Article

Automatic Generation of Moodle Cloze Questions for the Assessment of Knowledge About Lexical Analysis Algorithms

Roberto Izquierdo-Amo [1], José Antonio Barbero-Aparicio [2], José Luis Garrido-Labrador [2], Alicia Olivares-Gil [2] and César Ignacio García-Osorio [2,*]

[1] Data and Analytics/Data Science Team, Zooplus SE, Calle de Génova 17, 28004 Madrid, Spain; robertoia1987@gmail.com
[2] Departamento de Ingeniería Informática, Universidad de Burgos, Avda. Cantabria s/n, 09006 Burgos, Spain; jabarbero@ubu.es (J.A.B.-A.); jlgarrido@ubu.es (J.L.G.-L.); aolivares@ubu.es (A.O.-G.)
* Correspondence: cgosorio@ubu.es; Tel.: +34-947-112457

Abstract: Moodle quizzes are a convenient way of online assessment, benefiting both teachers and students. Unfortunately, their preparation is slow, cumbersome, and error-prone. In addition to the effort of designing the questions, it is necessary to enter them in Moodle. Furthermore, for questions that require images, these must first be created and then uploaded to Moodle. If this process has to be repeated with many questions, the required work becomes prohibitive. This paper presents a tool that solves all these problems, allowing the automatic and massive generation of questions for the evaluation of the knowledge about lexical analysis algorithms. The evaluation of these algorithms is relevant in the subjects of both automata and formal languages and in those of compiler design. The tool allows the creation of exercises of configurable complexity, after which the text, tables, and associated images are generated for quick upload to Moodle. The main impact of this tool is the substantial reduction of question preparation time, reducing hundreds or thousands of interactions with Moodle forms to just a few simple steps. In addition, the ease of question generation makes it possible to prepare self-assessment questionnaires for the students, something that they greatly appreciate as a study aid.

Keywords: Moodle quizzes; continuous assessment; cloze questions; quiz generation; finite state automata; regular expressions; genetic programming;

check for updates

Academic Editor: João Piedade

Received: 16 November 2024
Revised: 31 December 2024
Accepted: 8 January 2025
Published: 13 January 2025

Citation: Izquierdo-Amo, R., Barbero-Aparicio, J. A., Garrido-Labrador, J. L., Olivares-Gil, A., & García-Osorio, C. I. (2025). Automatic Generation of Moodle Cloze Questions for the Assessment of Knowledge About Lexical Analysis Algorithms. *Education Sciences*, *15*(1), 75. https://doi.org/10.3390/educsci15010075

1. Introduction

Currently, universities around the world are widely adopting online learning systems. These systems, known as Learning Management Systems (LMSs), allow for digital interaction between teachers and students, revolutionizing educational dynamics. During the terrible COVID-19 pandemic, the adoption and use of LMSs has proven to be a crucial and timely solution for educational institutions (Veluvali & Surisetti, 2022). These systems make it possible to centralize learning resources, allowing 24/7 access; in addition, they capture data on student engagement, progress, and performance, which can be used to tailor instructional strategies and interventions for individual learners. An indication of the success of these systems is the number of different options on the market (Kasim & Khalid, 2016; Xin et al., 2021): Moodle, Sakai, ATutor, Blackboard, SumTotal.

One of the most popular LMSs is Moodle (Nash & Rice, 2018). This platform has gained wide acceptance in the academic field due to its user-friendly design, flexibility, and complete set of features, but above all, for being open-source and supported by a large community of users and developers who contribute to its improvement with plugins and help forums for troubleshooting and sharing best practices. As with most LMSs, Moodle makes it easy to create an online environment where teachers can share study materials, such as readings and videos, in a format accessible to students. Moodle offers additional tools that enrich the educational experience. For example, teachers can design quizzes and exams online, which speeds up assessment and reduces paper consumption.

Moodle quizzes are a convenient way to assess the knowledge and skills acquired by students (López-Tocón, 2021). They can be used both for continuous evaluation and for the final evaluation of subjects. Moodle offers a wide range of question types, from the simplest, such as short answer, true or false, multiple choice, or numerical or matching questions, to the most complex and sophisticated, such as calculated, click-and-drag text or images, or cloze questions. The latter are the most flexible, allowing several answer types to be combined in the same question, such as fill-in-the-blank, numerical answer, single answer selection, or multiple choice of several answers; however, their preparation is also the most complex.

The versatility of cloze questions makes them especially relevant in subjects like compiler design and formal languages, where the intricate nature of the algorithms fundamental to these fields requires more than simple true/false or multiple choice questions for effective assessment. However, the complexity involved in preparing cloze questions can be daunting, given the time and effort required for manual creation. Here, a tool like PLQuiz offers invaluable help.

PLQuiz is specifically designed to streamline the generation of cloze questions, particularly for evaluating knowledge related to the algorithms necessary in the lexical analysis process. Without this automation, educators might be tempted to reserve the limited number of questions they can manually prepare exclusively for subject assessments, preventing the greater use of questionnaires as an educational tool. But with PLQuiz, it is very easy to generate as many quizzes as necessary without much extra work. These quizzes can be shared with students as a useful self-study tool. Students can review them at their own pace, identifying areas where they need more practice. This allows students to take control of their learning and improve their understanding of these challenging subjects. The availability of these additional questionnaires could even have a motivating component for students, as the challenge of solving them could encourage them to devote more time to studying the subject.

The following two subsections provide more information on cloze questions and on the lexical analysis algorithms for which PLQuiz generates questions automatically; the third one presents some related works.

1.1. Moodle Quizzes

When creating questions for Moodle quizzes, the most immediate option is to use the built-in forms that Moodle provides for this purpose. The primary advantage is that educators are already familiar with the Moodle interface, making it an accessible option regardless of their level of technical expertise. It is a perfectly valid choice when the number of questions to be created is not very high. For instance, it could be suitable for crafting small quizzes that restrict access to new content until it is verified that fundamental concepts of previous content have been thoroughly understood. In this context, there is no issue if the questions remain the same in each attempt, as the goal is to encourage students to pay special attention to the concepts considered most relevant. But beyond the creation of small quizzes, or when one want to create quizzes for evaluation, this approach falls short.

For evaluation quizzes, if they exclusively use multiple choice questions, much more productive than Moodle forms is to use text files in Aiken format (MoodleDocs, 2022a).

This is a very simple format, which includes just the text of the question followed by the possible answers, preceded by a letter followed by a period or closing parenthesis; the letter for the correct answer is specified at the end, just after the keyword 'ANSWER'. Figure 1 shows an example. With this format, creating questions is very fast since one only has to write the question and answers. It allows instructors to focus on the content instead of dealing with cumbersome interaction sequences with graphical interfaces. Additionally, they can work offline and use their favorite text editor, taking advantage of features like spell checking to improve the overall quality of questions.

```
What is the purpose of a finite automaton?
A. To recognize regular languages
B. To generate context-free languages
C. To parse context-sensitive languages
D. To define recursively enumerable languages
Answer: A
```

Figure 1. Example of a question in Aiken format.

While Aiken is great for simple multiple choice questions with only one correct answer, it has its limits. If you need questions where more than one answer could be correct or if you want to use other question types, such as true/false or matching, the Aiken format is not sufficient. There is another more powerful text-based format, the GIFT format (MoodleDocs, 2022b). With this format, you can create multiple response, true-false, short answer, matching, and essay questions. In addition, students can be given feedback, or penalized when the chosen question is not correct, as shown in Figure 2. But this power comes at the cost of a slightly more complex syntax. Additionally, there are still some question types, such as cloze questions, that cannot be defined in GIFT. It is also not possible to include images with GIFT unless using an additional Moodle plugin.

```
$CATEGORY: $course$/top/GIFT/Examples

::Just the question name::Which one of the following regular expressions
represents the set of all binary strings with an odd number of 1s? {
  =10*(0*10*10*)*#Well done!
  ~((0 | 1)*1(0 | 1)*1)*10*#Actually, the language represented by this
  RE includes the string 1111 that has an even number of 1s.
}

::Multiple choice question::Which of the following strings belong to
the language defined by the regular expression (a|(b|ε)c)*a*b {
  ~%-20%aaaa
  ~%-20%acac
  ~%50%bcbccab
  ~%50%ccccb
}

::Matching question::Match the following acronyms with its meaning.{
  =DFA -> Deterministic Finite Automaton
  =NFA -> Non-deterministic Finite Automaton
  =PDA -> Pushdown Automaton
}
```

Figure 2. Example of several questions in GIFT format.

There is a third format, also textual, that covers all the shortcomings of the two previous formats, Moodle XML (MoodleDocs, 2024). It is the most versatile format for creating questions for Moodle. However, it is not designed to be directly used by humans, but rather to be used as an interchange format. This means that it is not very user-friendly or easy to read and write. It requires a significant amount of tags, attributes, and values to define each question and its elements (see Figure 3). It also has a strict structure and syntax that must be followed exactly, otherwise the file will not be valid and will not be imported by Moodle. Therefore, it is not recommended to use the Moodle XML format manually, although it is an ideal format for automatic question generation tools. In fact, it is the format chosen in PLQuiz to import the questions it generates into Moodle. This format can handle all the question types available in Moodle, as well as all aspects related to feedback, grading, and categorization of questions. It is also possible to embed media files directly into the XML file using Base64 encoding. This can make the XML file more portable and self-contained.

```
<question type="multichoice">
  <name><text>Just the question name</text></name>
  <questiontext format="moodle_auto_format">
      <text>Which one of the following regular expressions represents
          the set of all binary strings with an odd number of 1s?</text>
  </questiontext>
  <generalfeedback format="moodle_auto_format">
    <text></text>
  </generalfeedback>
  <defaultgrade>1.0000000</defaultgrade>
  <penalty>0.3333333</penalty>
  <hidden>0</hidden>
  <idnumber></idnumber>
  <single>true</single>
  <shuffleanswers>true</shuffleanswers>
  <answernumbering>abc</answernumbering>
  <showstandardinstruction>0</showstandardinstruction>
  <correctfeedback format="moodle_auto_format">
    <text></text>
  </correctfeedback>
  <partiallycorrectfeedback format="moodle_auto_format">
    <text></text>
  </partiallycorrectfeedback>
  <incorrectfeedback format="moodle_auto_format">
    <text></text>
  </incorrectfeedback>
  <answer fraction="100" format="moodle_auto_format">
    <text>10*(0*10*10*)*</text>
    <feedback format="moodle_auto_format"><text>Well done!</text></feedback>
  </answer>
  <answer fraction="0" format="moodle_auto_format">
    <text>((0 | 1)*1(0 | 1)*1)*10*</text>
    <feedback format="moodle_auto_format">
        <text>Actually, the language represented by this RE
            includes the string 1111 that has an even number of 1s.</text>
    </feedback>
  </answer>
</question>
```

Figure 3. Example of a question in Moodle XML format. Note that this is just a single question, specifically the first question in Figure 2.

1.2. Lexical Analysis Terminology and Algorithms

This section provides a quick overview of the background knowledge needed to understand the algorithms implemented in PLQuiz; for a more in depth treatment, some references that can be consulted are (Esparza & Blondin, 2023; Linz & Rodger, 2022).

In the context of formal languages, an **alphabet** is nothing more than a finite and non-empty set of symbols, and a **language** is simply a set, finite or infinite, of sequences of symbols in an alphabet. These sequences are called strings. The structure of these sequences and of these languages (arrays of strings) can be simple or more complex, making their generation and recognition more immediate or more difficult. The simplest languages to work with are the so-called **regular languages**. These are defined by direct enumeration of their strings, if they are finite, or by combining other regular languages by applying only three operations:

- **Union**: The union combines elements from two sets to create a new set that contains all distinct elements from both sets. In formal language theory, union is used to combine languages, resulting in a language that contains all strings present in either of the original languages.

$$L_1 \cup L_2 = \{w : w \in L_1 \vee w \in L_2\}$$

- **Cartesian product**: The Cartesian product uses elements of two sets to form pairs of all possible combinations. In formal language theory, the Cartesian product of two languages, also known as language concatenation, is the language that results from concatenating all strings of the first language with all strings of the second language.

$$L_1 \cdot L_2 = \{s_1 s_2 : s_1 \in L_1 \wedge s_2 \in L_2\}$$

  Repeated concatenation of the same language can be represented by a superscript that indicates the number of times the language is concatenated with itself. In this way, L^3 would represent the concatenation $L \cdot L \cdot L$.
- **Kleene Closure**: Kleene Closure, also known as the **star operation** or the **star closure**, takes a set or language and produces a new set or language that includes all possible combinations of elements, including repetitions. In formal language theory, applying the Kleene closure to a language generates a new language containing all possible strings that can be formed by concatenating elements from the original language, including the **empty** string, denoted by ε.

