Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company's public news and information website.

Elsevier hereby grants permission to make all its COVID-19-related research that is available on the COVID-19 resource centre - including this research content - immediately available in PubMed Central and other publicly funded repositories, such as the WHO COVID database with rights for unrestricted research re-use and analyses in any form or by any means with acknowledgement of the original source. These permissions are granted for free by Elsevier for as long as the COVID-19 resource centre remains active.
An intelligent tutoring system for supporting active learning: A case study on predictive parsing learning

J.J. Castro-Schez a,⇑, C. Glez-Morcillo a, J. Albusaca a, D. Vallejo a

aUniversity of Castilla-La Mancha, Department of Technology and Information Systems, Escuela Superior de Informática, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain

ARTICLE INFO

Article history:
Received 3 October 2018
Received in revised form 26 June 2020
Accepted 22 August 2020
Available online 17 September 2020

Keywords:
Online learning
Intelligent tutoring systems
Predictive parsing
Formal language processing

ABSTRACT

The way in which people learn and institutions teach is changing due to the ever-increasing impact of technology. People access the Internet anywhere, anytime and request online training. This has brought about the creation of numerous online learning platforms which offer comprehensive and effective educational solutions which are 100% online. These platforms benefit from intelligent tutoring systems that help and guide students through the learning process, emulating the behavior of a human tutor. However, these systems give the student little freedom to experiment with the knowledge of the subject, that is, they do not allow him/her to propose and carry out tasks on his/her own initiative. They are very restricted systems in terms of what the student can do, as the tasks are defined in advance.

An intelligent tutoring system is proposed in this paper to encourage students to learn through experimentation, proposing tasks on their own initiative, which involves putting into use all the skills, abilities tools and knowledge needed to successfully solve them. This system has been designed developed and applied for learning predictive parsing techniques and has been used by Computer Science students during four academic courses to evaluate its suitability for improving the student’s learning process.

1. Introduction

The importance and contribution of e-learning systems, used as a supplement in the learning process, is unquestionable in normal situations, and it is more obvious in cases of lockdown scenarios, such as the one we are experiencing now due to the CORONAVIRUS health crisis (COVID-19 pandemic). In this context, providing e-learning systems and promoting their use was one of the major challenges addressed by many universities.

Designing and developing systems which help or promote learning has always been a multi-disciplinary concern. Since the 90’s these have been based on Information and Communication Technologies (ICT), spurred on by the rapid evolution of the Internet and the state of the worldwide web today. There are numerous software systems aimed at assisting the teaching/learning process online (e.g. Learning Management System-LMS, Learning Content Management Systems-LCMS, Virtual Learning Environment-VLE, e-Learning) whose main purpose is to provide resources which enable skills, knowledge, and behaviors to be acquired by means of study, instruction, reasoning and observation.

⇑Corresponding author.
E-mail address: JoseJesus.Castro@uclm.es (J.J. Castro-Schez).

https://doi.org/10.1016/j.ins.2020.08.079
0020-0255/© 2020 Elsevier Inc. All rights reserved.
These systems usually enable courses to be created and listed in online learning platforms (Edx, Udacity, Udemy, Coursera, Khan Academy). An online course is made up of many different components and activities which, to a large extent, depend on the professor and the methodology he or she uses to achieve the learning outcomes. Some of these are lessons (with varying degrees of complexity), readings, images, audios, videos, games, debates, surveys and tests. Moreover, there are various communication tools which participants on the course share such as emails and discussion forums, among others. These systems also provide components for scheduling activities and monitoring students’ progress in them.

New components are being researched and developed, in order to make these systems more interesting to learners and also different from the others in the competition. These components focus on promoting active learning [1], i.e. they involve learners in their own learning process, and also on encouraging learners’ independence, i.e. they support the learning process without the need for instructor intervention. Ultimately, these systems should guarantee the autonomy of the learner, i.e. they should give the learner the opportunity to take charge of his or her own development.

In this paper an ITS has been designed and developed for providing a rich learning environment in which students can learn actively, freely creating exercises, which allow them to operate and manipulate subject matter concepts. The developed system behaves as an expert system simulating the behaviour of the instructor [2] who applies an active teaching methodology that promotes learning through experimentation. This type of teaching promotes active learning and learners’ autonomy, making the learning process easy, passionate and creative. When a task, however difficult, is carried out on one’s own initiative, all the capacities, skills, tools and knowledge necessary to solve it successfully are put to use. Thus, learning through experimentation provides a number of advantages to the learners such as to acquire and refine control knowledge, to augment an incomplete domain theory and to refine an incorrect domain theory [6].

Currently, existing ITS turn this learning through experimentation into “knowing through doing”. The following approaches can be found [7]:

First approach: Systems that extract and propose exercises taken from a repository to learners with the purpose that they solve them. Then, these systems check how good the solution given by the learners is and propose new exercises to them, which are always taken from the repository. These exercises are chosen on the basis of different criteria, such as their progress in the learning process or the results obtained in previous exercises [4,8–10].

