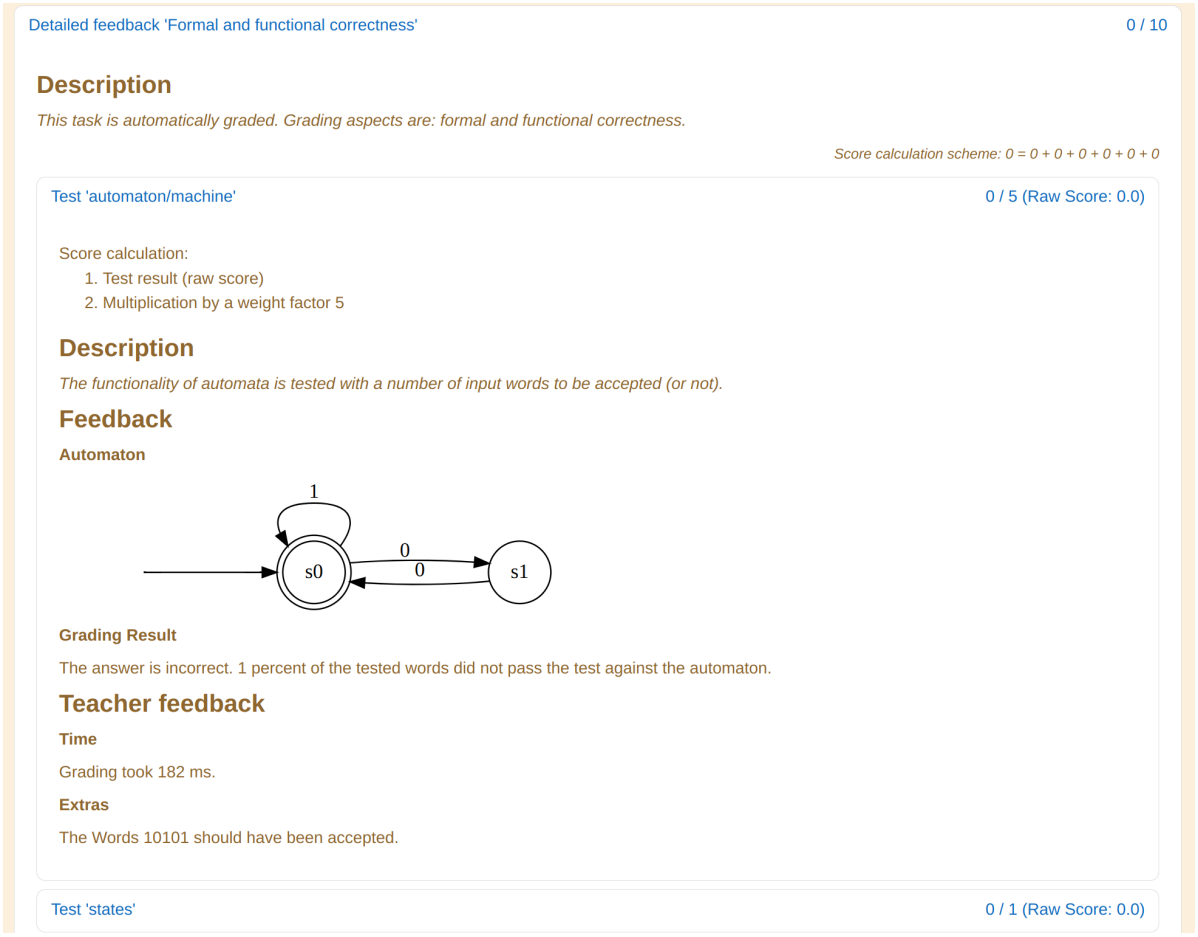


Figure 4: Result of a grading in MooPT question for the finite automaton in figure 2.