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup \bigcup_{i=1}^{\infty} L^i = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L^i$$

Regular expressions (REs) are a compact and powerful way of representing regular languages. The union of languages is represented by the selection operator '|', the closure is represented by the operator '*', and the concatenation is not represented explicitly (although some authors use '·'). For example, the regular expression $(cc)^*c|a(a|b)^*$ represents a language over the alphabet $\Sigma = \{a, b, c\}$ that includes all strings containing an odd number of the letter c or strings composed of the letters a and b that start with a. In the context of compiler design, regular expressions are used in the lexical analysis phase as a concise way to describe the elements of a programming language, such as reserved words, integers, floating point numbers, or identifiers, that will be converted to the tokens used in the subsequent phases. Regular expressions are also used in editors and command line utilities to perform tasks like string searching and manipulation.

In order to find the strings described by a regular expression, internally all these tools build a finite automaton from the regular expression. **Finite automata** are recognition devices that are characterized by having a finite number of **states** and **transitions** between these states. They have a single **initial state** and one or more **final states**. The transitions are labeled with one or several symbols, and these transitions are followed whenever one of the symbols labeling the transition is found in the input being analyzed. To recognize a string, the automaton starts from a configuration in which the only active state is the initial state and reads the symbols of the string in sequence, changing the active states according to the transitions. The string belongs to the language recognized by the automaton if, after reading the last symbol of the string, the automaton has reached a configuration with one or more active final states. There are two types of automata (see Figure 4):

- **Deterministic finite automata** (DFA): In a DFA, each state has exactly one transition for every input symbol, ensuring deterministic behavior. This means that, given the current state and an input symbol, the next state is uniquely determined. At any point in time, the DFA can only be in a single state. If, after reading the entire input string, the DFA is in a final state, the string is accepted by the automaton, meaning that it belongs to the regular language recognized by the DFA. Simulating a DFA to recognize strings belonging to a regular language is very efficient (linear complexity in the size of the string), but the size of the DFA can be prohibitive, potentially becoming exponential in the length of the regular expression.

- **Nondeterministic finite automata** (NFA): In an NFA, for a state and input symbol, there can be multiple transitions to different states or even **ε-transitions** (transitions that can be followed without reading any symbol from the input). This non-deterministic behavior allows for more flexibility in handling transitions. On the other hand, this non-determinism is as if the automaton could have several active states at any given moment. The acceptance of a string occurs when, after completing its reading, a final state is found among the potentially active states, verifying at this moment that the string belongs to the regular language recognized by the NFA. The recognition of regular languages with NFAs is less efficient than the use of DFAs, since the simulation of NFAs has a complexity that is proportional to the product of the length of the string times the length of the regular expression, but they have in their favor that their size does not have a worse case as bad as the DFAs; in fact, their spatial complexity is linear in the size of the regular expression.

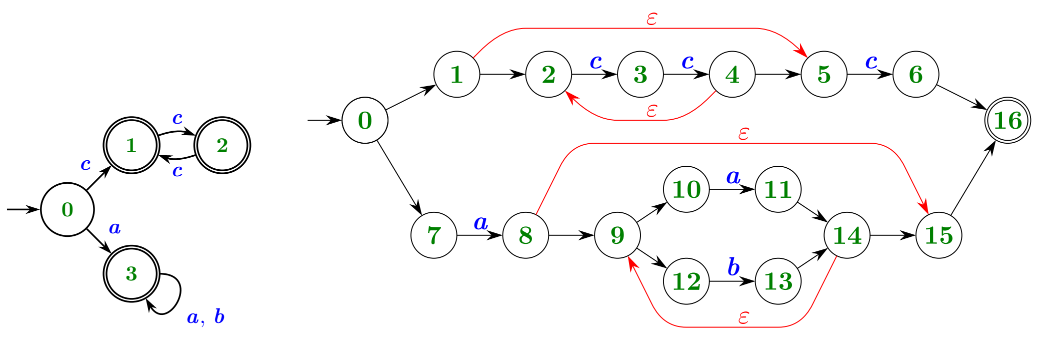


Figure 4. Examples of finite automata for recognition of the language represented by the regular expression $(cc)^*c|a(a|b)^*$. The left one is a DFA, the right one is a NFA with ε-transitions.

Both types of automata possess the same capability to recognize regular languages: they can recognize any regular language, and all the languages they recognize are regular. Every NFA has an equivalent DFA that recognizes the same language, and conversely, every DFA can be seen as a specialized NFA. This relationship between regular expressions, NFA, and DFA is materialized in the existence of various algorithms that allow us to obtain DFA and NFA from a regular expression, or obtain a DFA from an NFA. Those for which PLQuiz generates questionnaires are the following:

- The algorithm proposed by McNaughton, Yamada, and Thompson (McNaughton & Yamada, 1960): This algorithm constructs an NFA from a regular expression by recursively analyzing its constituent parts (sub-expressions). It creates an NFA for each sub-expression and combines them using ε-transitions, taking into account the operator that joins them. In the base case, the sub-expression is a symbol or epsilon, and the NFA will have only two states, an initial state joined to the final state with a transition labeled with the symbol or epsilon. Now, if two regular expressions, α and β, are combined, for which the corresponding NFAs are already available, $N(\alpha)$ and $N(\beta)$, the new NFA will be constructed combining these using ε-transitions and adding new states, or merging existing ones in some cases, according to the rules shown in Figure 5.

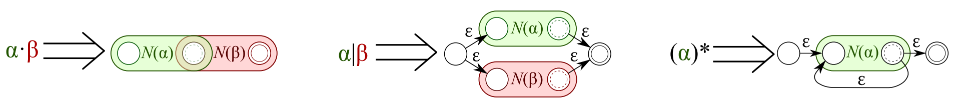


Figure 5. How to combine the NFAs $N(\alpha)$ and $N(\beta)$ that recognize the languages described by the regular expressions α and β to obtain a new NFA to recognize $\alpha \cdot \beta$, $\alpha|\beta$, and $(\alpha)^*$.

- The power-set construction algorithm (Rabin & Scott, 1959): This method allows the transformation of an NFA into a DFA. Since the NFA has a finite number of states, the possible combinations of active states will also be finite (some of these combinations may not occur in practice due to the transition function structure). The idea is to associate each state in the DFA with each of these combinations of states in the NFA and build a transition function that reflects how these combinations change in the original NFA. It is as if the behavior of the NFA was "simulated" with the DFA. Combining the McNaughton, Yamada, and Thompson algorithm (MYT) with the power-set construction algorithm, it is possible to obtain a DFA from a regular expression.
- Lastly, PLQuiz implements the algorithm described by Aho, Sethi, and Ullman in "*the dragon book*" (Aho et al., 1986) (ASU) to create a DFA from the regular expression, eliminating the need for an NFA. This approach uses the syntax tree representation of a regular expression to compute in sequence several functions that capture essential properties of the regular expression:
  - The function *nullable* determines if a sub-expression can match an empty string. It is true for ε leaf nodes and closure nodes. For concatenation nodes, it is true if both children are nullable, and for selection nodes, if any child is nullable.
  - The functions *first-pos* and *last-pos* identify the positions where sub-expression strings can start and end, respectively. For leaf nodes, they hold the position of the symbol at the leaf or an empty set for leaves with ε. For selection nodes, they are the union of the corresponding values of their children. For Kleene closure nodes, they inherit the values of their children for the corresponding function. And for concatenation nodes, *first-pos* is just *first-pos* of its left child if it is not nullable, or the union of its children's values if the left child is nullable. The determination of *last-pos* follows a similar logic, although now it is the nullability of the right child that has to be considered.
  - The function *follow-pos*, which is constructed only from concatenation and closure nodes, reveals positions that can follow others. In concatenation nodes, the positions in the *first-pos* of the right child add to the *follow-pos* of the positions in the *last-pos* of the left child. In closure nodes, the positions in the *last-pos* of the child add to the *follow-pos* of the positions in the *first-pos* of the child.

  Finally, the *follow-pos* function is used to construct the DFA transition function. The states in the automaton are sets of positions, and the initial state consists of the positions in the *first-pos* of the root node. Now, for a state S, transitioning at symbol a leads to the state $f(S, a)$, which consists of the union of all *follow-pos*(p) for positions p in S, corresponding to occurrences of a in the regular expression.

1.3. Related Works

Teachers of courses in automata, formal languages, and compilers are well aware of how challenging these subjects are for students (Chesnevar et al., 2004a; Chesnevar et al., 2004b; Knobelsdorf et al., 2014). As a result, many tools for visualizing algorithms have been developed (Chakraborty et al., 2011; Chesnevar et al., 2003; Stamenkoviæ & Jovanoviæ, 2021), including not only those related to lexical analysis—for which PLQuiz generates questionnaires—but also tools for algorithms in subsequent phases, including syntactic analysis (Castro-Schez et al., 2021; Muñoz et al., 2024), semantic analysis (Golemanov & Golemanova, 2020; Steingartner, 2021), and code generation (Demaille et al., 2008; Steingartner & Sivý, 2023).

Among the tools for the lexical analysis phase, the most prominent is JFLAP (Rodger & Finley, 2006), which is written in Java and is very popular and comprehensive. It allows the construction and visualization of not only finite automata, but also other recognizing devices such as push-down automata and Turing machines. It also provides the ability to convert a regular expression into a finite automaton, although it only provides one algorithm for this, which is different from the one implemented in PLQuiz.

Another tool in this category is THOTH (García-Osorio et al., 2007; García-Osorio et al., 2008), which, like JFLAP, allows the graphical creation of finite automata and implements a broader set of algorithms to convert regular expressions into finite automata. It provides algorithms such as Aho–Sethi–Ullman, McNaughton–Yamada–Thompson, and the subset construction mentioned in the previous section, along with a third algorithm based on the derivatives of regular expressions.

While the previous tools are desktop applications, there are also web-based tools that can be used directly from a web browser. One such tool is SELFA-Pro (Gallardo Casero et al., 2016), which is based on a domain-specific language to describe regular expressions, automata, grammars, and operations on them. Another example is Seshat (Arnaiz-González et al., 2018), which allows a step-by-step visualization of some algorithms for converting regular expressions into finite automata. Although JFLAP is a desktop application, its code base has recently been refactored and reused to create a web application known as OpenFLAP (Mohammed et al., 2021), which also includes an auto-grade module. Automata Tutor v3 (D'Antoni et al., 2020) is a further development of an earlier tool, which initially supported only automata construction problems, but the new version expands its scope to include regular expressions, grammars, push-down automata, and Turing machines, while also introducing automatic problem generation. AutomaTutor (Jordaan et al., 2024a, 2024b) is another web application that works with finite automata and regular expressions. It offers exercises where users are asked to transform one into another, treating these as challenges that rely on the user understanding and intuition of automata and regular expressions rather than prescribing a specific transformation algorithm. Designed with a mobile-first approach and optimized for touch-screen interaction, AutomaTutor is particularly well suited for use on smartphones and tablets.

In addition to these web applications designed with the mobile first philosophy, there are applications specifically designed to run natively on mobile devices, such as CMSimulator (Chuda et al., 2015) and Automata Simulator (Singh et al., 2019), both of which focus on constructing, simulating, and manipulating finite automata rather than generating them from regular expressions.

There are also interactive tools that help to teach the phases after lexical analysis, such as LLparse and LRparse (Blythe et al., 1994), BURGRAM (García-Osorio et al., 2008), Proletool 3.0 (Castro-Schez et al., 2021), ComVis (Stamenković & Jovanović, 2023), and SIETTE (Muñoz et al., 2024), which provide simulations of parsing algorithms and visualization of LL and LR parsing processes.

These tools represent only a selection of the applications available to facilitate the learning of formal language concepts and compilers. For a more comprehensive review, the interested reader can refer to (Stamenković et al., 2020). However, even this review does not mention tools such as PLQuiz, which is designed to automatically generate Moodle quizzes on automata and regular expressions. Although other tools, such as OpenFlap, Automata Tutor v3, Proletool 3.0, ComVis, and SIETTE, include an assessment component, the quizzes are tool-specific and cannot be exported to Moodle, which means that the possibilities for integrating the results of these quizzes into the assessment of Moodle-based courses are limited. Furthermore, the generation of exercises for use in face-to-face assessment with written exams is also not possible.

