





Teaching Compilers: Automatic Question Generation and Intelligent Assessment of Grammars' Parsing

Ricardo Conejo Muñoz , Beatriz Barros Blanco , José del Campo-Ávila , and José L. Triviño Rodríguez 

Abstract—Automatic question generation and the assessment of procedural knowledge is still a challenging research topic. This article focuses on the case of it, the techniques of parsing grammars for compiler construction. There are two well-known techniques for parsing: top-down parsing with LL(1) and bottom-up with LR(1). Learning these techniques and learning to design grammars that can be parsed with these techniques requires practice. This article describes an application that covers all the tasks needed to automatize the learning and assessment process: 1) automatically generate context-free languages and grammars of different complexity; 2) pose different types of questions to the student with an appropriate response interface; 3) automatically correct the student answer, including grammar design for a given language; and 4) provide feedback on errors. The application has been implemented as a plug-in of the SIETTE assessment system that, in addition, can provide adaptive behavior for question selection. It has been successfully used by more than a thousand students for formative and summative assessment.

Index Terms—Adaptive feedback, automatic assessment, compiler construction, educational technology, procedural knowledge, question generation, top-down parsing.

I. INTRODUCTION

COMPILER construction is a compulsory subject in almost all computer science degrees. Here, the students learn different algorithms, tools, and methods necessary to understand how a compiler for a programming language is constructed [1]. A core part of compiler construction is the design of the language grammar and the construction of an efficient parser without the need of backtracking. There are two well-known techniques for parsing: 1) the LL(1) technique allows us to construct efficient top-down parsers based on theoretical grounds, but it requires some conditions to be met for the grammar design and 2) the family of LR(1) techniques for bottom-up parsing. One of them is the SLR(1) technique, a simplification of LR(1) technique. Let us introduce some concepts and notation that are used in this article.

A context-free grammar (CFG) is defined as $G(N, T, P, S)$, where N is a set of *nonterminal* symbols, T is a set of *terminal* symbols, P is a set of *production rules* (of the form $A \rightarrow \gamma$,

where A is called the *antecedent* and $\gamma \in (N \cup T)^*$ is called the *consequent*), and S is the *axiom*. The languages that can be generated by a CFG are called context-free languages (CFLs). Two CFGs are *equivalent* if they generate the same CFL.

A *left derivation* \Rightarrow of a string $\alpha A \beta \in (N \cup T)^+$ is defined as a substitution of the left-most *nonterminal* symbol of A by any right-hand side of a production rule $A \rightarrow \gamma$, that is, if the rule $A \rightarrow \gamma \in P$, then $\alpha A \beta \Rightarrow \alpha \gamma \beta$. Multiple applications of the left derivation are denoted as \Rightarrow^k (applying k left derivations) or \Rightarrow^* if zero, one or more *left derivations* are applied.

A *sentential form* is defined as a string $\alpha \in (N \cup T)^*$ so that $S \Rightarrow^* \alpha$, which is a generalization of the concept of a *sentence* of the language, which is defined in the same way but with all symbols of $\alpha \in T^*$

LL(1) grammars are a subset of CFG that accomplish the LL(1) condition. There is a well-known algorithm to efficiently determine if a CFG is LL(1) and construct its parsing table. It is based on the construction of the functions FIRST and FOLLOW and the directive symbols of each production rule DS [1]. *LL(1) languages* are those CFLs that can be generated by *LL(1) grammars*. *LL(1) languages* are a proper subset of CFL.

SLR(1) grammars are also a proper subset of CFG although they are less restricted than LL(1) grammars. This technique requires the definition of the concept of *LR(0) item*, which is a triple (A, α, β) , where $A \rightarrow \alpha \beta \in P$. Let call J the set of all possible items for a given grammar. Sets of LR(0) items, $I \in \mathcal{P}(J)$, are called states. A special set of states, called the *LR(0) collection* \mathcal{C} , is obtained by applying two functions: the *CLOSURE* and *DELTA* functions [1]. Given the *LR(0) collection* \mathcal{C} , the SLR(1) technique generates a table that can be used to guide bottom-up parsing. If the table has no duplicate entries, then parsing can be done without backtracking, and it is said that the grammar accomplishes the SLR(1) condition and then the language is called an *SLR(1) language*. *SLR(1) languages* are a proper subset of CFL and a proper superset of *LL(1) languages*.

There are many tools that, given a CFG, can automatically construct the LL(1) and SLR(1) tables (see Section II). As far as we know, these systems are not designed for assessment, nor to generate the grammar given a target language, but only to check whether a given grammar accomplishes the LL(1) or SLR(1) conditions and generates the corresponding tables.

One of the main problems that students face is the design of the grammar given the target language. In the general case, designing a grammar that generates a given CFL cannot be done automatically. Neither can be decided if a given CFG has an *equivalent grammar* that accomplishes SLR(1) or LL(1)

Manuscript received 8 February 2024; revised 16 April 2024; accepted 20 May 2024. Date of publication 28 May 2024; date of current version 7 June 2024. This work was supported in part by the Universidad de Málaga / Consortium of University Libraries of Andalusia (CBUA). (Corresponding author: José L. Triviño Rodríguez.)

The authors are with the Department of Computer Science and Programming Languages, University of Málaga, 29071 Málaga, Spain (e-mail: conejo@uma.es; bbarros@uma.es; jcampo@uma.es; jltrivino@uma.es).

Digital Object Identifier 10.1109/TLT.2024.3405565

| | LISA | SESHAT | PROLETOOL | PAMOJA | COMVIS | SIETTE |
|---|-------------------------|------------------|--|------------------|------------------|-------------------------------------|
| USAGE: Simulation, Training, Assessment | Simulation; Training | Simulation | Simulation; Training; Assessment | Training | Simulation | Training; Assessment |
| TASK: Teacher-authored, Student-authored, Auto-generated | Teacher-authored | Teacher-authored | Teacher-authored; Student-authored | Teacher-authored | Teacher-authored | Teacher-authored; Auto-generated |
| AUTOMATIC ASSESSMENT: Yes, No | No | No | Yes | No | No | Yes |
| PLATFORM: Desktop, Web-based | Desktop | Web-based | Web-based | Desktop | Web-based | Web-based |

Fig. 1. Comparative table of SIETTE features with other tools that address similar educational tasks published along last decade: LISA [2], SESHAT [3], PROLETOOL [4], PAMOJA [5], and COMVIS [6].

conditions. However, in practice, for most syntactic structures of programming languages, the equivalent LL(1) or SLR(1) grammars can be obtained using some heuristic rules. It requires a little art and a lot of practice from the designer. The aim of this work is to promote this practice for students by means of automatically generation of cases where a solution is known to exist.

This article describes the implementation of a computer-based assessment application, constructed as a plug-in of the SIETTE assessment system (see Section III) that is able to automatically generate a random CFG, determine if it is suitable for LL(1) and/or SLR(1) parsing, construct the parsing tables step by step, and evaluate the student answers to the following types of questions.