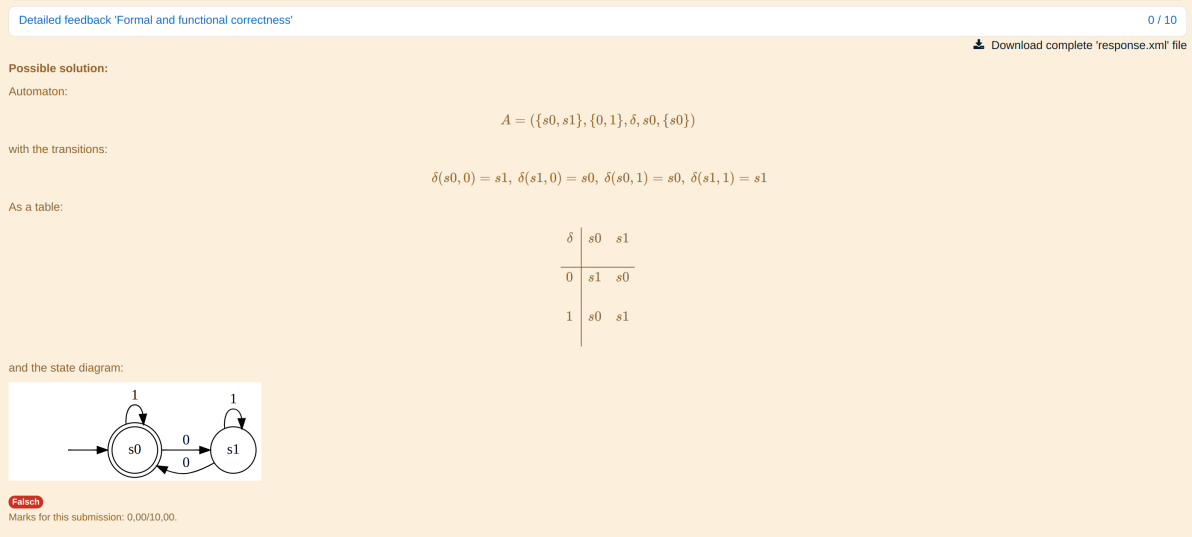


Figure 5: General feedback with model solution in MooPT question for the finite automaton in figure 2.

Our existing tasks were originally stored as LON-CAPA problem XML files with additional Perl libraries. LON-CAPA's straightforward randomization capabilities made it easy to create different versions of exercises to prevent cheating. To maintain this functionality, we employed LCJFLAP2Mdl[5], a fork of LC2Mdl[6] [DS21], to generate randomized ProFormA tasks from single LON-CAPA problems. Each ProFormA task creates one question in Moodle's question bank. Different task distribution among students is achieved through Moodle's built-in "choose a random question from the given category" feature.

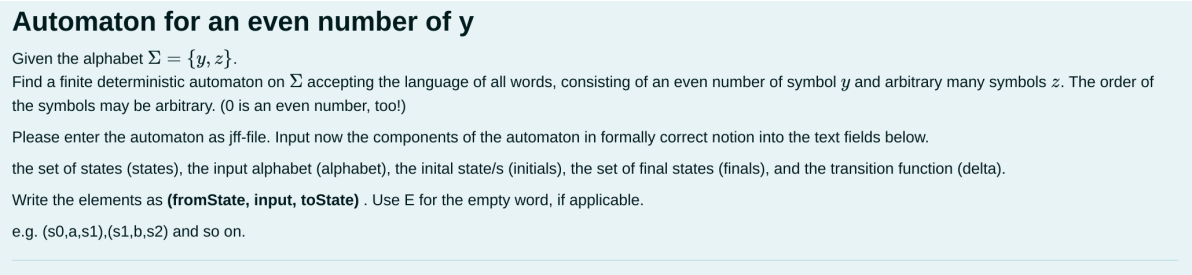


Figure 6: Other variant of the MooPT question for the finite automaton in figure 2.

[5] `https://github.com/nfa019/lcJflap2mdl`

[6] `https://github.com/kiliandangendorf/lc2mdl`, converter for LON-CAPA problems to Moodle STACK questions

7 Summary

We have developed GraFLAP, a comprehensive grading system for automata, grammars, regular expressions, and machines that builds upon JFLAP's established foundation. The system successfully combines JFLAP's intuitive interface for construction and testing with robust assessment capabilities integrated into modern learning management systems.

Our implementation provides several key features:

- Seamless integration with Moodle through the Grappa middleware and MooPT question type plugin
- Support for multiple problem types and assessment formats
- Automated testing with immediate feedback
- Implementation of ProFormA format for standardized task descriptions
- Capabilities for randomized exercise variants

The system benefits both instructors and students: instructors gain efficient automated grading tools with flexible assessment options, while students receive immediate feedback within their familiar learning environment. By combining JFLAP's interactive features with automated assessment capabilities, we have created a practical solution that enhances both teaching and learning in theoretical computer science education.