Although we have not found any other tools for the specific creation of questionnaires on automata and formal languages for the Moodle platform, there are some general tools and libraries that aim to simplify questionnaire creation in Moodle, or focus on the creation of specific questionnaires in other areas of knowledge:

- *Libre-gift* (https://github.com/clopezno/libre-gift, accessed on 7 January 2025): A LibreOffice template to create questionnaires in this word processor that are then exported to formats that can be imported from Moodle.
- *MoodleQUIZ_template_UVa* (https://github.com/juacas/MoodleQUIZ_template_UVa, accessed on 7 January 2025): Another template, in this case for Word, that facilitates the editing of questions that are then exported to the Moodle GIFT format.
- *Moodle Cloze and GIFT Code Generator* (https://hbubecc.wixsite.com/jordan/tools, accessed on 7 January 2025): A macro that allows questions to be edited using Excel and then exported into GIFT format. This tool is presented in (Svien, 2019).
- *GIFT Quiz Editor* (https://workspace.google.com/marketplace/app/gift_quiz_editor/1038395345285, accessed on 7 January 2025): An add-on for Google Forms that allows them to be exported to GIFT.
- *Moodle-questions* (https://github.com/gethvi/moodle-questions, accessed on 7 January 2025): A library for manipulating Moodle questionnaires in Python.
- *Pygiftgenerator* (https://gitlab.com/EHU/pygiftgenerator, accessed on 7 January 2025): Another Python library, this time specifically designed for the automatic generation of physics problems. This library is presented in (Sáenz et al., 2020).
- *GIFTGenerator* (https://www.giftgeneratorapp.com/, accessed on 7 January 2025): A commercial tool for generating questionnaires in GIFT format.
- *Typeform* (https://www.typeform.com/quizzes/, accessed on 7 January 2025): Another commercial tool to facilitate the creation of quizzes.
- R/Exams (http://www.r-exams.org, accessed on 7 January 2025): A package for the R system focused on the automatic generation of exams that can be imported into Moodle. Its development is quite active, and it has a significant user base.
- *Incrustada* (https://www.escinf.una.ac.cr/discretas/Archivos/Packages/Paquete_Cloze.zip, accessed on 7 January 2025): A package for Wolfram Mathematica to facilitate the automatic generation of questions and its subsequent export to Moodle. Like PLQuiz, it is focused on generating cloze-type questions (the most complex and versatile in Moodle) and also automatically generates tables if the question needs them. This package is presented in (Quesada & Herrera, 2024).

2. Materials and Methods

2.1. Overview of PLQuiz

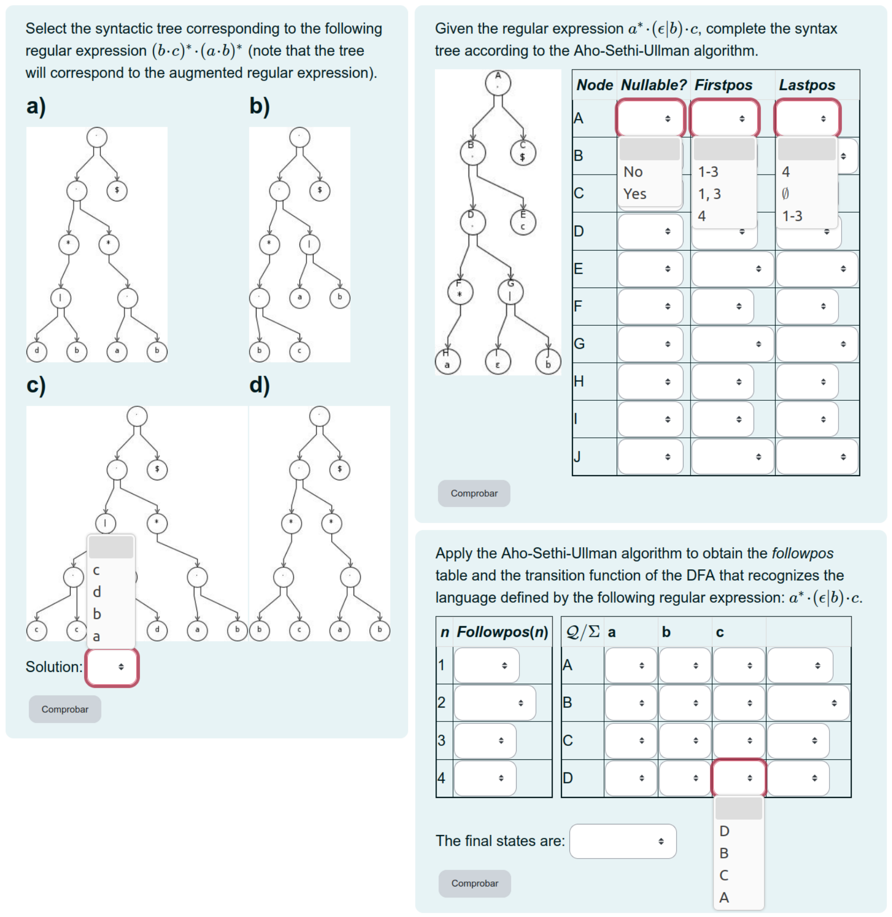
PLQuiz is a software tool developed in Java, ensuring cross-platform compatibility across major operating systems. The source code is openly available on GitHub at https://github.com/RobertoIA/PLQuiz. The application supports multilingual functionality, adapting its interface and quiz generation features to the host machine's language settings, with built-in support for English and Spanish.

Specifically, it generates various exercises of increasing complexity for inclusion in questionnaires to evaluate two procedures for transforming regular expressions into finite automata that recognize the regular languages they represent:

- Application of the ASU algorithm to directly obtain a DFA from the RE. The exercises for this procedure include the following:
  - Identifying the syntactic tree corresponding to the RE.
  - Filling in the values for the *nullable*, *first-pos*, and *last-pos* functions at different nodes of a syntactic tree of a given RE.
  - Deriving the *follow-pos* table and the transition function of the DFA resulting from the ASU algorithm.

  Figure 6 shows how these exercises are presented to the students in Moodle.
- Application of the MYT algorithm first to obtain an NFA from an RE, followed by application of the subset construction algorithm to transform the NFA into a DFA. The exercises for this procedure include the following:
  - Identifying the NFA corresponding to the RE.
  - Deriving the transition function of a DFA from the transition diagram of the DFA.
  - Obtaining the transition function of a DFA resulting from the successive application of the MYT and subset construction algorithms.

  Figure 7 shows how the exercises in this conversion approach are presented to the students in Moodle.


Select the syntactic tree corresponding to the following regular expression $(b \cdot c)^* \cdot (a \cdot b)^*$ (note that the tree will correspond to the augmented regular expression).

Solution:

Given the regular expression $a^* \cdot (\epsilon | b) \cdot c$, complete the syntax tree according to the Aho-Sethi-Ullman algorithm.

Node	Nullable?	Firstpos	Lastpos
A			
B			
C			
D			
E			
F			
G			
H			
I			
J			

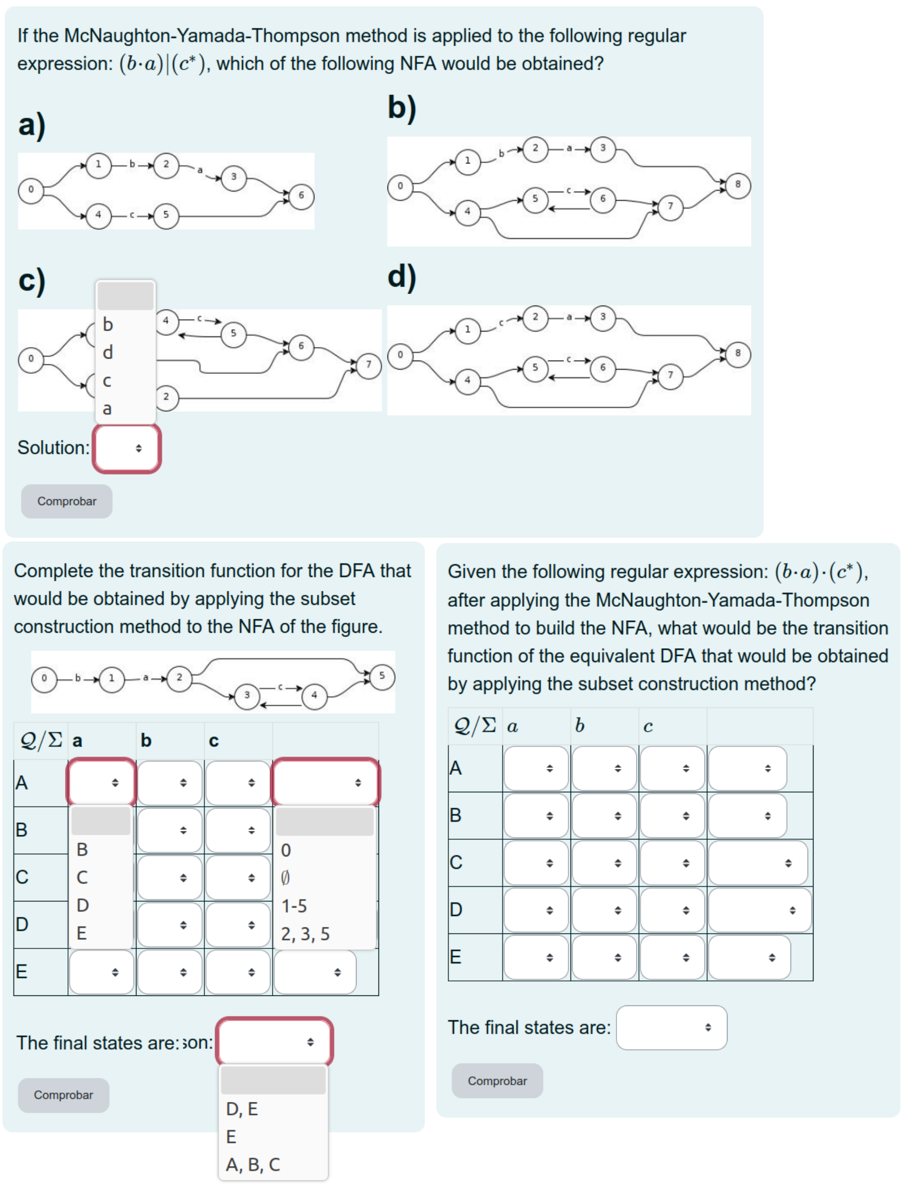
Apply the Aho-Sethi-Ullman algorithm to obtain the *followpos* table and the transition function of the DFA that recognizes the language defined by the following regular expression: $a^* \cdot (\epsilon | b) \cdot c$.

n	*Followpos(n)*
1	
2	
3	
4	

Q/Σ	a	b	c	
A				
B				
C				
D				

The final states are:

Figure 6. The figure shows what the different types of exercises for the ASU algorithm look like. At the top left, the simplex of the three types of exercise, consisting of selecting the syntax tree that corresponds to the RE. At the top right, the exercise to label the nodes of the syntax tree with the values of the functions *nullable*, *first-pos*, and *last-pos*, an exercise of medium complexity. At the bottom, the exercise to obtain the function *follow-pos* and the transition function for the DFA that recognizes the language specified by the RE, the most complex of the three types of exercise for the ASU algorithm.



If the McNaughton-Yamada-Thompson method is applied to the following regular expression: $(b{\cdot}a)|(c^*)$, which of the following NFA would be obtained?

a) **b)** **c)** **d)**

Solution:

Complete the transition function for the DFA that would be obtained by applying the subset construction method to the NFA of the figure.

$\mathcal{Q}/\Sigma$	a	b	c	
A				
B				
C				
D				
E				

The final states are:

Given the following regular expression: $(b{\cdot}a){\cdot}(c^*)$, after applying the McNaughton-Yamada-Thompson method to build the NFA, what would be the transition function of the equivalent DFA that would be obtained by applying the subset construction method?

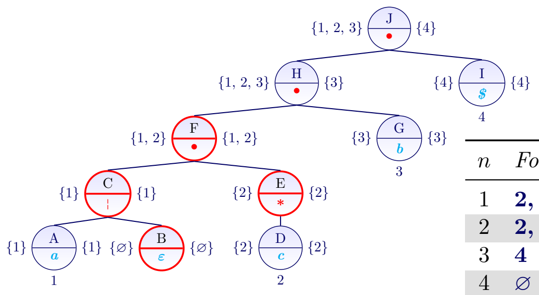
$\mathcal{Q}/\Sigma$	a	b	c	
A				
B				
C				
D				
E				

The final states are:

Figure 7. The figure shows what the different types of exercise look like for the MYT algorithm. At the top, a very simple exercise in which the student needs to select the NFA that corresponds to the regular expression. Below on the left, an exercise in which the student must apply the subset construction algorithm to the given NFA to obtain the transition function of the equivalent DFA. In the lower right part, a more complex exercise where the student needs to do the whole conversion process applying the MYT algorithm first, and then use the subset construction algorithm to obtain the transition function of the DFA that recognizes the language represented by the given RE.

Once generated, the quizzes can be exported to Moodle XML format for uploading to this platform for online assessment, or to LaTeX format for integration into documents intended for sit-in exams. The LaTeX documents can be compiled into PDFs, allowing for easy printing and distribution during face-to-face exams (see Figure 8).

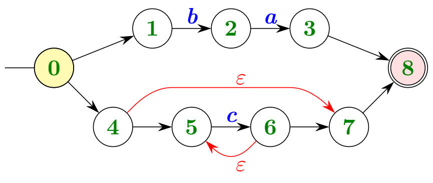
1.- Apply the Aho-Sethi-Ullman algorithm to obtain the *followpos* table and the transition function of the DFA that recognizes the language defined by the following regular expression: $(a|\epsilon) \cdot c^* \cdot b$



n	*Followpos*(n)
1	**2, 3**
2	**2, 3**
3	**4**
4	**∅**

Q	a	b	c	*Positions*
A	**B**	**C**	**B**	**1, 2, 3**
B	**D**	**C**	**B**	**2, 3**
(C)	**D**	**D**	**D**	**4**
D	**D**	**D**	**D**	**∅**

2.- Complete the transition function for the DFA that would be obtained by applying the subset construction method to the NFA of the figure.