For the LL(1) technique:

- Q1) Given a string $\alpha \in (N \cup T)^*$, find the set $FIRST(\alpha)$.
- Q2) Given a nonterminal $A \in N$, find the set $FOLLOW(A)$.
- Q3) Given a production rule $A \rightarrow \alpha \in P$, find the set $DS(A \rightarrow \alpha)$.
- Q4) Given a CFG, find the LL(1) parsing table.
- Q5) Given a sentence $\alpha \in T^*$, find left derivations so that $S \Rightarrow^* \alpha$.
- Q6) Given a CFL L , find an LL(1) grammar G so that $L(G) = L$.

For the SLR(1) technique:

- Q7) Given a set of LR(0) items $I \in \mathcal{P}(J)$, find the set $CLOSURE(I)$.
- Q8) Given a set of LR(0) items $I \in \mathcal{P}(J)$ and a symbol $X \in (N \cup T)$, find the set $DELTA(I, X)$.
- Q9) Given a CFG, find the LR(0) collection \mathcal{C} .
- Q10) Given a CFG, find the SLR(1) parsing tables.

The application has been used for seven academic years at the University of Málaga and has been used by more than a thousand students for formative and summative assessment (see Section V). Finally, some conclusions and future evolution lines are proposed (see Section VI). A preliminary version of this article covering just LL(1) techniques was presented in [7].

II. RELATED WORK

There are several systems for teaching and training the creation of compilers in computer science, many of them discussed

in the review by Stamenković et al. [8] considering only the simulation tools, but there are other systems that take into account other aspects, such as gamification [9], training [7], [10], and assessment of students' knowledge [4], [7], [11], [12].

Fig. 1 shows a set of systems (all written in Java with graphical output and, although they have a relatively long history, with an updated publication in the last decade) with an educational objective similar to that of the system presented here. In the table, we consider: 1) how the tool is used for teaching compiler knowledge: teaching and presenting algorithms with simulations (Simulation), training these algorithms (Training), and assessing the student's knowledge (Assessment); 2) how the tasks or examples used by the tool are defined: defined by a teacher (Teacher-authored) or a student (Student-authored) in some kind of form or automatically generated by the system (Auto-Generated); 3) whether the tool performs an automatic assessment of the student: Yes or No; and 4) whether the system operates as a Desktop program [stand-alone program or within an integrated development environment (IDE)] or is a Web-based platform.

The first difference is that these systems have been developed with the sole purpose of supporting the teaching of compilers, whereas SIETTE is a general-purpose evaluation framework for training and automatic assessment that includes a specific module covering the subject of compilers. This is why the other systems have a simulation module, whereas SIETTE focuses specifically on the completion of tasks and the assessment of the student's knowledge throughout the course.

To our knowledge, none of the systems consulted in the bibliography use generative tasks based on the characteristics of each type of grammar, as SIETTE does. This most remarkable differentiating element, together with its intelligent assessment module for training and evaluating students, represents a breakthrough in the development of systems for teaching compilers. It goes a step further than other systems because it automatically generates problems to be solved, recognizes students' answers, and provides customized feedback on their errors in context.

Most of the automated question generation systems are associated with texts that often require the use of natural language analysis techniques [13], [14], [15]. Our approach focuses on the evaluation of procedural knowledge [16], specifically related to complex algorithmic problem solving; what is asked and what is

evaluated are programs that a student solves as part of a practical task. In this sense, we are referring to *automated task generation*.

III. SYSTEM ARCHITECTURE

In order to assess the student knowledge and skills needed to design and implement LL(1) and SLR(1) parsers for a given language, we have implemented a plug-in extension of the SIETTE assessment environment. Using this plug-in and some of the standard features of SIETTE, we are able to automatically generate different types of questions.

A. SIETTE Assessment System

SIETTE [17] is a general-purpose automatic assessment environment that supports the generation of different questions based on JSP templates, different types of questions, and student answer interfaces; automated recognition of students' open answers based on regular expression patterns; and a flexible support of any other assessment requirement based on the construction of a plug-in extension. SIETTE implements the classical test theory [18], item response theory (IRT) [19], and computerized adaptive testing [20], and it provides built-in statistical and psychometric tools to analyze students, tests, and questions' results.

SIETTE includes an authoring tool for teachers to create questions and define assessment criteria, and a user interface for students to take the assessment (see Fig. 2).

Teachers can create different JSP templates using the classes included in the plug-in application programming interface (API). Templates' performance can be tested and previewed in the authoring tool and will be instantiated to questions when the students take the assessment (see Fig. 3).

Some of the questions listed in Section I can be answered by the student using a simple HTML text-area field. Other questions (those that require a table) require a form that is composed in JavaScript and transformed into a structured text that is sent to SIETTE.

The student answer is given to SIETTE in a plain or structured text format. SIETTE recognizes whether the answer is correct using a pattern matching process. Patterns are provided by the teacher, and the matcher algorithm is implemented as a plug-in. There are some default matcher plug-ins that are already implemented in SIETTE. One of them is the *SIETTE regular expression* whose behavior is described in the SIETTE manual [21]. It extends regular expression to recognize small sets of strings and is used to recognize the answer of simpler questions. However, to recognize grammars (i.e., question types Q6 and Q9), a new plug-in is needed to compare the grammar proposed by the teacher with the one provided by the student answer.

B. SIETTE CFG Plug-In

The main features of the CFG plug-in are as follows: 1) automatic generation of context-free SLR(1) and LL(1) grammars; 2) automatic construction of SLR(1) and LL(1) parsers; and 3) checking grammar equivalence.

1) *Automatic Generation of CFGs*: One of the first challenges of this project is to define a way to generate small CFLs that can be used to pose questions to students. The alphabet of these languages (*terminal symbols*) is restricted to lowercase letter in order to be easy to write it in text format.

Small CFGs can be generated just by setting the antecedent and a random length string that combines terminal and nonterminal symbols. This strategy requires validating the generated grammar and repeating the process until a correct grammar is obtained. On the other hand, a well-defined CFG can be generated based on the composition of "building block" grammars. The building blocks are tiny CFGs with just two or three production rules. Some of them are listed as follows:

$$\begin{aligned} A &\rightarrow Aa \\ A &\rightarrow a \end{aligned} \quad (1)$$

$$\begin{aligned} A &\rightarrow aAb \\ A &\rightarrow ab. \end{aligned} \quad (2)$$

The plug-in defines some building block grammars, but they can be easily extended as needed. Using these building blocks, we apply a composition rule just by replacing a *terminal* symbol with a *nonterminal* symbol of another building block. For instance, combining block (1) and block (2) in this order will generate the following grammar:

$$\begin{aligned} A &\rightarrow AB \\ A &\rightarrow B \\ B &\rightarrow aBb \\ B &\rightarrow ab. \end{aligned}$$

On the other hand, combining block (2) and block (1) grammars can give one of these four grammars

$$\begin{aligned} A &\rightarrow BAbA \rightarrow BAb \\ A &\rightarrow Bb \quad A \rightarrow Bb \\ B &\rightarrow Ba \quad B \rightarrow Bb \\ B &\rightarrow a \quad B \rightarrow b \end{aligned}$$

$$\begin{aligned} A &\rightarrow aABA \rightarrow aAB \\ A &\rightarrow aB \quad A \rightarrow aB \\ B &\rightarrow Ba \quad B \rightarrow Bb \\ B &\rightarrow a \quad B \rightarrow b. \end{aligned}$$

Note that there are four possible ways to combine block (2) and block (1) grammars, because we have two alternative options: 1) in block (2), there are two terminal symbols, so we have two options to replace a *terminal* with a *nonterminal* of the second grammar and 2) we have to choose if the *terminal* symbols of the resulting grammar are the same or not. Nevertheless, without loss of generality, we can always assume that *terminals* are different, and at the end of the generation process, two or more symbols can be merged as a single *terminal*. That is, in the last example, options 2 and 4 can be obtained from options 1 and 3 just by considering that *terminal a* and *terminal b* are the same. This process is delayed until we finish the combination process.