References

[Bec+13] S. Becker et al. "Prototypische Integration automatisierter Programmbewertung in das LMS Moodle". In: *Proceedings of the First Workshop Automatische Bewertung von Programmieraufgaben*. CEUR, Vol. 1067. Hannover, 2013. URL: https://ceur-ws.org/Vol-1067.

[Bez+22] Ivona Bezáková et al. "Effective Succinct Feedback for Intro CS Theory: A JFLAP Extension". In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*. SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, pp. 976–982. ISBN: 9781450390705. DOI: 10.1145/3478431.3499416. URL: https://doi.org/10.1145/3478431.3499416.

[BS21] Robert Thomas Bickerton and Chris J Sangwin. "Practical online assessment of mathematical proof". In: *International Journal of Mathematical Education in Science and Technology* (2021), pp. 1–24. DOI: 10.1080/0020739X.2021.1896813. URL: https://doi.org/10.1080/0020739X.2021.1896813.

[CSK11] Pinaki Chakraborty, P. C. Saxena, and C. P. Katti. "Fifty years of automata simulation: a review". In: *ACM Inroads* 2.4 (2011), pp. 59–70. ISSN: 2153-2184. DOI: 10.1145/2038876.2038893. URL: https://doi.org/10.1145/2038876.2038893.

[Dan+15] Loris D'antoni et al. "Automata Tutor and what we learned from building an online teaching tool". In: *Bull. EATCS* 117 (2015). URL: https://api.semanticscholar.org/CorpusID:13591267.

[DAn+20] Loris D'Antoni et al. "Automata Tutor v3". In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 3–14. ISBN: 978-3-030-53291-8.

[DS21] Kilian Dangendorf and Frauke Sprengel. "Half-Automatic Migration of LON-CAPA Problems to Moodle STACK". In: *Proceedings of the International Meeting of the STACK Community 2021*. Tallinn, 2021. DOI: 10.5281/zenodo.4916148. URL: https://doi.org/10.5281/zenodo.4916148.

[ER23] Caleb Evans and Eliot W. Robson. "automata: A Python package for simulating and manipulating automata". In: *Journal of Open Source Software* 8.90 (2023), p. 5759. DOI: 10.21105/joss.05759. URL: https://doi.org/10.21105/joss.05759.

[GHW15] Robert Garmann, Felix Heine, and Peter Werner. "Grappa - die Spinne im Netz der Autobewerter und Lernmanagementsysteme". In: *DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik*. Ed. by Hans Pongratz and Reinhard Keil. Bonn: Gesellschaft für Informatik e.V., 2015, pp. 169–181. URL: https://dl.gi.de/handle/20.500.12116/2048.

[GS24a] Arthur van Goethem and Willem Sonke. *GraphChecker*. 2024. URL: https://edin.win.tue.nl/graphchecker/index.html.

[GS24b] Arthur van Goethem and Willem Sonke. *GraphChecker Repository*. 2024. URL: https://github.com/graphchecker/moodle-qtype_graphchecker.

[Hel16] Benjamin Held. "Erweiterung einer Bewertungssoftware für LON-CAPA unter Verwendung von JFLAP". Master's Thesis. Hannover: University of Applied Sciences and Arts Hannover, 2016. URL: https://doi.org/10.25968/opus-1924.

[Kor+01] Gerd Kortemeyer et al. "The LearningOnline Network with CAPA Initiative". In: *IEEE Frontiers in Education Conference Proceedings, vol. 31*. 2001, p. 1003. URL: https://loncapa.org.

[KWM23] Ankit Kumar, Andrew Walter, and Panagiotis Manolios. "Automated Grading of Automata with ACL2s". In: *Electronic Proceedings in Theoretical Computer Science* 375 (2023), pp. 77–91. ISSN: 2075-2180. DOI: 10.4204/eptcs.375.7. URL: http://dx.doi.org/10.4204/EPTCS.375.7.

[LH16] Richard Lobb and Jenny Harlow. "Coderunner: a tool for assessing computer programming skills". In: *ACM Inroads* 7 (2016), pp. 47–51.