Q	a	b	c	*NFA states*
(A)	**B**	**C**	**D**	**0, 1, 4, 5, 7, 8**
B	**B**	**B**	**B**	**∅**
C	**E**	**B**	**B**	**2**
(D)	**B**	**B**	**D**	**5, 6, 7, 8**
(E)	**B**	**B**	**B**	**3, 8**

Figure 8. Example of exercises in LaTeX to be used in sit-in exams. Two exercises are shown, one on obtaining an DFA from a regular expression (the tree and the tables are part of the response expected from the student) and another on obtaining an DFA from the transition diagram of an NFA (the finite automaton on the left is provided as part of the question, and the table on the right is the answer the student is expected to give). The solutions are also generated by PLQuiz, but they are only shown if the keyword `answer` is passed to the LaTeX class.

2.2. How to Use PLQuiz

The main elements of the PLQuiz GUI are shown in Figure 9; these are as follows:

- Menu bar (*a*): Located at the top; it offers standard options such as "File", "Edit", and "Help".
- Left panel (*b*): This panel is essential for question configuration. This is where the sub-panels will appear for each question that is added to the quiz. From this panel, it is possible to control the deletion, order, and display of questions.
- Dropdown menu for question creation (*c*): Located at the bottom of the left panel; this menu allows users to select the type of question they want to add, and whether it is related to the ASU or MYT algorithm.
- Right panel (*d*): This panel will show a preview of the generated question as it will appear in the quiz.

The process of creating a questionnaire in PLQuiz is simple and intuitive; the steps are as follows:

1. Add a question, using the drop-down menu at the bottom of the left panel (Figure 9c), where it is possible to select the type of problem for (ASU or MYT) for which the question will be generated. It is also possible to add these sub-panels with the last two options of the "File" menu. Each added question will have its own configuration sub-panel in the left panel.
2. Configure the questions in the corresponding sub-panels, with several common elements for both types of problem (see Figure 10). In these sub-panels it is possible to perform the following operations:
    - Delete the question (*a*).
    - Enter the regular expression in the text field (*b*) that will be used to generate the exercise.

- Select the type of exercise with the radio buttons (*d*) that will be generated for the type of problem.
- View the question (*c*): The right arrow button allows the question to be previewed in the left panel.
- Change the position of the questions (*e*): It is possible to reorder the questions using the up and down buttons.

3. Generate a regular expression (optional): Users can automatically generate a regular expression by clicking the "Generate" button (*f*). The complexity of the generated regular expression can be customized by adjusting the following parameters (*h*):
   - Include empty string (ε).
   - The number of symbols to be used in the regular expression.
   - The number of states of the resulting automaton.

   Blocks of questions can also be generated using the "File | Generate Blocks of Questions" option in the "File" menu (Figure 11 shows all available options).
4. View and export the questionnaire: The questionnaire can be reviewed in the preview panel on the right; it can then be exported either to Moodle XML format for later import into the Moodle platform, or to LaTeX format for printing and use in face-to-face exams.

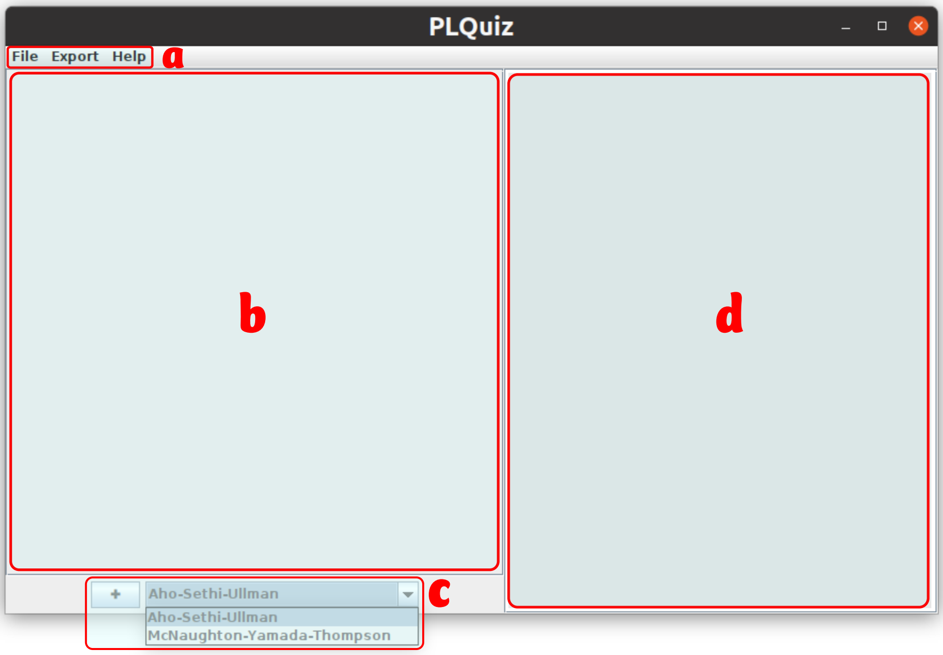


Figure 9. The main elements of the application: (**a**) menu bar, (**b**) left panel to show the question sub-panels, (**c**) drop-down to choose the algorithm type, (**d**) right panel for question visualization.

2.3. Some Technical Implementation Details

2.3.1. Generating Tree and Finite Automaton Diagrams

The transformation of regular expressions into visual representations is a multistep process that includes parsing, tree construction, and diagram generation. To facilitate the parsing of user-provided regular expressions, we employ JavaCC (Community, 2023; Copeland, 2007), a parser generator for Java. This allows us to convert the input string into a structured tree data format, which serves as the foundation for subsequent steps in the visualization process.

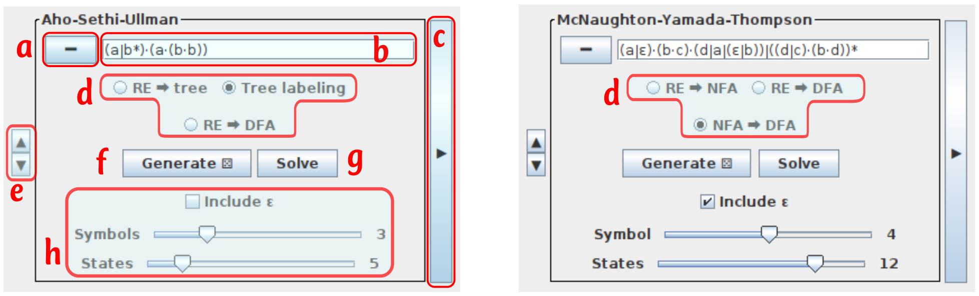
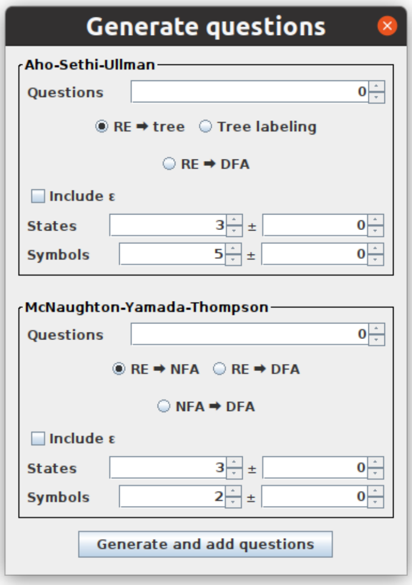


Figure 10. The sub-panels for question generation configuration. Left is for questions related to the ASU algorithm. Right is for questions related to the MYT algorithms. Button (a) is for question removal. Button (c) is to visualize the selected question in the right panel. Buttons in (e) are for question ordering. The RE is introduced in text field (b). The type of question is selected with the radio buttons in (d). To automatically generate a question, it is possible to use button (f) together with the controls in (h). Button (g) is to solve an exercise; with each click on this button, exercises are generated with different alternative answers to the correct one.


Generate questions

Aho-Sethi-Ullman

Questions 0

[✓ RE ➡ tree] Tree labeling

RE ➡ DFA

Include ε

States 3 ± 0

Symbols 5 ± 0

McNaughton-Yamada-Thompson

Questions 0

[✓ RE ➡ NFA] RE ➡ DFA

NFA ➡ DFA

Include ε

States 3 ± 0

Symbols 2 ± 0

Generate and add questions

Figure 11. Dialog window to configure the automatic generation of blocks of questions.

In the early stages of PLQuiz, JGraph (Alder, 2013), a Java library, was used to create syntax trees and finite automata diagrams for both the graphical interface and Moodle questionnaires. While JGraph served its purpose for these platforms, it fell short in terms of producing high-quality graphics when exporting exercises to LaTeX. This led us to explore alternatives for rendering syntax trees and automata diagrams that could deliver higher graphic quality.

As a solution for syntax trees, we turned to TikZ (Tantau, 2015), a powerful graphics package for LaTeX that enables precise and customizable illustrations within documents.

Within the TikZ ecosystem, we used two specialized libraries tailored for the creation of syntax trees: `tikz-qtree` and `tikz-qtree-compat` (Chiang, 2012). These libraries build upon the functionality of TikZ, offering a streamlined and user-friendly approach to designing hierarchical tree structures. Using these libraries and implementing specific node styles, we achieved significantly improved visual results (see the first exercise of Figure 8).

Unlike syntax trees, we faced a different challenge with NFA. We did not find a library that produces automata in the specific format used during class explanations. Therefore, we took a low-level approach using TikZ primitives. By explicitly calculating the coordinates for positioning nodes, arcs, and labels, we achieved the desired finite automaton representation (see the second exercise of Figure 8).

The process works in two stages. In the first one, the parser reads the regular expression and builds, bottom-up, a tree data structure with information about the height and width of the bounding box for the automaton associated with each sub-expression, along with the vertical distance of the initial and final states from the top of the bounding box. Using these values, in the second stage, the tree is traversed top-down, calculating the coordinates of several reference points that will be used with the low-level TikZ commands to draw the states, the transitions, and the labels. Figure 12 shows the template for the two base cases, a transition annotated with a symbol, and a ε transition. Figure 13 shows the template for the automata associated with a Kleene closure regular expression, and Figure 14 for the selection.

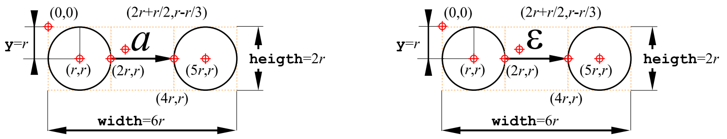


Figure 12. Drawing templates for the automata associated with the most basic regular expressions, just a symbol, of the empty string.

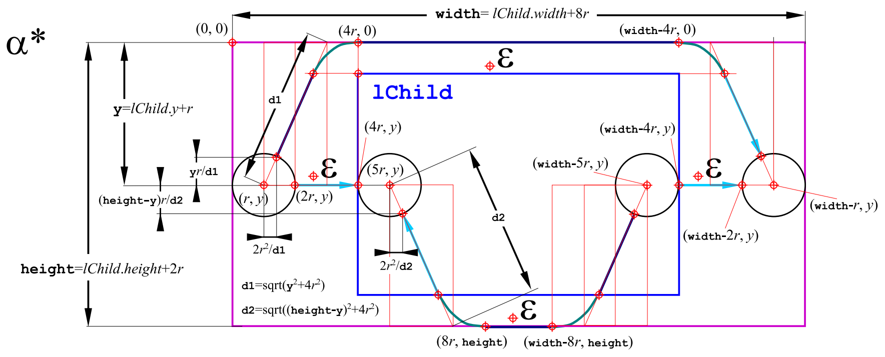
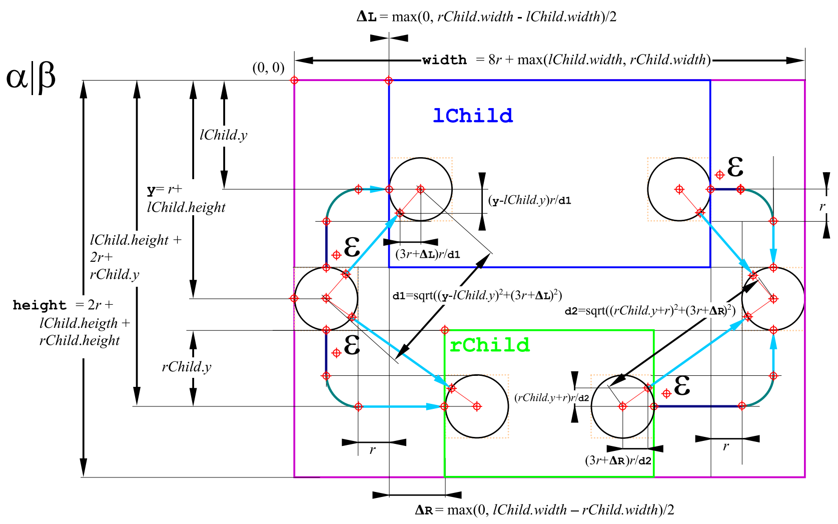


Figure 13. Drawing template for the automata associated with a regular expression, consisting of the Kleene closure of another regular expression.