Thus, a CFG can be randomly generated by selecting the building blocks to combine the number of combinations to apply

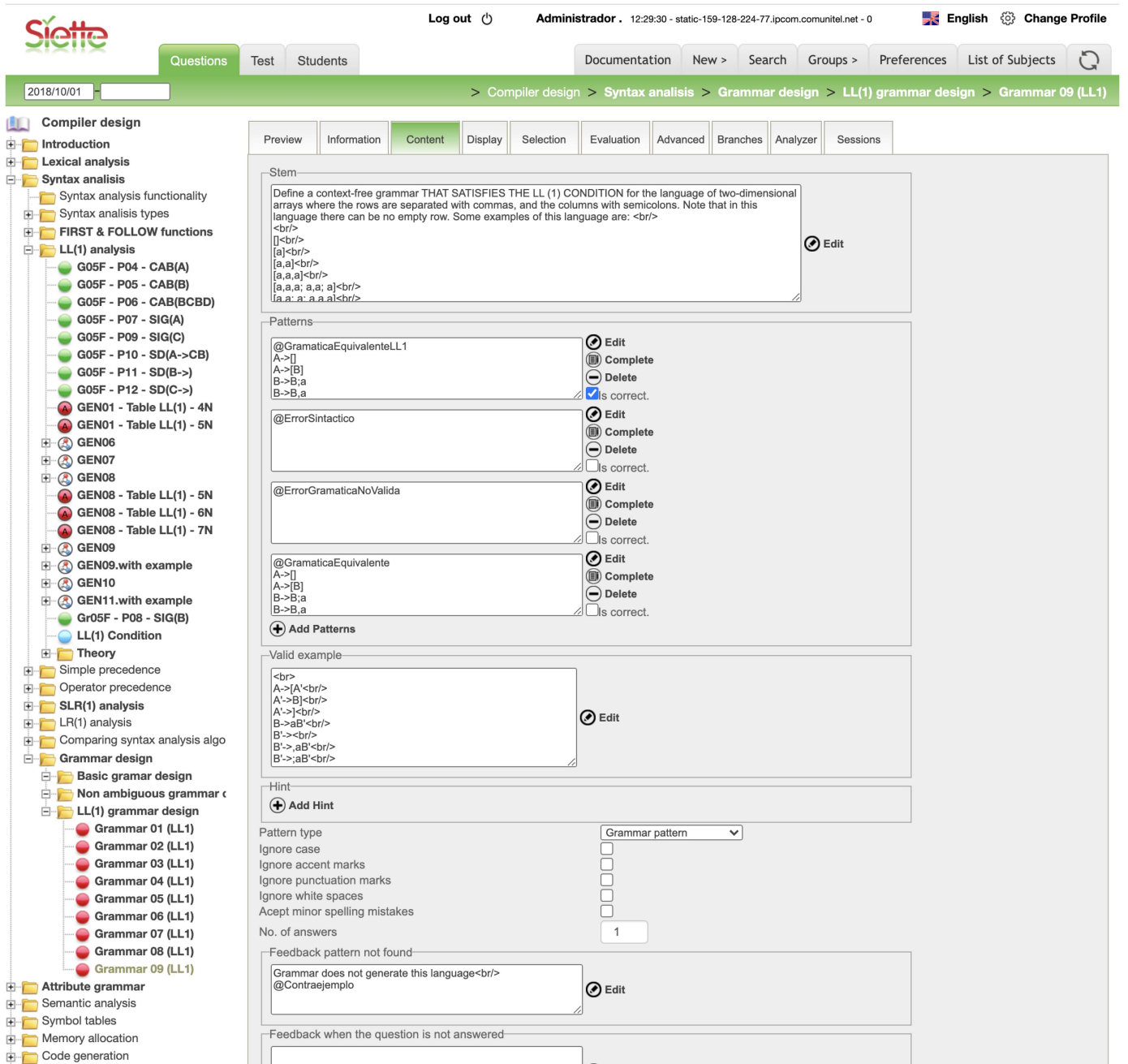


Fig. 2. Screenshot of the SIETTE authoring tool, editing a question that requires the design of an LL(1) CFG.

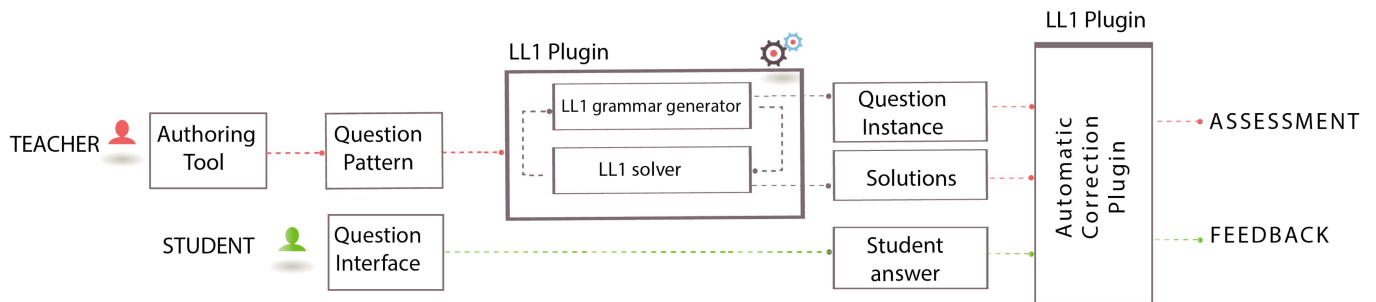


Fig. 3. Workflow of the SIETTE CFG plug-in.

Question number 1: GEN01 (1298726)

Given the grammar:

S → A\$
 A → ab
 A → ABb
 A → ε
 B → aB
 B → ABB
 B → b

NOTES:

- In the answers, write the list of yes symbols separated by blanks.
- If necessary, to write the empty string ε in the response, you must write EPSILON

Question number 1.1: GEN01-1 (1298762)

Find **FIRST(A)**

a EPSILON

[a, b, EPSILON]

Question number 1.2: GEN01-2 (1298798)

Find **FIRST(B)**

a b

Question number 1.3: GEN01-3 (1298834)

Find **FOLLOW(A)**

b

[a, b, \$]

Fig. 4. Question about FIRST and FOLLOW sets.

(or alternatively the number of production rules in the final grammar) and the final number of *terminal* symbols (which will randomly merge two symbols until the desired number of *terminal* symbols is met).

Finally, a validation and refinement process is triggered to eliminate unused rules or symbols, and/or duplicate rules, to guarantee that the CFG is correct.