Second approach: Systems that present to learners an exercise and how to solve it step by step (worked-examples). Then, these systems create similar exercises, usually changing data, which can be solved by following the learnt steps. In subsequent stages, they may also gradually fade out the number of steps there are in the worked solution (faded worked examples) so that the students can explain how they arrived to the final solution [11–14].

Research within both approaches focuses primarily on adapting the rules which recommend learning actions more than on giving the learner the freedom to experiment with [15–19]. Experimentation in both approaches is controlled, that is, the instructor establishes the exercises that learners should or can do. These learners have no freedom to propose new exercises that they might come up with. Recent research has focused on open learning environments, where the set of exercises is not limited to a predefined set, or where the learners can create new exercises [20–26]. These systems provide a language that enables the generation of exercises and the communication with the learners, which may be a standard language (e.g. Extensible Markup Language – XML) [22,26] or a Domain Specific Language (DSL) specifically designed for the problem to be solved [23–25]. On the other hand, there are systems that focus on the ability for the students to create exercises on their own initiative and upload them into the system, which will solve them and explain how the solutions are reached [20,21].

These last systems are undoubtedly a major advance in getting the learners to construct their own knowledge through experimentation, working with abstractions which require high level thinking skills. They enable the learners to work at higher levels of Bloom’s Taxonomy [27,28] and help them to gain independent thinking skills. However, these systems do not include an ITS module that adapts to the needs of the learners by making recommendations of the following activities to be done in order to guide their learning process, nor do they provide feedback on what they are doing right or wrong. Designing and developing ITS with this last approach is complex since they are usually highly-dependent on the subject and problems to which they are applied. Moreover, they must work with the exercises that the learner creates at any time, which is highly unpredictable.

In this paper an ITS has been designed and developed for providing a rich learning environment in which students can learn actively, freely creating exercises, which allow them to operate and manipulate subject matter concepts. Our proposal is specially focused on the tutoring module and how it corrects exercises, gives feedback and promotes progress in student learning. The developed system behaves as an expert system simulating the behaviour of the instructor [2] who applies an active teaching methodology that promotes learning through experimentation.

The developed ITS helps students learn predictive parsing algorithms, which are a type of exercises found in Compiler Design or Formal Language Processing subjects. These are part of Computer Science theory, and are included in the syllabuses for this degree at most universities.

The remaining sections have been organized as follows. In Section 2, the subject and type of exercises the ITS helps to learn will be briefly presented. In Section 3 the system set out herein will be developed with a special focus on the system architecture (Section 3.1), the interface module (Section 3.2) and tutor module (Section 3.3) with a strong focus on the way in which the system automatically corrects exercises (Section 3.3.3) and guides student learning by making recommendations (Section 3.3.4). In Section 4, there will be an initial evaluation of the system put forward herein. Lastly, in the final section of this paper, there will be a conclusion and future lines of work will be mentioned.
2. Context and problem statement

2.1. Compiler design

Compiler Design, in most curriculums at introductory level in Computer Science, is a highly complex and wide-ranging topic [29–32]. The contents of an introductory course in Compiler Design are usually organized into the following units:

U1. Introduction: An overview of compiler design is covered in this unit.
U2. Notation and concepts for formal languages and grammars: Basic notation and concepts related to discrete mathematics, grammars and languages are covered in this unit.
U3. Programming-language design: The items to be taken into account when designing and developing a language are presented in this unit (i.e. lexis or language vocabulary, the syntax or sentence structure, the semantic or sentence meaning).
U4. Lexical analysis: This unit focuses on designing and developing lexical analyzers (i.e. scanners).
U5. Syntax analysis: The focus of this unit is syntactic analysis techniques which are used to check whether a sentence has a valid structure or not. Top-down (LL(1)) and bottom-up (LR(0), SLR(1), LR(1), LALR(1)) predictive parsing techniques are studied.
U6. Semantic analysis or context sensitive analysis: In this unit what the grammatically correct sentences mean is studied, with special focus on syntax-directed translation.
U7. Synthesis task: This unit describes how the output of a compiler is generated, and the code generation and code optimization phases are presented.

In these introductory courses, the focus is on the analytical part of the compilation process, with greater emphasis on syntactic analysis (U5), since this provides basic knowledge about compiler theory. Therefore, this is the part which most time is dedicated to.

As a result of what they have learnt, students will gain a solid foundation in compiler design. They are expected to master the fundamental theories behind programming languages and lexical, syntactic and semantic processing techniques. Moreover, they will know how to use these when designing languages and developing processors associated with them. To be precise, students must understand the top-down parsing techniques (LL(1)), and the bottom-up or shift-reduce parsing techniques (SLR(1), LR(1) and LALR(1)), and develop processors based on these.

2.2. Key problem statement

In order to master the syntactic analysis phase and predictive analysis techniques, doing a large number of exercises about parsing is recommended. When students practice, they learn theory and acquire practical skills on parsing techniques. Developing an ITS that enables students to perform as many exercises as they want, as well as correcting them and providing guidance on the next steps in the learning process is essential in online learning. In addition, it could be used as an auxiliary tool for face-to-face studying, where it provides additional learning support for students and their professor.

The two types of parsing exercises the students must carry out are: LL and LR. Each exercise consists of a grammar and constructing a of a parser for it (LL(1), SLR(1), LR(1) and/or LALR(1)).