ΔL = max(0, rChild.width - lChild.width)/2
width = 8r + max(lChild.width, rChild.width)
α|β
(0, 0)
lChild
lChild.y
ε
y= r+ lChild.height
(y-lChild.y)r/d1
r
lChild.height + 2r+ rChild.y
ε
(3r+ΔL)r/d1
height = 2r + lChild.heigth + rChild.height
d1=sqrt((y-lChild.y)²+(3r+ΔL)²)
d2=sqrt((rChild.y+r)²+(3r+ΔR)²)
ε
rChild
rChild.y
(rChild.y+r)r/d2
ε
r
(3r+ΔR)r/d2
r
ΔR = max(0, lChild.width – rChild.width)/2

Figure 14. Drawing template for the automata associated with a regular expression consisting of the selection of other two. Two possible approaches are shown with straight transitions and with curved transitions.

2.3.2. Regular Expression Generation

Although regular expressions are entered as strings, they are stored internally as binary trees. Tree generation is a well-studied operation in the context of genetic programming, so we can apply some of the existing techniques to this field; for example, initializing the starting population of regular expression syntax trees or mutating trees.

Initial Generation of Regular Expressions

In genetic programming, each individual is represented as a tree. The initial creation of these trees is done recursively from a set, N_{func}, of functions, $F = \{f_1, f_2, \ldots, f_{N_{\text{func}}}\}$, with a number of arguments or arities, $z(f_1), z(f_2) \ldots, z(f_{N_{\text{func}}})$, and from a set of N_{term} terminals of $T = \{a_1, a_2, \ldots, a_{N_{\text{term}}}\}$, therefore without children in the trees (Koza, 1992).

In the case of regular expressions, the functions in F are as follows: selection and concatenation of arity 2 and Kleene closure of arity 1. The set of terminals, T, for regular expressions consists of the symbols of the alphabet of the language for which the regular expression describes strings, together with the empty string ε (or only symbols of the alphabet if the use of ε has been excluded).

The automatic generation of regular expressions for ASU problems, as well as for the generation of MYT problems, initially uses the "full" method of creating trees (Koza, 1992). The "full" method builds trees in which all leaf nodes are at the same depth. To do this, in the construction process, the nodes take their operation from F if their depth is less than the maximum depth and from T if their depth is already the maximum depth (in contrast, in the "grow" method, the nodes take their label from $F \cup T$ if their depth is less than the maximum depth, and from T if they are already at the depth limit).

Regular Expressions for MYT and Subset Construction Exercises

To obtain regular expressions for this type of problem, the "full" method is first used to generate a small initial population of eight trees with a guaranteed minimum depth of 2, but where there may also be trees of depths 3 to 5. The initial population is created with a size equal to the sum of the elitism count (ELITISM = 2), the mutation count (MUTATION = 2), and the number of new solutions (NEW = 4).

Each regular expression in the population is used to obtain a DFA (by first obtaining an NFA using the MYT method and then obtaining a DFA using subset construction). The fitness score is calculated as the sum of the absolute values of the following:

1. The difference between the number of symbols in the regular expression and the number of symbols specified by the user.
2. The difference between the number of states in the obtained DFA and the number of states configured by the user.

A lower value indicates a better regular expression.

From the initial population, an evolutionary process is started, in which the population for the next generation is obtained by the following:

1. Keeping the two best regular expressions (*elitist subset*).
2. Mutating two other regular expressions. The mutation consists of replacing one of the subtrees of the regular expression with a new subtree.
3. Generating four new regular expressions with a depth that is a small variation of the depth of the best regular expression in the current population, but always within the limits of the minimum depth and the maximum depth.

This process continues until one of the following conditions occurs:

- A regular expression with a fitness score of 0 is found.
- A maximum number of iterations is reached (MAX_ITERATIONS = 500).

Although finding perfect fitness solutions is not guaranteed, this simple process produces satisfactory regular expressions for exercise generation.

Regular Expressions for the ASU Method

For the generation of exercises for the Aho–Sethi–Ullman method, acceptable regular expressions can be obtained using an even simpler process. Instead of using a population of candidate regular expressions, a single expression constructed using the "full" method is used as a starting point. A random search is then performed, where in each iteration a new regular expression is generated, the depth of which is adjusted depending on whether the previous regular expression has generated a DFA with more or fewer states than required. The best solution so far is retained throughout the process. As with the previous case, the search ends when a regular expression with perfect fitness is found or when the maximum number of iterations is reached (MAX_ITERATIONS = 3000), at which point the best regular expression found is returned.

2.4. Experimental Setup

2.4.1. Measuring Time in Manual Question Creation

To get a rough and objective idea of the time saved by using PLQuiz, we designed a small experiment focused on the process of manually typing questions within Moodle. Specifically, we wanted to validate the efficiency of directly importing Moodle-compatible XML files instead of manually entering questions into Moodle.

The experiment consisted of manually entering two types of question generated by PLQuiz.

- Question 1 (*select 1 of 4 images*): a "*Tree construction*" type exercise for Aho–Sethi–Ullman (option "RE ➡tree" in the ASU panel). In this exercise, the student is shown the starting regular expression and several trees from which they must choose the one that matches the regular expression shown. It involves loading four images: one correct and three distractors.
- Question 2 (*table of multi-choice selections*): a "*Getting the transition function from DFA*" type exercise using the MYT method and subset construction (option "RE ➡DFA" in the MYT panel). The student has to select the correct elements of the transition function represented with a table in which each cell is a multiple selection defined using Moodle's cloze syntax, where the correct answer and the distractors are provided.

The experiment was carried out by three of the authors of this article, all of them experienced Moodle users. Two of them were very familiar with the Moodle question bank, while the third was using it for the first time.

Each participant was provided with the following resources to complete the tasks:

- The text of the questions. Figure 15 shows the text for question 1, which is relatively short, with the main challenge being the process of uploading the images. Figure 16 shows the text for question 2, which is significantly more complex and requires the use of the 'MULTICHOICE' element for multiple-choice cloze selections.
- The images generated by PLQuiz for question 1.

The initial focus of the data collection was on two main metrics:

- Time: the time taken to manually enter each question into Moodle, from the start of the process to the final submission.
- Mouse interactions: the total number of clicks and actions taken during the editing process, which can also be considered a measure of the complexity of the process.

```
<p>Select the syntactic tree corresponding to the following regular
expression $$(b·c)*·(a·b)*$$ (note that the tree will correspond
to the augmented regular expression).
<table> <tbody>
    <tr>
      <td> <h2>a)</h2> <p>INSERT TREE1 IMAGE HERE</p> </td>
      <td> <h2>b)</h2> <p>INSERT TREE2 IMAGE HERE</p> </td>
    </tr>
    <tr>
      <td> <h2>c)</h2> <p>INSERT TREE3 IMAGE HERE</p> </td>
      <td> <h2>d)</h2> <p>INSERT TREE4 IMAGE HERE</p> </td>
    </tr>
</tbody> </table>
<p>Solution: {1:MULTICHOICE:~%-25%d~%100%c~%-25%a~%-25%b}</p>
```

Figure 15. Example of content for a question with four images. The text indicating where the images should be inserted are highlighted in bold. Also in bold is the cloze syntax for the multi choice answer.

```
<p>Given the following regular expression: $$(a|b)·(b|c)*·c*$$, apply
the McNaughton-Yamada-Thompson method and the subset construction method
to fill the transition table for the obtained NFA:</p>
<p><br></p> <table border="1" cellspacing="0" align="left"><tbody>
<tr>
  <th scope="col">$$\mathcal{Q}/\Sigma$$</th>
  <th scope="col">a</th> <th scope="col">b</th>
  <th scope="col">c</th> <th scope="col">&nbsp;</th></tr>
<tr> <td>A</td>
  <td>{1:MULTICHOICE:~%-25%D~%100%B~%-25%C~%-25%E}</td>
  <td>{1:MULTICHOICE:~%-25%E~%-25%F~%-25%D~%100%C}</td>
  <td>{1:MULTICHOICE:~%100%D~%-25%A~%-25%F~%-25%E}</td>
  <td>{1:MULTICHOICE:~%100%0, 1, 3~%-25%1, 3~%-25%2, 4, 5,
       6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}</td></tr>
<tr> <td>B</td>
  <td>{1:MULTICHOICE:~%-25%C~%100%D~%-25%A~%-25%B}</td>
  <td>{1:MULTICHOICE:~%-25%B~%-25%C~%-25%D~%100%E}</td>
  <td>{1:MULTICHOICE:~%-25%D~%100%F~%-25%E~%-25%C}</td>
  <td>{1:MULTICHOICE:~%-25%0, 1, 3, 4, 8, 10, 11, 16~%100%2, 5, 6, 7, 9,
       12, 13, 14, 15, 17~%-25%5, 6, 7, 9, 12, 13, 14, 15, 17}</td></tr>
<tr> <td>C</td>
  <td>{1:MULTICHOICE:~%-25%A~%-25%C~%100%D~%-25%F}</td>
  <td>{1:MULTICHOICE:~%100%E~%-25%F~%-25%D~%-25%B}</td>
  <td>{1:MULTICHOICE:~%-25%D~%-25%A~%-25%E~%100%F}</td>
  <td>{1:MULTICHOICE:~%100%4, 5, 6, 7, 9, 12, 13, 14, 15, 17~%-25%0, 1,
       2, 3, 8, 9, 11, 16~%-25%4, 5, 6, 7, 8, 13, 14, 15, 17}</td></tr>
<tr> <td>D</td>
  <td>{1:MULTICHOICE:~%-25%C~%-25%F~%100%D~%-25%A}</td>
  <td>{1:MULTICHOICE:~%-25%A~%100%D~%-25%B~%-25%F}</td>
  <td>{1:MULTICHOICE:~%100%D~%-25%A~%-25%E~%-25%F}</td>
  <td>{1:MULTICHOICE:~%-25%0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
       13, 14, 15, 16, 17~%100%Empty set}</td></tr>
<tr> <td>E</td>
  <td>{1:MULTICHOICE:~%-25%C~%-25%F~%100%D~%-25%A}</td>
  <td>{1:MULTICHOICE:~%-25%C~%-25%B~%-25%F~%100%E}</td>
  <td>{1:MULTICHOICE:~%-25%D~%-25%E~%-25%A~%100%F}</td>
  <td>{1:MULTICHOICE:~%100%6, 7, 9, 10, 11, 12, 13, 14, 15, 17~%-25%2,
       4, 5, 6, 8, 9, 14, 15, 17~%-25%0, 1, 2, 3, 4, 5, 8, 16}</td></tr>
<tr> <td>F</td>
  <td>{1:MULTICHOICE:~%100%D~%-25%A~%-25%C~%-25%E}</td>
  <td>{1:MULTICHOICE:~%-25%D~%-25%C~%100%E~%-25%A}</td>
  <td>{1:MULTICHOICE:~%100%F~%-25%D~%-25%E~%-25%B}</td>
  <td>{1:MULTICHOICE:~%-25%0, 1, 2, 3, 4, 5, 6~%100%6, 7, 8, 9, 11, 12,
       13, 14, 15, 16, 17~%-25%2, 3, 4, 5, 6, 7, 8, 9, 15, 17}</td></tr>
</tbody></table> <p>&nbsp;</p> <p>The final states are:
   {1:MULTICHOICE:~%100%B, C, E, F~ %-25%B, C, E~%-25%A, D}</p>
```

Figure 16. Example of content of a question with a table of cloze selections.

These metrics were chosen because they were directly related to the time and effort saved by using PLQuiz's XML export feature. No specific instructions were given about the editing process they had to follow.

After completing the tasks, the participants compiled their measurements and discussed the details of the process. This post-task reflection revealed additional aspects that were considered interesting to include in the analysis

- Number of errors: Errors made during question editing, such as syntax errors (e.g., incorrect cloze syntax), formatting problems, or missing elements. The time taken to correct any errors detected was included in the time measured.
- Editing strategies: Differences in the methods used by participants, such as using the Moodle visual editor, direct HTML editing, or hybrid approaches. These strategies influenced both efficiency and the types of error found.

Finally, the same participants were asked to use PLQuiz to generate each question type, export it to Moodle XML, and upload it into the Moodle question bank, this time using Moodle's import options and the generated XML file (which already contained the embedded images and tables). The reported times and number of clicks include the whole process, including actions taken in both PLQuiz and Moodle.

2.4.2. Obtaining Student Feedback

To evaluate the effectiveness of the quizzes generated by our tool, we conducted a survey among students of a third-year computer science course, "Language Processors", about compiler design, in the academic years 2021–2022 and 2022–2023. The survey was designed to be completely anonymous in order to encourage honest feedback and to ensure compliance with ethical considerations. No personally identifiable information was collected at any stage of the survey.

The survey was administered using a Google form, with the link shared via the subject's Moodle forum. Participation was voluntary, and students were required to confirm their understanding of the purpose of the survey with a mandatory first question before proceeding.