2) *Automatic Construction of LL(1) Parsers:* This is the easiest phase. Given a CFG, it is always possible to compute FIRST and FOLLOW functions and obtain the directive symbols of each production rule [1]. The result of these functions are a set of symbols. Determining if a CFG accomplishes the LL(1) condition depends on these sets.

The first type of the system questions (Q1, Q2, and Q3) is related to the construction of these sets. Students should be able to obtain these sets by mentally applying the algorithm for small CFGs. Student answers can be checked against the correct answer using standard SIETTE regular expressions patterns. The student response can shuffle the order of *nonterminal* symbols in the set, but the pattern will recognize the answer anyway. Fig. 4 presents a composed question where a common grammar has

Giving the grammar:

S → A\$
 A → ε
 A → Bc
 A → ac
 B → bC
 B → cadC
 C → ε
 C → bC

Find the LL(1) table

| | a | b | c | d | \$ |
|---|-----|-----|---------|---|-----|
| S | A\$ | A\$ | A\$ | | A\$ |
| A | ac | Bc | Bc | | |
| B | | bC | cadC | | |
| C | | bC | EPSILON | | |

NOTES:

- To write down the empty string you should type EPSILON.
- Complete the LL(1) table for this grammar leaving blank the error cells. If there are conflicts, write down both rules consequents separated by a comma.
- This exercise is corrected by rows, assigning a partial credit to each correct table row

Giving the grammar:

S → A\$
 A → ε
 A → Bc
 A → ac
 B → bC
 B → cadC
 C → ε
 C → bC

Find the LL(1) table

| | a | b | c | d | \$ |
|---|-----|-----|------|---|-----|
| S | A\$ | A\$ | A\$ | | A\$ |
| A | ac | Bc | Bc | | |
| B | | bC | cadC | | |
| C | | bC | ε | | |

NOTES:

- To write down the empty string you should type EPSILON.
- Complete the LL(1) table for this grammar leaving blank the error cells. If there are conflicts, write down both rules consequents separated by a comma.
- This exercise is corrected by rows, assigning a partial credit to each correct table row

Fig. 5. Example of the LL(1) table response layout. On the left is the response of the student, and on the right is the correction given by the system.

been generated, and some questions about FIRST and FOLLOW sets are posed. Each question is evaluated independently.

Question types Q4 and Q5 require the implementation of the LL(1) parsing algorithm, which is based on a bidimensional table, that is given by function $LL(1)table : N \times T \rightarrow (N \cup T)^*$. This table is asked by question type Q4 and requires its own answer interface. The application of this algorithm to an input string is based on a stack whose content is asked in question type Q5. Fig. 5 presents the answer layout filled with the response of a student and the correction given by the system. The item is corrected by each row, with partial credit if the answer is partially correct.

3) *Automatic Generation of an Equivalent LL(1) Grammar:* In general, it is undecidable if there exists an LL(1) grammar that generates a given CFL, or to find an equivalent LL(1) grammar of a given CFG [22]. This does not mean that it is impossible to do it, or that there is no way to construct LL(1) grammar for a given language. It does imply that it will not be possible to do it for ANY language.

There are two interesting cases where a non-LL(1) grammar can be heuristically transformed into an equivalent grammar that

Define a context-free grammar THAT SATISFIES THE LL (1) CONDITION for the language of two-dimensional arrays where the rows are separated with commas, and the columns with semicolons. Note that in this language there can be no empty row. Some examples of this language are:

```

[]
[a]
[a,a]
[a,a,a]
[a,a,a; a,a; a]
[a,a; a; a,a,a]
...
    
```

✓

```

A->[A'
A'->B]
A'->]
B->aB'
B'->
B'->,aB'
B'->:aB'
        
```

✗

```

A->[L]
L->L,a
L->L;a
        
```

The grammar that has been read is not valid, it may contain inaccessible nonterminal symbols or it may not generate any language strings, or it may simply not be well written.

✗

```

A->[L]
L->L,a
L->L;a
L->a
        
```

Grammar does not generate this language
The grammar does not generate the sentence [] that belongs to the language.

✗

```

A->[L]
A->[]
L->L,a
L->L;a
L->a
        
```

The proposed grammar generates this language but it is not LL(1)

Fig. 6. Example of different answers to the same question (one correct and three incorrect) and different feedback generated by the system according to the answer.

might be LL(1): 1) factorization and 2) left recursive elimination [1]. For instance, the grammar

$$\begin{aligned}
 S &\rightarrow A\$ \\
 A &\rightarrow Aba \\
 A &\rightarrow a
 \end{aligned}$$

can be transformed by eliminating the left recursion to its equivalent

$$\begin{aligned}
 S &\rightarrow A\$ \\
 A &\rightarrow aA' \\
 A' &\rightarrow baA' \\
 A' &\rightarrow \epsilon.
 \end{aligned}$$

Both grammars generate the same language (they are equivalent). The first one is not LL(1), but the second one is. This heuristic transformation does not guarantee that the resulting grammar will always accomplish the LL(1) condition in the general case, even if we apply the heuristic transformation repeatedly over the resulting grammar.

However, we do not need to worry about the general case; there are plenty of cases in which the heuristic construction works, and those are the ones that we will use for assessment. Using this strategy, we can generate the type of questions shown in Fig. 6, that is, given an automatic generated CFG that is not LL(1), find an equivalent LL(1) grammar.

4) *Checking Grammar Equivalence*: Nevertheless, we have to face another undecidable problem if we want to recognize the student response. Given two CFGs, it is undecidable if they are

equivalent [23]. Nevertheless, this result does not discourage us to try to solve the problem using brute force algorithms for the small cases that are used for assessment.

Given a CFG $G(N, T, P, S)$, $L^n(G)$ is defined as the set of sentences of length $\leq n$; $L_k(G)$ is defined as the set of sentences that can be obtained with a maximum of k left derivations; and $L_k^n(G)$ is defined as the set of sentences of length $\leq n$ that can be obtained with a maximum of k left derivations. Of course, $L_k^n(G) \subseteq L^n(G)$ and $L_k^n(G) \subseteq L_k(G)$.

In order to check if two grammars G and G' are equivalent, we check that $L_{k_1}^n(G) \subseteq L_{k_2}^n(G')$ and $L_{k_1}^n(G') \subseteq L_{k_2}^n(G)$, where $k_1 \ll k_2$, that is, we check that all sentences of length n that can be obtained with k_1 left derivations using grammar G can also be obtained using k_2 left derivations using grammar G' and vice versa. These conditions are necessary (but not sufficient) to guarantee that G and G' are equivalent.¹

However, grammars used for student exercises are relatively small. They commonly have no more than four or five nonterminal symbols and no more than ten production rules. Production rule consequents are composed of no more than three or four symbols. In this case, the number of possible sentential forms that can be obtained from the grammar axiom is limited by the upper bound l^k , where k is the depth of the tree and l the maximum length of a production rule consequent. The maximum length of a sentential form can be upper bound by $l \times n$. On average, rule consequents' lengths are about 2, so on average, the number of sentential forms is around 2^n , which is a number that can be computed for values of $k < 20$ ². To sum up, in practical sets, using $n \leq 10$, $k_1 = 10$, and $k_2 = 20$, all cases have been successfully solved in less than a few seconds.³

In addition, to determine the equivalence of the grammar provided by the student and that proposed by the teacher, the plug-in check if the grammar given as an answer is ambiguous and if it accomplishes the LL(1) condition. This checking speeds up the assessment process in case of incorrect answer and provides detailed feedback to the student.