[Lob18] Richard Lobb. *Moodle Plugin CodeRunner with GraphUI for Graphs*. 2018. URL: https://coderunner.org.nz/mod/forum/discuss.php?d=99.

[Lob24] Richard Lobb. *Moodle Plugin CodeRunner with GraphUI*. 2024. URL: https://coderunner.org.nz/mod/page/view.php?id=552.

[Moh20a] Mostafa Kamel Osman Mohammed. *OpenFLAP Editors and Tools*. 2020. URL: https://opendsa.cs.vt.edu/ODSA/AV/OpenFLAP/OpenFLAP.html.

[Moh20b] Mostafa Kamel Osman Mohammed. "Teaching Formal Languages through Visualizations, Simulators, Auto-graded Exercises, and Programmed Instruction". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: Association for Computing Machinery, 2020, p. 1429. ISBN: 9781450367936. DOI: 10.1145/3328778.3372711. URL: https://doi.org/10.1145/3328778.3372711.

[MSR21] Mostafa Mohammed, Clifford A. Shaffer, and Susan H. Rodger. "Teaching Formal Languages with Visualizations and Auto-Graded Exercises". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 569–575. ISBN: 9781450380621. DOI: 10.1145/3408877.3432398. URL: https://doi.org/10.1145/3408877.3432398.

[Rod06] Susan H. Rodgers. *JFLAP – An Interactive Formale Language and Automata Package*. Boston: Jones and Bartlett Publishers, 2006. URL: http://jflap.org.

[Rol22] Lennart Rolfes. "Neue Frageverhalten für Aufgaben zur automatisierten Programmbewertung". Bachelor's Thesis. Hochschule Hannover, 2022. URL: https://doi.org/10.25968/opus-2371.

[RRE24] Eliot Wong Robson, Sam Ruggerio, and Jeff Erickson. *FSM Builder: A Tool for Writing Autograded Finite Automata Questions*. 2024. arXiv: 2405.01717 [cs.CY]. URL: https://arxiv.org/abs/2405.01717.

[SG06] Christopher J. Sangwin and Michael Grove. "STACK: addressing the needs of the neglected learners". In: *Proceedings of the Web Advanced Learning Conference and Exhibition, WebALT*. 2006, pp. 81–96. URL: https://stack-assessment.org.

[She+14a] Vinay Shekhar et al. “JFLAP Extensions for Instructors and Students”. In: *2014 IEEE Sixth International Conference on Technology for Education*. 2014, pp. 140–143. DOI: 10.1109/T4E.2014.22.

[She+14b] Vinay S. Shekhar et al. “Enhancing JFLAP with automata construction problems and automated feedback”. In: *2014 Seventh International Conference on Contemporary Computing (IC3)*. 2014, pp. 19–23. DOI: 10.1109/IC3.2014.6897141.

[Sie24] Andreas Siebel. *DFA checking using the automata library*. 2024. URL: https://coderunner.org.nz/mod/forum/discuss.php?d=308&lang=de.

[Son22] Mathias Sonderfeld. “GraFLAP - LMS-unabhängige Bewertung von JFLAP-Aufgaben”. Bachelor’s Thesis. Hochschule Hannover, 2022. URL: https://doi.org/10.25968/opus-2297.

[Spr16] Frauke Sprengel. *E-Assessment in Theoretical Computer Science using JFLAP*. Tech. rep. Hochschule Hannover, 2016. URL: https://doi.org/10.25968/opus-2229.

[SR17] Frauke Sprengel and Oliver Rod. “Integration automatisierter Programmbewertung in LON-CAPA”. In: *Automatisierte Bewertung in der Programmierausbildung*. Ed. by Oliver J. Bott et al. Münster, New York: Waxmann, 2017, pp. 295–310. URL: https://www.waxmann.com/automatisiertebewertung.

[Str+15] Sven Strickroth et al. “ProFormA: An XML-based exchange format for programming tasks”. In: *eleed* 11.1 (2015). ISSN: 1860-7470. URL: http://nbn-resolving.de/urn:nbn:de:0009-5-41389.

[Tos13] Ufuk Tosun. “Automatische Bewertung von Studierendenabgaben in der theoretischen Informatik auf Basis von JFlap”. Bachelor’s Thesis. Hochschule Hannover, 2013.