As the survey was anonymous from start to finish, and no personal data or potentially sensitive information were collected, it did not require approval from the university's ethics committee. In addition, there were no risks—physical, psychological, or social—posed to the participants. Requiring signed informed consent could have compromised participants' anonymity and discouraged voluntary participation.

We received responses from 53 students: 20 from the 2021–2022 course and 33 from the 2022–2023 course. The details of the questions asked and the analysis of the results are presented in the following section.

3. Results

To provide some perspective on the convenience of the tool, this section provides personal experience on its use and shows the students' opinions on the self-assessment questionnaires that are now possible to generate.

For four courses, the continuous assessment questionnaires for the subject "Language Processors" have incorporated exercises generated with PLQuiz. From our point of view, the main advantage of PLQuiz is the ease of being able to generate numerous exercises automatically. These exercises can be grouped into Moodle question bank categories that can be used to add random questions to quizzes. Since PLQuiz generates six different types of exercise, it is possible to have a category for each of them, allowing up to six different random questions to be added to the questionnaires. This randomization process minimizes the risk that some students will be tempted to share answers to exercises since, with a sufficient number of exercises, the chances of two students getting the same questions are very low.

3.1. Time Savings

Table 1 shows the results of the small experiment to measure manual editing time for questions. For both types of question, the time was considerable. For the question type "*Select 1 of 4 trees*", the times ranged from 5:56 to 7:24 min, with an average time of 6:46 (average number of clicks was 37). While for the more complex question "*Table of multi-choice selections*", the time ranged from 23:24 to 26:32 min, with the average time being 25:01 (average number of clicks was 34). These times are a direct reflection of the effort required to manually enter the question text, upload the images, format the multiple choices, and ensure that the question was correct and displayed without problems.

Table 1. Results of the editing experiment: time, clicks, errors, and some procedure details.

User	**Question Type**	**Time (min:s)**	**Clicks**	**Errors**	**Notes on Procedure**
1	Select 1 of 4 trees	7:00	39	1 'MULTICOICE' instead of 'MULTICHOICE'.	—
	Table of multi-choice selections	25:09	13	4 `\Simga` instead of `\Sigma`. Unclosed 'MULTICHOICE'. Duplicated table row. Wrong score.	Table was edited manually using directly HTML syntax.
2	Select 1 of 4 trees	7:24	38	1 Use of $ instead of %.	—
	Table of multi-choice selections	23:24	35	2 Wrong score. Omitted % in score.	Table inserted using Moodle's built-in table editor.
3	Select 1 of 4 trees	5:56	24	0	Images were dragged and dropped directly into the question text.
	Table of multi-choice selections	26:32	83	0	Table inserted using Moodle's built-in table editor. Rows were copied using mouse selection and copy and paste.
	Average	6:46	37		
		25:01	34		

In contrast, when questions were generated and imported using Moodle-compatible XML files produced by PLQuiz, the time required was significantly reduced (see Table 2). All participants completed the process in under 2 min per question, with the fastest participant achieving a time of just 59 s. The process was consistent across question types, regardless of whether the question included multiple images or a complex table of multi-choice cloze selections, as only a single XML file needed to be imported. This consistency allowed for a common average calculation across all questions, resulting in an average time of 1:15 and approximately 22 clicks per question.

When the preparation of questions takes a long time, one tends to adopt a protective attitude with the exercises that have been prepared, trying to reserve them only for the evaluation and with a certain reluctance to allow them to circulate and be shared among students. With PLQuiz, this secrecy is not necessary, since it is very easy to generate new exercises. For example, some courses ago, in addition to the question bank for continuous assessment, additional questions were generated for self-assessment questionnaires that students could use to prepare for the exam. In general, these self-assessment questionnaires were very well received by the students. The following section shows the results of the survey conducted with students on these questionnaires.

Table 2. The times and number of mouse operations required to import the two types of question into Moodle.

User	Question Type	Time (min:s)	Clicks
1	Select 1 of 4 trees	1:14	21
	Table of multi-choice selections	1:24	23
2	Select 1 of 4 trees	1:39	28
	Table of multi-choice selections	0:59	30
3	Select 1 of 4 trees	1:15	13
	Table of multi-choice selections	1:00	15
	Average	1:15	22

3.2. Students Feedback

In this section, we present the results of the survey conducted among students to evaluate the questionnaires generated by our tool. The survey aimed to gather information on the effectiveness and user experience of the generated questionnaires and to evaluate the impact that students perceive the availability of the self-assessment questionnaires generated by the tool may have. We initiated the survey by collecting general information about the participating students, including their opinion about the subject, if it was their first year in the subject, their route of access to the university, and their programming experience. After that, students were asked to rate the following aspects using a Likert scale:

Q1 The possibility of unlimited quizzes helps me better understand the algorithms.
Q2 Doing the questionnaires, I have realized that there were aspects that I had not understood.
Q3 I hope to get better grades thanks to the possibility of having practiced before.
Q4 In evaluation questionnaires, the time fixed for their completion is generally sufficient.
Q5 Self-assessment questionnaires should be available from the beginning of the lesson.
Q6 The challenge of obtaining a good grade in the questionnaires motivates me to study the subject.

Figure 17 shows the overall results. Figures 18–21 show the results broken down by different characteristics of the students surveyed. Of these characteristics, the one related to how the students gained access to the university (Figure 18) needs a little explanation. In the Spanish university system, there are several access routes for students to gain admission to a university degree program. These routes accommodate various educational backgrounds and qualifications, ensuring that students have multiple pathways to higher education. Here is an overview of the main access routes:

1. **Spanish University Entrance Examination**. Students typically spend two years completing their *Bachillerato*, the Spanish equivalent of high school. After earning their Bachillerato diploma, they must pass the *EBAU* (university entrance exam). Admission to university programs is based on a combination of their Bachillerato grades and EBAU scores. This route ensures that students have a solid academic foundation before entering higher education.
2. **Vocational training** or **technical schools**. Students who have completed a higher-level vocational training program (*Ciclo Formativo de Grado Superior* in Spanish) can access university degree programs. They must meet specific admission requirements and may also take optional subject-specific tests to improve their entrance score.
3. **Mature students**. In the Spanish university system, there is a special admission process for mature students who did not follow the traditional educational pathway. This

route allows adults to access higher education based on their age and life experience rather than on conventional academic qualifications. Individuals over 25 years of age who do not have traditional qualifications can take a special entrance exam (*Prueba de Acceso para Mayores de 25 Años*). Individuals over 40 can be admitted based on their professional experience and a personalized interview process.

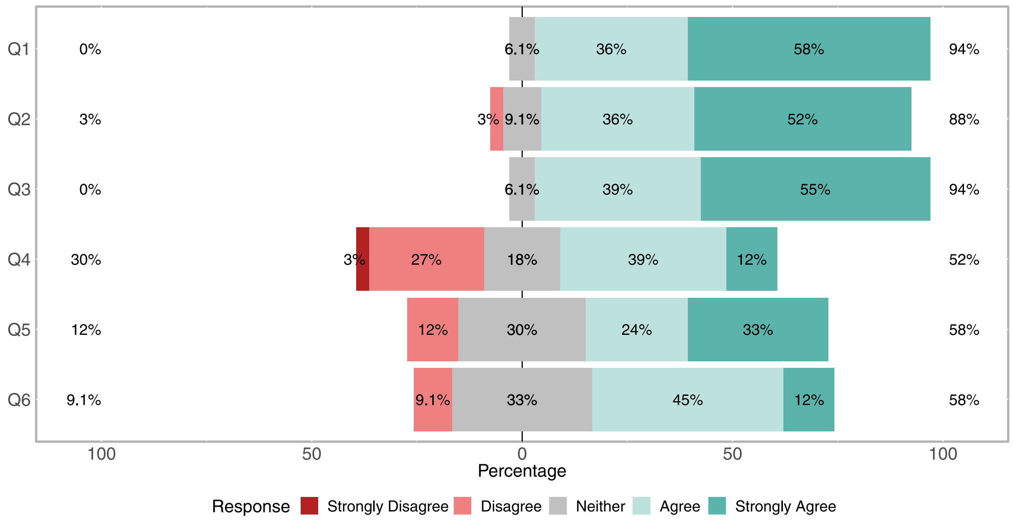


Figure 17. Overall results of the questionnaire.

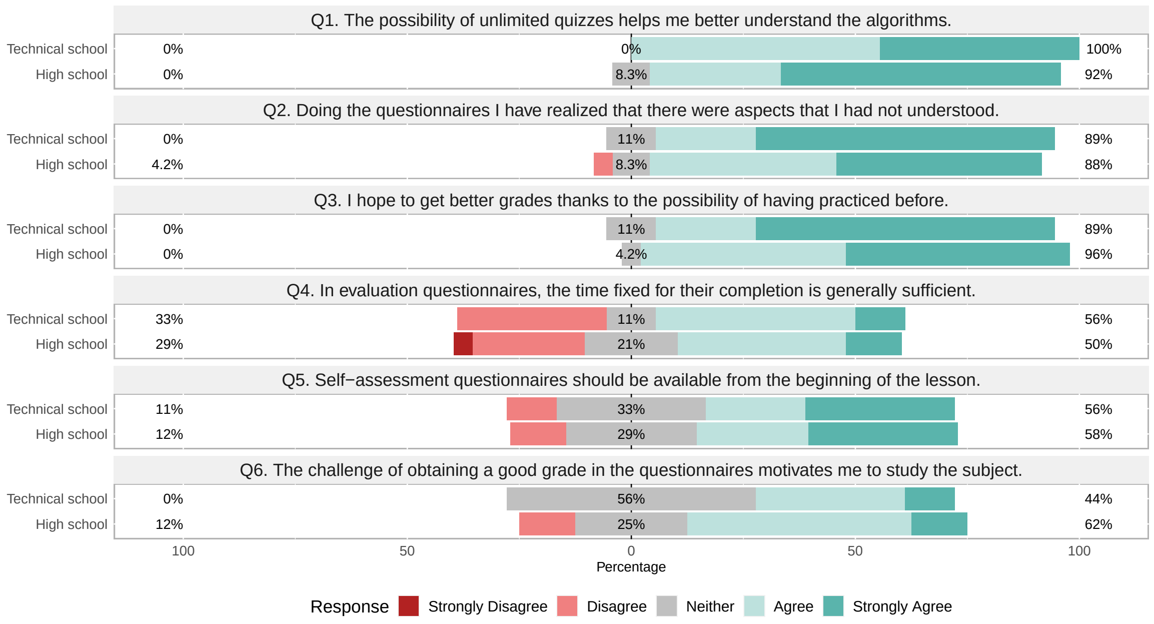


Figure 18. Results grouped by access to the university: technical school (24.53%), high school (75.47%).

All the students that answered the survey had entered university via the entrance exam or through vocational training.

Figure 20 also requires a brief explanation; it compares the responses given by students who are taking the subject for the first time ('1st-year') with students who have taken it in previous courses, have not passed the subject, and have needed to take it again ('Repeater').

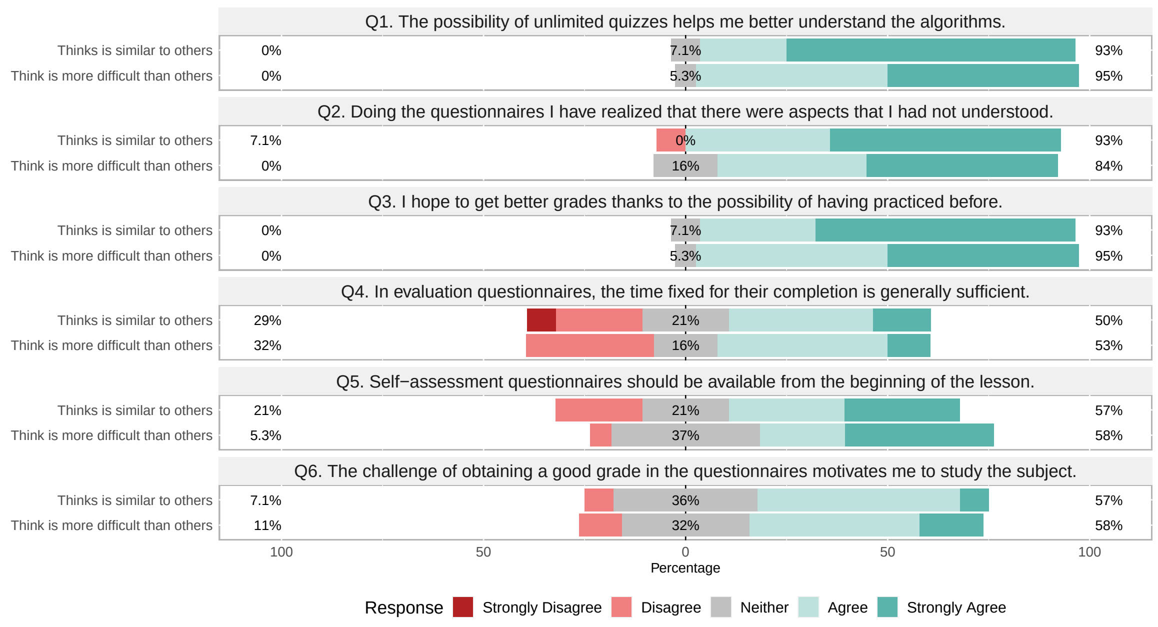


Figure 19. Results grouped by perceived difficulty: similar to others (33.96%), more difficult than others (66.04%).