The grammar proposed by the teacher can be manually written or automatically generated, as described in Section III-B1. In the last case, the system randomly generates grammars by composition and guarantee that they can be transformed into LL(1) grammars using heuristic transformations.

The plug-in defines different response patterns that are given to the system using the SIETTE authoring tool. These patterns are compared to the student answer to determine which one corresponds to the answer. For instance, the patterns given for the exercise in Fig. 2 are labeled “@SyntaxError,” “@Incorrect,” “@Equiv <G>,” and “@LL1Equivalent<G>,” plus the “Not recognized” default pattern, where <G> stands for a grammar that generates the language. SIETTE tries to match each pattern

¹The computation of $L^n(G)$ might be more complex or incomplete if the grammar has empty rules. On the other hand, the computation of $L_k(G)$ might involve too many sentential forms. Limiting both makes the problem affordable.

²The computation of $L_k^n(G)$ can be improved by pruning those branches where there are more than n terminal symbols in the sentential forms.

³It is beyond the scope of this article to prove the minimum conditions that guarantee this assertion, nor to measure computational efficiency. These upper-bound limits have been obtained empirically according to the limitation of the system and the user interaction.

one by one. First, it checks the syntax of the answer and the validity of the student grammar. If none of this matches, the student answer is a correct CFG. The remaining patterns check if the grammar is equivalent to the given grammar and it accomplishes the LL(1) condition. This cascade of checking allows the system to provide the appropriate feedback to the student. Fig. 6 shows four different answers to the same question and different feedback generated by the system according to the answer.

On the left, Fig. 6 shows the stem and the right answer. On the upper right, it shows a invalid grammar; in the middle right, it shows a grammar that does not generate the same language (including an counterexample string); in the lower right, it shows an equivalent grammar that does not satisfy LL(1) condition. The teacher can select to also show the correct solution.

5) *Practical Issues:* There is no specific limitation on the number of rules that the grammar can contain, although since it is an application with didactic objectives, it does not make sense to use too many rules. In practice, all the grammars used either manually or automatically generated have less than ten production rules, less than seven terminal symbols, and less than seven nonterminal symbols. In the case of LL(1) tables, the limit would be in tables with a 7×7 dimension, which encompasses most of the practical teaching cases. In the case of SLR(1) tables, an exact number of N states can be imposed when defining the problem. In this case, the system generates grammars and calculates the tables until obtaining a problem that results in a table with N states. The teacher can also specify a range of the number of states of the SLR tables. In all the cases used, the number of states varies between 10 and 15, to be accomplished by the students in less than 30–45 min. The processing time of each exercise is variable. When configuring the exercise, the teacher has the possibility of testing it and adjusting the parameters using the SIETTE authoring tool so that the system response occurs in a reasonable time (of the order of a few seconds at most). During the use of the plug-in in evaluation sessions, usability tests were conducted with 150 students during an hour-long exam with acceptable performance. However, this limitation can be easily overcome by adding new SIETTE system servers that is designed to run on a cluster.

IV. METHODOLOGY

The plug-in was developed in 2016 and has been extensively used for formative and summative assessment for seven academic years for the Compiler Construction course at the Schools of Computer Engineering at the University of Málaga. Three types of exercises have been created. The first of them encompasses questions Q1, Q2, and Q3 (FIRST&FOLLOW); the second type contains questions Q4, Q5, and Q6 [LL(1)]; and the third type of exercise is dedicated to questions Q7–Q10 [SLR(1)]. For each type of exercise, a test created by the teacher and one automatically generated have been created in SIETTE. In the authored by the teacher test, questions have been manually introduced and are fixed (always the same grammar). In the autogenerated test, the SIETTE plug-in creates a new grammar each time the student accesses the test. For each type of exercise, students have initially received a series of theory classes where

concepts, problem types, and their solving methods have been explained. Subsequently, students have attended a practical class where they were instructed on the usage of the SIETTE tool and the functioning of each of the described exercises. Upon completion of this explanation, students have had access to the corresponding exercise in SIETTE for that session. From that moment on, access to these exercises in SIETTE has been available to the students even outside the classrooms. Once all theory classes and practical sessions have been conducted, the students take a summative test in which they use the SIETTE tool to solve exercises chosen from these three types.

All assessments conducted by the students (both formative and summative) are recorded by the SIETTE tool. From these data, a series of indicators has been selected to address the hypothesis posed in this section. The selected indicators are as follows: interval between assessments calculated from the date and time of each assessment recorded by SIETTE; assessment duration; and grade obtained in each assessment.

Sessions with a duration shorter than necessary to properly understand all questions and perform the necessary calculations to answer them have been discarded.⁴ Sessions with a duration longer than 3 h have also been discarded as they are assumed to be abandoned by the students.

With this study, we aim to conduct a comparative statistical analysis of the usage and results obtained from both types of tests by students. The null hypothesis we intend to test is that there is no significant difference in the usage or results between autogenerated tests and teacher-authored tests. In other words, we posit that students do not exhibit a preference for one type of test over the other, and that both are equally effective in assessing learning. The comparison of the different collected indicators has been conducted using a Wilcoxon signed-rank test and calculating the Spearman correlation coefficient.

In order to compare the effectiveness and performance of automatically generated exercises versus nongenerated exercises, a series of analyses have been carried out on different indicators of the two types of exercises. The information from the sessions carried out by the students on these exercises has also been cross-referenced with the grades obtained on the corresponding topic of the subject's midterm exam.

V. RESULTS

Table I shows the number of students and the number of sessions performed for each test from 2016 to 2022.

A first analysis of the evolution of the grade obtained in each session and the time used shows how, as students devote more sessions, the time needed to complete them decreases and the achieved grade increases. This behavior is similar for both automatically generated and teacher-authored exercises. Fig. 7 shows the evolution for each of the exercises.

Some interesting results can be observed considering the information in Fig. 7.

- 1) The number of students that keep making attempts decreases with the number of attempts (possibly because

⁴Students sometimes simply connect to the system to retrieve the questions and do not try to complete the task.