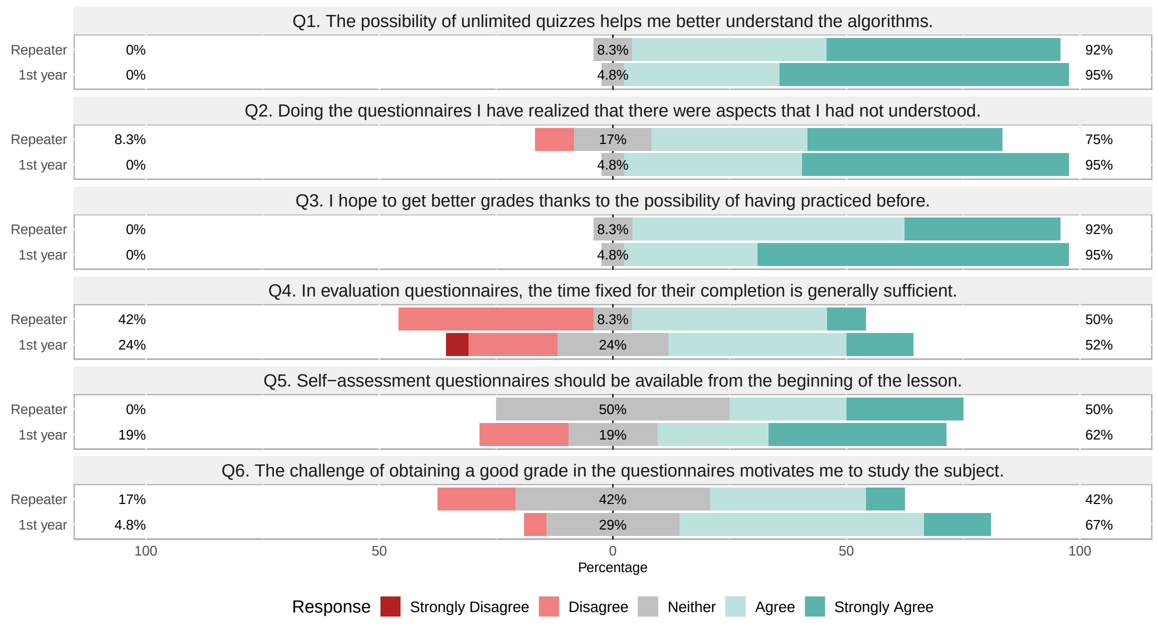


Figure 20. Results grouped by 1st year enrolled: repeaters (32.08%), 1st year (67.92%).

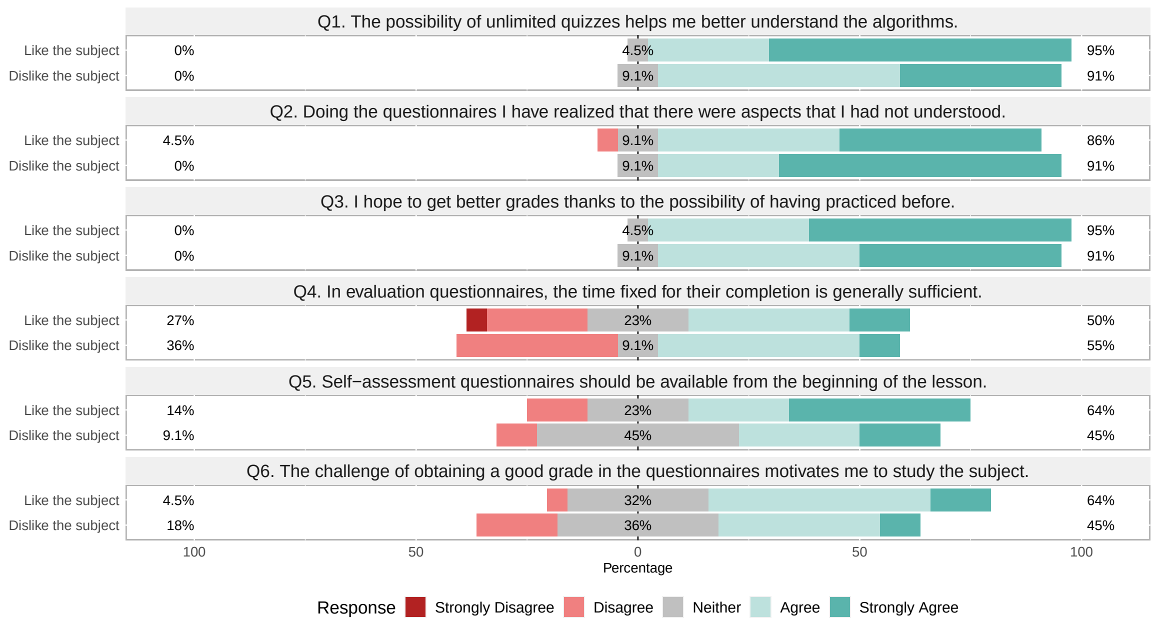


Figure 21. Results grouped by liking: like the subject (69.81%), dislike the subject (30.19%).

4. Discussion

4.1. Time Savings

A clear advantage of using PLQuiz is the automatic generation and solution of complete exercises, which includes the following:

- Generating automaton and syntax trees images, something that would otherwise require specialized tools such as graphics design software or LaTeX-based drawing libraries. These tools often demand prior knowledge and additional time for configuration, design, and validation. PLQuiz eliminates this workload entirely.
- Providing correct answers and distractors for inclusion in questionnaires. While having the solution to the exercise is important, it is equally necessary to create plausible incorrect alternatives for the questionnaires, a task that can be tedious and time-consuming. PLQuiz automatically generates these alternatives, simplifying the process.

Even with the images, solution, and distractors prepared in advance, creating a questionnaire in Moodle remains a difficult, time-consuming, and error-prone task. This was confirmed in the small experiment carried out, which demonstrated significant time reduction: 81.5% for the first type of question and 95% for the second. Interestingly, the best time for the first question was achieved by the user with the least prior experience with Moodle's question bank. This result can be attributed to his approach, which involved using drag-and-drop operations to upload the images directly into the question, avoiding the more tedious process of navigating through multiple HTML forms.

Obviously, the automatic generation of the questions makes the process easier, which is reflected in the reduction in the number of mouse interactions, although the reduction is less pronounced compared to the time reductions, only 38%. It should be noted that the impact of these reductions becomes more evident with larger questionnaires, where the time remains nearly the same regardless of the number of questions. Again, the less experienced user had a very different performance from the other two, with a higher number of mouse interactions in question 2, due to the fact that he also used the mouse during editing to select and copy elements from the previous rows of the table.

When using PLQuiz, it can be seen that the second type of question systematically required two more clicks than the first. This is because the drop-down menu for adding problems is set up by default to generate exercises for ASU, whereas it requires two additional interactions for the MYT exercises.

Finally, it is interesting to see what happened with the errors. Surprisingly, the less experienced user did not make any errors; again, the reason seems to be the approach used to edit the questions. It is also interesting to note that the highest number of errors occurred in the question with the cloze selector tables, no doubt because the user chose to edit directly in HTML.

4.2. Student Feedback

The responses to the survey on the questionnaires generated by PLQuiz offer valuable insights into its effectiveness and the students' perceptions. Feedback is generally positive, indicating that the use of the tool has been well received and serves its intended purpose. Below, we analyze the responses to each question and discuss the implications.

Q1: Unlimited Practice Opportunities. A significant majority of the students (94%) either agreed or totally agreed that the ability to complete an unlimited number of questionnaires helps them better understand the algorithms. This overwhelming positive response suggests that repeated practice is crucial for mastering the complex topics of lexical analysis and deterministic finite automata construction. The absence of negative responses indicates unanimous support for the tool's core functionality.

Q2: Identifying knowledge gaps. Similarly, 88% of students agreed or totally agreed that the questionnaires helped them identify areas they had not understood. This result underscores the diagnostic value of the tool, allowing students to recognize and address their misunderstandings early. Very few students disagreed, suggesting that almost all participants found the tool beneficial in this regard.

Q3: Anticipation of better grades. The expectation of improved exam performance due to practice with the questionnaires was also strongly supported, with 94% of students expressing agreement or strong agreement. This positive outlook implies that students feel more prepared and confident in their exams, which is likely to lead to better results.

Q4: Adequacy of time allocation. Feedback on the adequacy of the time provided to complete the questionnaires was more mixed. While 52% of students agreed or totally agreed, 30% of students disagreed or totally disagreed, and 18% were neutral. This divergence suggests that the time allocation may need adjustment. It might be beneficial to explore individual differences in time management or to consider offering different time options to cater to various student needs.

Q5: Availability from the beginning. Opinions were divided on whether or not self-assessment quizzes should be available from the beginning of the lesson. While 58% of students supported early availability, 12% disagreed, and 30% remained neutral. This mixed response indicates that, while many students see the value in early access for continuous learning, some may prefer a more structured approach in which quizzes are introduced after foundational knowledge has been established.

Q6: Motivational impact. Lastly, motivation to study driven by the challenge of improving quiz scores received moderate support. While 58% of students agreed or totally agreed, 42.1% of students were neutral or disagreed. This suggests that while the competitive aspect of the quizzes motivates some students, others may not find it as compelling. It could be useful to integrate additional motivational strategies to engage a broader range of students.

4.2.1. Results by Different Groupings

When analyzing the answers to the survey taking into account the different factors considered to divide the students into groups, it does not appear that the answers of each group differ significantly from each other. The path to university, the perceived difficulty of the subject in relation to other subjects in the degree, the fact that it is the student's first year taking the subject, or whether or not the student particularly likes the subject do not seem to have any influence on the answers provided to the questionnaire. To confirm this, both a t-test and a Mann–Whitney U test were performed in all groups for each question (De Winter & Dodou, 2010). The results showed that the differences are not statistically significant; in all cases the p-value was greater than 0.05, indicating that there are no significant differences in the survey responses for any of these partitions.

4.2.2. Implications for Teaching Practice

The survey results highlight several key points for enhancing the teaching and learning process:

1. **Reinforcement through practice**: The ability to generate unlimited questionnaires is highly beneficial for student learning, suggesting that frequent and varied practice opportunities should be integrated into the curriculum.
2. **Diagnostic value**: The tool effectively helps students identify gaps in their knowledge, emphasizing the importance of incorporating diagnostic assessments to guide personalized learning.
3. **Preparation and confidence**: The anticipation of better grades through practice indicates that such tools can boost student confidence and prepareness for exams, likely leading to better academic performance.
4. **Time management**: Mixed feedback on time adequacy suggests a need for flexible timing options or additional support in time management strategies.
5. **Early access**: Providing quizzes from the beginning of the lesson may benefit many students, although a balance must be struck to accommodate different learning preferences.
6. **Motivational strategies**: Although some students are motivated by the challenge of improving their grades, others may need different incentives. Incorporating a variety of motivational strategies could improve engagement.

In conclusion, the tool's ability to easily generate an arbitrary number of questionnaires has proven to be a valuable asset in teaching complex topics in lexical analysis and finite automata.

5. Conclusions

This paper presented PLQuiz, an open-source multilingual tool to automatically generate Moodle quizzes that assess student understanding of lexical analysis algorithms, particularly the conversion of regular expressions into finite automata. PLQuiz offers significant advantages over manual question creation, allowing educators to easily produce diverse and numerous questions for both online and in-person assessments. Key contributions of PLQuiz include the following:

- Time-saving automation: It dramatically reduces the time and effort required to design and implement quizzes, addressing a major problem for educators.
- Multiple output formats: It supports Moodle XML for seamless online integration and LaTeX for high-quality printable exams, offering versatility for different assessment scenarios.
- Variety of question types: The diversity of questions in PLQuiz, which vary in complexity and cover different aspects of lexical analysis algorithms, improves students'

engagement with the subject by having them face various challenges, avoiding simple memorization, and demanding a deeper understanding of algorithms.

- Customization and flexibility: Users can manually enter regular expressions or use the built-in generator with parameters to control the complexity of the quiz.
- Positive student feedback: The surveys indicated that the students found the quizzes beneficial for comprehension, identifying knowledge gaps, and improving exam preparation.
- Improvement of competencies: PLQuiz is designed to assess students' knowledge, not their competencies, but it indirectly contributes to the improvement of some of them, such as organization and planning, through timed questionnaires. It also contributes to independent learning through self-assessment, which allows students to practice and deepen their knowledge on their own.

In order to further improve the tool and increase its impact on the teaching of lexical analysis algorithms, several lines of future work have been identified that could be addressed in future versions. From a functional point of view, new types of exercise could be added. For example, it would be interesting to include questions on other aspects of lexical analysis algorithms, such as automata minimization (J. Hopcroft, 1971), the algorithm to identify the equivalence between automata (J. E. Hopcroft & Karp, 1971), or obtaining a regular expression from a finite automaton (Morazán, 2023).