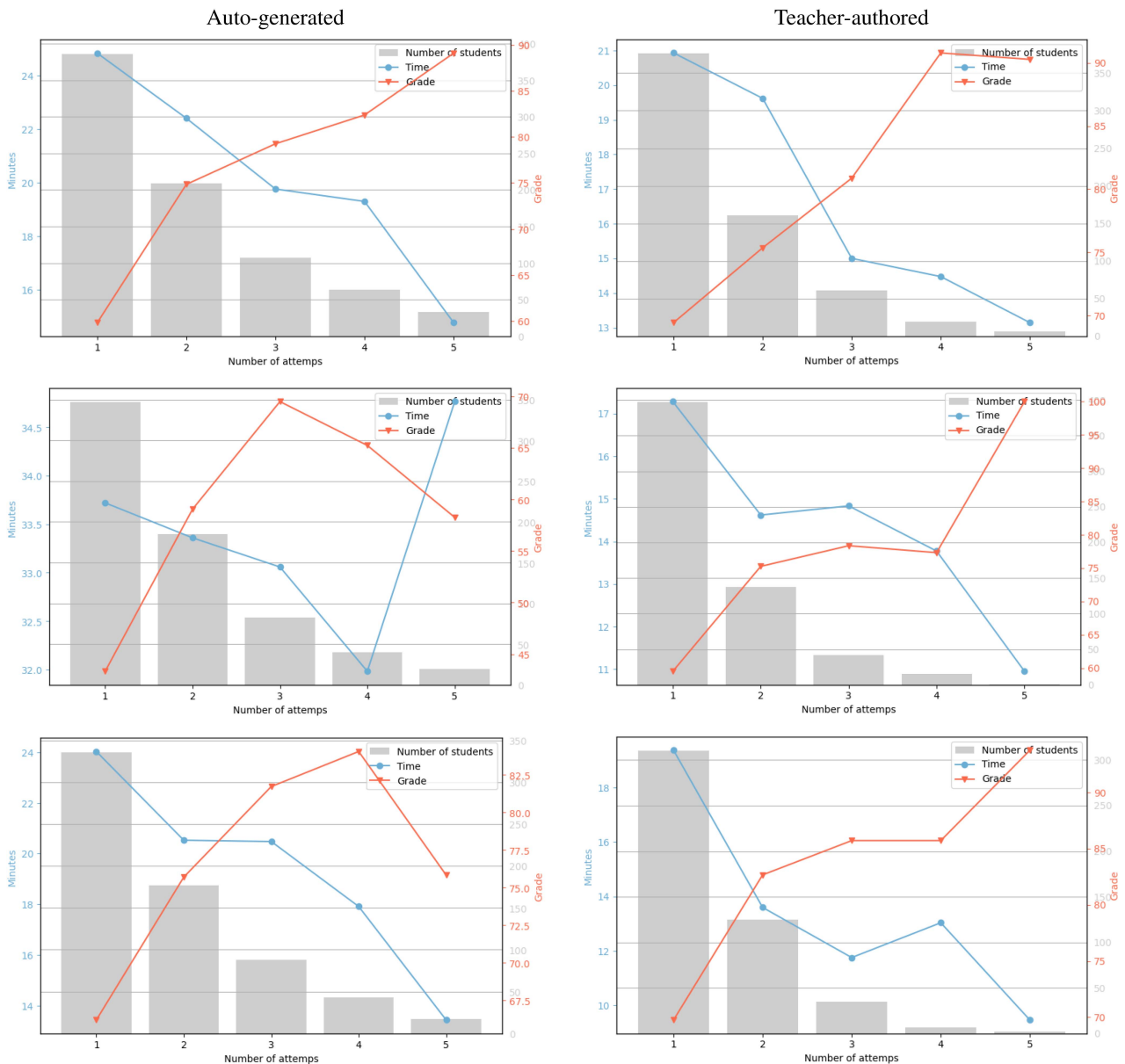


Fig. 7. Description of results achieved by students at the different attempts they made for every task: average grade (triangle line, scale on the right) and average assessment duration (dotted line, scale on the left). The first line corresponds to the FIRST&FOLLOW exercises, the second to the exercises on LL(1) grammars, and the third to the exercises on SLR(1) grammars.

TABLE I
TOTAL NUMBER OF STUDENTS AND SESSIONS POSED FROM THE ACADEMIC YEARS 2016–2022

| Task | Exercise type | #Students | #Sessions |
|--------------|------------------|-----------|-----------|
| FIRST&FOLLOW | Teacher-authored | 416 | 784 |
| | Autogenerated | 392 | 876 |
| LL(1) | Teacher-authored | 431 | 737 |
| | Autogenerated | 359 | 708 |
| SLR(1) | Teacher-authored | 310 | 479 |
| | Autogenerated | 336 | 661 |

Q1, Q2 and Q3 question types are grouped in the first line of the table. Q4, Q5, and Q6 question types are grouped in the second line of the table. Q7, Q8, Q9, and Q10 question types are grouped in the third line of the table.

they estimate that they have learned and do not need more training).

- Grades increase with the number of attempts (because students increase expertise about the tool, the types of questions, and the subject itself).
- The duration of attempts decreases with the number of attempts (as above, because students increase expertise about the tool, the types of questions, and the subject itself).

Regarding result 1), it has been interesting to observe how students do not aim to obtain a perfect result when completing exercises. The number of attempts for each exercise is not limited. Therefore, theoretically, a student could keep repeating an exercise until obtaining a perfect result. In the case of the

TABLE II
AVERAGE OF THE BEST GRADE OBTAINED BY EACH STUDENT IN EACH TYPE OF EXERCISE AND NUMBER OF ATTEMPTS

| Task | Autogenerated | | Teacher-authored | |
|--------------|---------------|-------------|------------------|-------------|
| | Grade | #Attempts | Grade | #Attempts |
| FIRST&FOLLOW | 76.15 (25.82) | 2.06 (1.29) | 79.35 (20.25) | 1.65 (0.92) |
| LL(1) | 57.44 (26.70) | 1.94 (1.15) | 67.7 (29.99) | 1.49 (0.79) |
| SLR(1) | 80.53 (21.24) | 1.97 (1.17) | 79.03 (20.96) | 1.54 (0.77) |

The standard deviation is in parentheses.

self-generated exercises, this would imply a complete mastery of the evaluated knowledge. In the case of exercises created by the teacher, this would be even easier. In this type of exercise, the student would not need to master all the required knowledge correctly. Since the grammar does not change from one attempt to another, by using a trial-and-error method, the student could correctly solve the exercise.

However, it has been observed that, in practice, this does not happen. Instead, the student repeats the exercise until they obtain a grade they consider adequate. In Table II, the average of the best grade obtained by each student for each type of exercise can be seen. It can be observed that the grade obtained is slightly better in the case of exercises created by the teacher. Actually, this difference is only statistically significant in the case of LL(1) exercises where a Wilcoxon signed-rank test with a p -value of $8.78e-09$ shows evidence that students achieve higher scores in teacher-authored tests than in autogenerated ones. In contrast, the FIRST&FOLLOW and SLR(1) tests, with p -values of 0.64 and 0.30, respectively (greater than 0.01), there is no statistically significant evidence that the scores differ.

In Table II, it can also be observed that the number of attempts is slightly lower for teacher-authored tests than for autogenerated ones. This difference is statistically significant in all cases with p -values of $2.35e-05$, $8.58e-05$, and $1.03e-05$, respectively, using a Wilcoxon signed-rank test (all less than 0.01). This information agrees with the fact that exercises created by the teacher maintain the same grammar from one attempt to another and, therefore, can be solved by trial and error in fewer attempts. This indicates that the behavior of students in both types of exercises is similar, and they stop trying to improve the exercise's result after reaching a certain grade.

Regarding results 2) and 3), it can be observed that, in the case of teacher-authored exercises, both the grade obtained and the duration are monotonically increasing and decreasing, respectively. This is easily explained since, being always the same grammar in each attempt, the student uses the information from the previous attempt to improve the new attempt without having to fully master the evaluated knowledge.

On the other hand, in the case of autogenerated exercises, the average grade obtained by students in a number of attempts may be lower than for the previous number of attempts. Similarly, the average duration may increase for the next number of attempts. This can be explained by considering that the grammars are different for each student in each attempt. Thus, the increase in the obtained grade depends only on learning. In addition, as explained earlier, since students who reach a certain level in the grade stop making new attempts, the students who continue to

perform the exercise beyond a certain number of attempts are those who have not yet acquired the necessary knowledge and, therefore, decrease the average grade obtained for that number of attempts.

This can be considered an advantage of autogenerated exercises over teacher-authored exercises. Autogenerated exercises require a better mastery of the evaluated knowledge to progress in the exercise. They also do not allow for trial and error resolution. For this reason, the student cannot think that they completely dominate the exercise without having learned the necessary knowledge. Another added advantage is that it encourages more work on the part of the student and eliminates the tedium of repeating the same grammar over and over again by presenting new challenges in each attempt.

It has also been observed that the pace at which students engage in sessions, that is, the time between one session and another, decreases as the student uses the tool. In this way, initially, the student returns to using the tool within a short period of time after the previous session. As they go through new sessions, that period increases. In this regard, a high correlation has been found between the number of sessions and the elapsed time between each session. A Spearman correlation coefficient of 0.69 has been calculated between the number of previous sessions and the interval between automatically generated sessions. Moreover, this coefficient has a statistical significance of 0.001. Similarly, the Spearman correlation coefficient between the number of previous sessions and the interval between sessions in teacher-authored sessions is also 0.69, with a statistical significance of 0.001. Therefore, it is not surprising that the Spearman correlation coefficient between the interval of automatically generated sessions and teacher-authored sessions is 0.62, with a significance of 0.001. This indicates that the pace at which students use this tool is similar between automatically generated tests and teacher-authored tests.

On the other hand, the relationship between the grades obtained in the practice tests (formative evaluation) and the grade obtained in the evaluation test (summative evaluation) of the same subject has also been analyzed. These tests has been, in some cases, teacher-authored and, in other cases, automatically generated. Although a statistically significant positive correlation (0.001) is observed between the results of the practice tests and the evaluation tests, this correlation is not very high, with a Pearson coefficient around 0.29.

Furthermore, using a Wilcoxon signed-rank test, it has been found that the results of the formative tests are statistically significantly higher than those obtained in the evaluation test for the FIRST&FOLLOW and SLR(1) tests. These tests have

shown a statistically significant difference with p -values of $3.92e-07$ and $7.02e-15$ for the generated and teacher-authored FIRST&FOLLOW tests, respectively, and $1.38e-08$ and 0.0005 for SLR(1). However, for the LL(1) tests, the results of the automatically generated formative tests are lower than those obtained in the evaluation test with a p -value of $1.50e-17$, while for the teacher-authored tests, no statistically significant differences have been found. In the case of the FIRST&FOLLOW tests, this result is expected given that these exercises are easier than the LL(1) and SLR(1) exercises that are part of the evaluation test. As for the SLR(1) tests, the result may be due to the time limit imposed in the evaluation test.

In summary, using the Wilcoxon signed-rank statistical test and the Spearman correlation coefficient, it has been found that the use of self-generated tests by students is not different from the use of teacher-authored tests. The comparison of the scores obtained and the number of sessions has shown that students repeat the tests until they achieve a similar score in both types of tests. It has also been observed that there is a reasonable relationship between the scores in the formative tests and the summative assessments.

VI. CONCLUSION

The application described in this article provides a way for the student to enhance the practice of design of the LL(1) and SLR(1) CFGs. The automatic question generation based on a combination of building block grammar ensures an unlimited number of problems to be posed. Although the generation of an LL(1) grammar and the recognition of grammar equivalence are unsolvable issues in the general case, a heuristic approach can provide a practical solution for assessment purposes.

The application has been designed and used for formative and summative assessment. It includes the automatic recognition of student answers and personalized feedback. We do not claim that the system itself is responsible for the increase in the student scores, but the data show that it helps students to practice and be aware of their progress.

The complexity and difficulty of the automatically generated questions is similar to that of the questions manually proposed by the teacher. The former also have the advantage of always being different, which is why they require in-depth learning of the domain concepts from the student and not simple recall of a specific case.

According to our experience, this tool is especially suitable for stand-alone formative assessment. Once the techniques have been presented and some examples have been completed in the classroom, this tool is ideal for students to practice freely at home. The order of presentation of the exercises that can be constructed using this plug-in coincides with the order of presentation of the course, and it is listed at the end of Section I. This is a complementary tool to others that offer simulation of the construction of LL(1) and SLR(1) analysis tables. In case of using both tools, the simulation tool could be used previously for training, or afterward to obtain a step-by-step explanation about a specific case. A possible line of future work would be

the integration of an assessment tool that provides feedback in a graphical and more detailed way.

The tool has also been used in exams (summative evaluation). In this case, it is advisable that students should have used the tool previously to avoid any cognitive overload in the construction of their answers. It is also advisable to fix the total number of rules, the number of terminals and nonterminals, and/or the size of the tables, so that the difficulty of the questions generated would be the same.

The application is embedded in the SIETTE system, which can provide additional features that can be used, such as adaptive question selection, scoring procedure selection, access control, etc. Question difficulty can be controlled by means of the number of building block grammar combinations, but it can also be obtained empirically through SIETTE question calibration and learning analytic tools. As future work, we plan to fit the parameters of the IRT characteristic curves of the questions and test whether SIETTE's adaptive mechanisms improve students' learning results. On the other hand, although the system is limited to SLR(1) tables, we are planning, as future work, to extend its functionality to LR(1) and LALR(1) tables.

APPENDIX

LINKS TO ONLINE ASSESSMENTS (ANONYMOUS LOGIN)

FIRST and FOLLOW: <https://www.siette.org/siette?idtest=631978&anon>

LL(1) grammar analysis: <https://www.siette.org/siette?idtest=633742&anon>

LL(1) table construction: <https://www.siette.org/siette?idtest=525382&anon>

LL(1) grammar design: <https://www.siette.org/siette?idtest=633700&anon>

SLR(1) closure function: <https://www.siette.org/siette?idtest=775618&anon>

SLR(1) goto function: <https://www.siette.org/siette?idtest=775660&anon>

SLR(1) grammar analysis: <https://www.siette.org/siette?idtest=815098&anon>

SLR(1) tables construction: <https://www.siette.org/siette?idtest=775702&anon>

REFERENCES

- [1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques, and Tools*. Boston, MA, USA: Addison-Wesley, 1986.
- [2] M. Memik and V. Zumer, "An educational tool for teaching compiler construction," *IEEE Trans. Educ.*, vol. 46, no. 1, pp. 61–68, Feb. 2003.
- [3] Á. Arnaiz-González, J.-F. Díez-Pastor, I. Ramos-Pérez, and C. García-Osorio, "Seshat—A web-based educational resource for teaching the most common algorithms of lexical analysis," *Comput. Appl. Eng. Educ.*, vol. 26, no. 6, pp. 2255–2265, Jul. 2018.
- [4] J. Castro-Schez, C. Glez-Morcillo, J. Albusac, and D. Vallejo, "An intelligent tutoring system for supporting active learning: A case study on predictive parsing learning," *Inf. Sci.*, vol. 544, pp. 446–468, 2021.
- [5] J. Sanyu, E. Bainomugisha, and B. Kanagwa, "Teaching language processing with the PAMOJA framework," *Sci. Comput. Program.*, vol. 229, 2023, Art. no. 102959.
- [6] S. Stamenković and N. Jovanović, "A web-based educational system for teaching compilers," *IEEE Trans. Learn. Technol.*, vol. 17, pp. 143–156, 2024, doi: [10.1109/TLT.2023.3297626](https://doi.org/10.1109/TLT.2023.3297626).