From a technical point of view, drawing trees in the generated LaTeX document using the `tikz-qtree` library has been very convenient for the rapid development of this part of the application, but it could be interesting to have more control in drawing trees by directly managing the coordinates of the nodes (as is now the case for automata), so in the future we would like to incorporate the Reingold and Tilford algorithm (Reingold & Tilford, 1981) to be able to generate more compact trees with greater control over their design. We would also like to improve the aesthetics of the automata and trees generated for Moodle quizzes and refactor the code so that graph generation can be common for both types of output.

The interface of the application could also be improved by providing more direct access to the quiz-generating actions. Instead of having to select the type of exercise to add, there could be a toolbar with individual buttons for each type of exercise.

Finally, although the application has been internationalized from an end-user point of view, the comments in the code, the log messages, and the variable names are still in Spanish. In the future, they are expected to be translated into English to make the code more accessible and to facilitate the participation of other developers in the open-source community.

Author Contributions: Conceptualization, C.I.G.-O.; software, R.I.-A., J.A.B.-A. and C.I.G.-O.; validation, R.I.-A., J.A.B.-A. and C.I.G.-O.; data curation, A.O.-G. and J.L.G.-L.; writing—original draft preparation, C.I.G.-O.; writing—review and editing, all authors; visualizations, A.O.-G. and J.L.G.-L.; supervision, C.I.G.-O. All authors have read and agreed to the published version of the manuscript.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author, C.I.G.-O, upon reasonable request.

Acknowledgments: We would like to thank Pedro Latorre-Carmona for his insightful suggestions on earlier versions of the article, and also to Raúl Marticorena-Sánchez for his invaluable technical help in solving some encoding problems with Java and Eclipse.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GIFT	General Import Format Template
XML	eXtensible Markup Language
DFA	Deterministic Finite Automata
NFA	Nondeterministic Finite Automata
MYT	McNaughton–Yamada–Thompson
ASU	Aho–Sethi–Ullman
RE	Regular Expression
SVG	Scalable Vector Graphics

References

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers, principles, techniques. *Addison Wesley*, *7*(8), 9.

Alder, G. (2013). *Jgraph.* GitHub. Available online: https://github.com/jgraph/jgraphx (accessed on 7 January 2025).

Arnaiz-González, Á., Díez-Pastor, J.-F., Ramos-Pérez, I., & García-Osorio, C. (2018). Seshat—A web-based educational resource for teaching the most common algorithms of lexical analysis. *Computer Applications in Engineering Education*, *26*(6), 2255–2265. [CrossRef]

Blythe, S. A., James, M. C., & Rodger, S. H. (1994). LLparse and LRparse: Visual and interactive tools for parsing. *ACM SIGCSE Bulletin*, *26*(1), 208–212. [CrossRef]

Castro-Schez, J., Glez-Morcillo, C., Albusac, J., & Vallejo, D. (2021). An intelligent tutoring system for supporting active learning: A case study on predictive parsing learning. *Information Sciences*, *544*, 446–468. Available online: https://www.sciencedirect.com/science/article/pii/S0020025520308331 (accessed on 7 January 2025). [CrossRef] [PubMed]

Chakraborty, P., Saxena, P. C., & Katti, C. P. (2011). Fifty years of automata simulation: A review. *Acm Inroads*, *2*(4), 59–70. [CrossRef]

Chesnevar, C. I., Cobo, M. L., & Yurcik, W. (2003). Using theoretical computer simulators for formal languages and automata theory. *ACM SIGCSE Bulletin*, *35*(2), 33–37. [CrossRef]

Chesnevar, C. I., González, M. P., & Maguitman, A. G. (2004a). Didactic strategies for promoting significant learning in formal languages and automata theory. *ACM SIGCSE Bulletin*, *36*(3), 7–11. [CrossRef]

Chesnevar, C. I., Maguitman, A. G., González, M. P., & Cobo, M. L. (2004b). Teaching fundamentals of computing theory: A constructivist approach. *Journal of Computer Science and Technology*, *4*(02), 91–97.

Chiang, D. (2012). *tikz-qtree: Better trees with tikz.* CTAN. Available online: https://ctan.org/pkg/tikz-qtree (accessed on 7 January 2025).

Chuda, D., Trizna, J., & Kratky, P. (2015, June 25–26). *Android automata simulator*. International Conference on e-Learning (pp. 80–84), Nassau, The Bahamas.

Community, J. (2023). *Javacc: A parser generator for building parsers from grammars.* GitHub. Available online: https://github.com/javacc/javacc (accessed on 7 January 2025).

Copeland, T. (2007). *Generating parsers with javacc: An easy-to-use guide for developers*. Centennial Books. ISBN 13: 978-0976221432.

Demaille, A., Levillain, R., & Perrot, B. (2008). A set of tools to teach compiler construction. *ACM SIGCSE Bulletin*, *40*(3), 68–72. [CrossRef]

D'Antoni, L., Helfrich, M., Kretinsky, J., Ramneantu, E., & Weininger, M. (2020, July 21–24). *Automata tutor v3*. Computer Aided Verification: 32nd International Conference, CAV 2020 (pp. 3–14, Proceedings, Part II 32), Los Angeles, CA, USA.

De Winter, J. C., & Dodou, D. (2010). Five-point Likert items: T test versus Mann-Whitney-Wilcoxon. *Practical Assessment, Research & Evaluation*, *15*(11), 1–12.

Esparza, J., & Blondin, M. (2023). *Automata theory: An algorithmic approach.* MIT Press.

Gallardo Casero, J., Castro Sánchez, J. J., & Sabariego, R. M. (2016). Experiencias de uso y evaluación de una herramienta de apoyo a la enseñanza de Teoría de Autómatas y Lenguajes Formales. In *Actas de las XXII JENUI* (pp. 327–334). Universidad de Almería.

García-Osorio, C., Arnaiz-Moreno, A., & Arnaiz-González, A. (2007). THOTH: A new tool for automata theory learning. In *International technology, education and development conference.* IATED.

García-Osorio, C., Gómez-Palacios, C., & García-Pedrajas, N. (2008). A Tool for Teaching LL and LR Parsing Algorithms. *SIGCSE Bull*, *40*(3), 317. [CrossRef]

García-Osorio, C., Mediavilla-Sáiz, I., Jimeno-Visitación, J., & García-Pedrajas, N. (2008, June 30–July 2). *Teaching push-down automata and turing machines*. 13th Annual Conference on Innovation and Technology in Computer science education (pp. 316–316), Madrid, Spain.

Golemanov, T., & Golemanova, E. (2020, June 19–20). *A set of tools to teach language processors construction*. 21st International Conference on Computer Systems and Technologies (pp. 244–250), Ruse, Bulgaria.

Hopcroft, J. (1971). An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of machines and computations* (pp. 189–196). Elsevier.

Hopcroft, J. E., & Karp, R. M. (1971). *A linear algorithm for testing equivalence of finite automata* (Vol. 114). Defense Technical Information Center.

Jordaan, S., Timm, N., & Marshall, L. (2024a). AutomaTutor: An Educational Mobile App for Teaching Automata Theory. In H. Barbosa, & Y. Zohar (Eds.), *Formal methods: Foundations and applications* (pp. 131–140). Springer Nature.

Jordaan, S., Timm, N., & Marshall, L. (2024b). Migrating teaching of automata theory to a digital platform. *South African Computer Journal*, *36*(2), 31–67.

Kasim, N. N. M., & Khalid, F. (2016). Choosing the right learning management system (LMS) for the higher education institution context: A systematic review. *International Journal of Emerging Technologies in Learning*, *11*(6). [CrossRef]

Knobelsdorf, M., Kreitz, C., & Böhne, S. (2014, March 5–8). *Teaching theoretical computer science using a cognitive apprenticeship approach*. 45th ACM Technical Symposium on Computer Science Education (pp. 67–72), Atlanta, GA, USA.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Linz, P., & Rodger, S. H. (2022). *An introduction to formal languages and automata* (7th ed.). Jones & Bartlett Learning.

López-Tocón, I. (2021). Moodle quizzes as a continuous assessment in higher education: An exploratory approach in physical chemistry. *Education Sciences*, *11*(9), 500. [CrossRef]

McNaughton, R., & Yamada, H. (1960). Regular expressions and state graphs for automata. *IRE Trans Electron Comput*, *9*(1), 39–57. [CrossRef]

Mohammed, M., Shaffer, C. A., & Rodger, S. H. (2021, March 13–20). *Teaching formal languages with visualizations and auto-graded exercises*. 52nd ACM Technical Symposium on Computer Science Education (pp. 569–575), Virtual Event, USA.

MoodleDocs. (2022a). *Aiken format*. Available online: https://docs.moodle.org/404/en/Aiken_format (accessed on 20 November 2024).

MoodleDocs. (2022b). *GIFT format*. Available online: https://docs.moodle.org/404/en/GIFT_format (accessed on 20 November 2024).

MoodleDocs. (2024). *Moodle XML format*. Available online: https://docs.moodle.org/404/en/Moodle_XML_format (accessed on 20 November 2024).

Morazán, M. T. (2023). *Programming-based formal languages and automata theory: Design, implement, validate, and prove*. Springer Nature.

Muñoz, R. C., Blanco, B. B., Campo-Ávila, J. d., & Rodriguez, J. L. T. (2024). Teaching Compilers: Automatic Question Generation and Intelligent Assessment of Grammars' Parsing. *IEEE Transactions on Learning Technologies*, *17*, 1694–1704. [CrossRef]

Nash, S. S., & Rice, W. (2018). *Moodle 3 e-learning course development: Create highly engaging and interactive e-learning courses with moodle 3*. Packt Publishing Ltd.

Quesada, E. V., & Herrera, J. F. Á. (2024). Generación de código para la elaboración de preguntas tipo cloze en Moodle usando Wolfram Mathematica: Code generation for creating cloze questions in Moodle using Wolfram Mathematica. *Revista Digital: Matemática, Educación e Internet*, *24*(1). [CrossRef]

Rabin, M. O., & Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, *3*(2), 114–125. [CrossRef]

Reingold, E. M., & Tilford, J. S. (1981). Tidier drawings of trees. *IEEE Transactions on software Engineering*, *SE-7*(2), 223–228. [CrossRef]

Rodger, S. H., & Finley, T. W. (2006). *JFLAP: An interactive formal languages and automata package*. Jones & Bartlett Learning.

Sáenz, J., Gurtubay, I. G., Izaola, Z., & López, G. A. (2020). Pygiftgenerator: A python module designed to prepare Moodle-based quizzes. *European Journal of Physics*, *42*(1), 015702. [CrossRef]

Singh, T., Afreen, S., Chakraborty, P., Raj, R., Yadav, S., & Jain, D. (2019). Automata Simulator: A mobile app to teach theory of computation. *Computer Applications in Engineering Education*, *27*(5), 1064–1072. [CrossRef]

Stamenkoviæ, S., & Jovanoviæ, N. (2021, February 16–20). *Improving participation and learning of compiler theory using educational simulators*. 25th International Conference on Information Technology (it) (pp. 1–4), Zabljak, Montenegro. [CrossRef]

Stamenković, S., & Jovanović, N. (2023). A Web-based Educational System for Teaching Compilers. *IEEE Transactions on Learning Technologies*, *17*, 143–156. [CrossRef]

Stamenković, S., Jovanović, N., & Chakraborty, P. (2020). Evaluation of simulation systems suitable for teaching compiler construction courses. *Computer Applications in Engineering Education*, *28*(3), 606–625. [CrossRef]

Steingartner, W. (2021). On some innovations in teaching the formal semantics using software tools. *Open Computer Science*, *11*(1), 2–11. [CrossRef]

Steingartner, W., & Sivỳ, I. (2023, September 4–7). *From high-level language to abstract machine code: An interactive compiler and emulation tool for teaching structural operational semantics*. European conference on advances in databases and information systems (pp. 544–551), Barcelona, Spain. [CrossRef]

Svien, J. (2019, March 1–27). *Improvements to the Moodle cloze and GIFT code generator*. Moodlemoot Japan 2019 Annual Conference (pp. 19–23), Shizuoka, Japan. Available online: https://www.academia.edu/40400476/Improvements_to_the_Moodle_Cloze_and_GIFT_Code_Generator (accessed on 7 January 2025).

Tantau, T. (2015). *The TikZ and PGF packages (manual for version 3.0. 1a; 2015.08.29)* (Tech. Rep.). Institut für Theoretische Informatik Universität zu Lübeck.

Veluvali, P., & Surisetti, J. (2022). Learning management system for greater learner engagement in higher education—A review. *Higher Education for the Future*, *9*(1), 107–121. [CrossRef]

Xin, N. S., Shibghatullah, A. S., & Abd Wahab, M. H. (2021). A systematic review for online learning management system. *Journal of Physics: Conference Series*, *1874*(1), 012030. [CrossRef]