- [7] R. Conejo, J. del Campo-Ávila, and B. Barros, "First steps towards automatic question generation and assessment of LL (1) grammars," in *Proc. Int. Conf. Artif. Intell. Educ.*, 2022, pp. 271–275.
- [8] S. Stamenković, N. Jovanović, and P. Chakraborty, "Evaluation of simulation systems suitable for teaching compiler construction courses," *Comput. Appl. Eng. Educ.*, vol. 28, no. 3, pp. 606–625, Mar. 2020.
- [9] D. Rodríguez-Cerezo, M. Gomez-Albarran, and J.-L. Sierra, "From collections of exercises to educational games: A process model and a case study," in *Proc. IEEE 11th Int. Conf. Adv. Learn. Technol.*, 2011, pp. 282–284.
- [10] R. Jabri, "A generic tool for teaching compilers," *Comput. Inf. Sci.*, vol. 6, no. 2, pp. 134–150, Apr. 2013, doi: [10.5539/cis.v6n2p134](https://doi.org/10.5539/cis.v6n2p134).
- [11] R. Cedazo, C. E. Garcia Cena, and B. M. Al-Hadithi, "A friendly online C compiler to improve programming skills based on student self assessment," *Comput. Appl. Eng. Educ.*, vol. 23, no. 6, pp. 887–896, May 2015, doi: [10.1002/cae.21660](https://doi.org/10.1002/cae.21660).
- [12] J. D. Velasquez, "Automatic assessment of programming projects in a compiler construction course," *IEEE Latin Amer. Trans.*, vol. 16, no. 12, pp. 2904–2909, Dec. 2018.
- [13] L. Rüdian and N. Pinkwart, "Towards an automatic Q&A generation for online courses—A pipeline based approach," in *Proc. Int. Conf. Artif. Intell. Educ.*, 2019, pp. 237–241.
- [14] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, "A systematic review of automatic question generation for educational purposes," *Int. J. Artif. Intell. Educ.*, vol. 30, no. 1, pp. 121–204, Nov. 2019.
- [15] B. Das, M. Majumder, S. Phadikar, and A. A. Sekh, "Automatic question generation and answer assessment: A survey," *Res. Pract. Technol. Enhanced Learn.*, vol. 16, no. 1, Mar. 2021, Art. no. 5.
- [16] M. Hernando, E. Guzmán, and R. Conejo, *Measuring Procedural Knowledge in Problem Solving Environments With Item Response Theory*. Berlin, Germany: Springer, 2013, pp. 653–656.
- [17] R. Conejo, E. Guzmán, and M. Trella, "The SIETTE automatic assessment environment," *Int. J. Artif. Intell. Educ.*, vol. 26, no. 1, pp. 270–292, 2016.
- [18] Z. Wang and S. J. Osterlind, *Classical Test Theory*. Rotterdam, The Netherlands: Sense Publishers, 2013, pp. 31–44.
- [19] W. Van der Linden, *Handbook of Item Response Theory, Models* (ser. Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences), vol. 1. London, U.K.: Chapman & Hall/CRC Press, 2017.
- [20] W. Linden and C. Glas, "Computerized adaptive testing: Theory and practice." Accessed: Mar. 2020. [Online]. Available: [http://lst-iiep.iiep.unesco.org/cgi-bin/wwwi32.exe/\[in=epidoc1.in\]/?t2000=012104/\(100](http://lst-iiep.iiep.unesco.org/cgi-bin/wwwi32.exe/[in=epidoc1.in]/?t2000=012104/(100)
- [21] R. Conejo, "Siette manual: SIETTE regular expression pattern," 2023. [Online]. Available: <https://wiki.siette.org/doku.php?id=es:manual:items:patron:siette>
- [22] D. J. Rosenkrantz and R. E. Stearns, "Properties of deterministic top down grammars," in *Proc. 1st Annu. ACM Symp. Theory Comput.*, New York, NY, USA: 1969, pp. 165–180.
- [23] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Boston, MA, USA: Course Technology, 2013.



Ricardo Conejo Muñoz was born in Archidona, Málaga, Spain, in 1960. He received the M.S. and Ph.D. degrees in Ingeniero de Caminos, Canales y Puertos (Civil engineer) from the Technical University of Madrid, Madrid, Spain, in 1983 and 1995, respectively.

He was the Director of the Artificial Intelligence Research Group, University of Málaga, Málaga, and the Director and Main Developer of the SIETTE system. Since 1986, he has been with the Department of Computer Science and Programming Languages,

University of Málaga, where he is currently a Full Professor. He has taught compilers and programming for more than 30 years. His research interests include adaptive testing, student knowledge diagnosis, and intelligent tutoring systems, as well as fuzzy logic, model-based diagnosis, multiagent systems, and artificial intelligence applied to civil engineering.

Dr. Conejo Muñoz was an Associate Editor for IEEE TRANSACTIONS ON LEARNING TECHNOLOGIES. He is a Regular Member of program committees of international conferences, such as ACM Conference on User Modeling Adaptation, and Personalization, International Conference on Intelligent Tutoring Systems, and International Conference on Artificial Intelligence in Education.



Beatriz Barros Blanco received the Ph.D. degree in computer science (artificial intelligence) from the Polytechnic University of Madrid, Madrid, Spain, in 1999.

Since 2017, she has been a Full Professor with the University of Málaga, Málaga, Spain. Her research interests include the design of adaptive and interactive learning environments, collaborative learning systems, and the development of efficient solutions for data analysis.



José del Campo-Ávila received the Ph.D. degree in software engineering and artificial intelligence from the University of Málaga, Málaga, Spain, in 2007.

He is currently an Associate Professor with the Department of Computer Science and Programming Languages, University of Málaga, where he is also a Member of the Artificial Intelligence Research Group. His research interests include incremental learning, data stream mining for classification, and multiple classifier systems.

Dr. del Campo-Ávila is an Associate Editor for the *International Journal of Data Science and Analytics*.



José L. Triviño Rodríguez was born in Málaga, Spain, in April 1972. He received the Diploma in computer science (with a specialization in cybernetics and theoretical computer science), and the M.S. and Ph.D. degrees in computer engineering from the University of Málaga, Málaga, in 1993, 1995, and 2002, respectively.

Since 2009, he has been an Associate Professor of languages and computer systems with the University of Málaga. His research interests include artificial intelligence, machine learning, and data analysis and

knowledge discovery through artificial intelligence.

Dr. Triviño Rodríguez is also a Member of the Spanish Association for Artificial Intelligence.