At this point, it may be beneficial for us to recall the classical definition of grammar, given by Noam Chomsky [33]. A formal grammar is the four-tuple $G = (N, T, P, S)$, where:

- $N$ is a set of non-terminal symbols (variables).
- $T$ is a set of terminal symbols (constants), so $N \cap T = \emptyset$.
- $S$ is a special symbol, the start symbol, so $S \in N$.
- $P$ is a finite set of production rules. Since we are going to work with context-free and regular grammars, each production rule $p_i \in P$ will have a structure similar to $A \rightarrow \alpha$, where $A$ is in $N$ and $\alpha$ is in $(T \cup N)^*$ with $^*$ being the Kleene’s closure.

The solution to an exercise will depend on the type of parser to be developed:

- For LL(1) parsers, it implies constructing a LL(1) parsing table, and for this purpose, the set of prediction symbols must be obtained for each production rule $p_i \in P$ (i.e. $\text{LOOKAHEAD}(p_i)$). The nullable symbols must be identified (i.e. those non-terminal symbols in the grammar that can be derived into the empty string $\lambda$) and FIRST and FOLLOW sets must be computed for each non-terminal symbol in the grammar. FIRST($\alpha$) is the set of terminal symbols in the grammar that begin the strings which are derivable from $\alpha$ (such that $\alpha \in (T \cup N)^*$) and FOLLOW($A$) is the set of terminal symbols in the grammar that can appear immediately to the right of $A$ in some partial derivation $(A \in N)$.
- For SLR(1), LR(1) and LALR(1) parsers, this entails constructing SLR(1), LR(1) and LALR(1) parsing tables. To do so, the automaton that recognizes viable prefixes and reduces them appropriately for deterministic SLR(1) or LR(1) parsing, must be built. The automaton for deterministic LALR(1) parsing is obtained from the LR(1) automaton.
These are the exercises we hope the student will practice with the ITS proposed in this paper. We will not explain here how to solve these exercises since any classical books on this topic includes it \cite{35,36,37}. However, we must point out that these exercises can be solved by means of algorithms.

2.3. Previous works

At present, a series of tools for this purpose have been proposed, designed and used \cite{35}. Among these we can mention ANAGRA, BURGRAM, JavaPars, PAVT, SEFALAS, JFLAP, VAST, CUPV, PPVT, LISA, VCOCO and PAG. Evaluators 2.0, VCOCO, Evaluators 2.0 and Proletool 2.0 depend on CUPV and COCO/R, which are responsible for creating analyzers, and they just visualize them in operation, the remaining tools do not depend on any others. Proletool 2.0, Proletool 2.0 or by means of graphic tools (JFLAP). Thirdly, they stand apart in their capacity to create visual representations which can be manipulated by students (ANAGRA, SEFALAS, Grammophone, LISA, VAST, PAG, PPVT, CUPV, VCOCO, Evaluators 2.0, Proletool 2.0) or by means of graphic tools (JFLAP).

There are several differences between these tools. Firstly, they differ in their execution environments, where we can find web applications (based on client-server architecture) (JavaPars, Grammophone, Proletool 2.0) or desktop applications (ANAGRA, BURGRAM, JFLAP, SEFALAS, VAST, CUPV, PPVT, LISA, VCOCO and PAG). Secondly, the ways exercises are entered into the tools are different, which can be done with a form (JavaPars, BURGRAM), by using a DSL (ANAGRA, SEFALAS, Grammophone, LISA, VAST, PAG, PPVT, CUPV, VCOCO, Evaluators 2.0, Proletool 2.0) or by means of graphic tools (JFLAP). Finally, the degree to which they depend on other tools varies. CUPV and VCOCO depend on CUP and COCO/R, which are responsible for creating analyzers, and they just visualize them in operation, the remaining tools do not depend on any others.

It should be noted that all the tools presented above allow the introduction of new exercises, but none of them can be considered to be an ITS, since they lack an essential component needed in these systems: the learning instructor. They are traditional computer aided instruction tools (CAI). Only one of them comes close to that idea, Proletool 2.0. This tool allows the student to propose solutions to the exercises and adds the possibility to make corrections (through the named exercises corrector component). The tool suggested in this paper will have models of instructional content that specify what to teach along with the teaching strategies that specify how to teach, incorporating within the latter the challenge for the student to propose new exercises that comply with certain restrictions (working at higher levels of Bloom’s Taxonomy). In addition, the proposed system should enable visualization and interaction mechanisms regarding the developed analyzers, showing how they work and how the syntax tree is constructed.

3. Intelligent Tutoring System for predictive parsing learning

3.1. ITS architecture overview

In this paper, an ITS, Proletool 3.0, is proposed for enabling students to understand the relationship between the phases of lexical and syntactic analysis and the technical fundamentals of top-down (LL(1)) and bottom-up or shift-reduce parsing techniques (SLR(1), LR(1) and LALR(1)). Moreover, this ITS will enable them to develop processors based on these techniques. The system proposed in this paper will contain a number of conceptual components, or modules, that interact with one another. Fig. 2 depicts the architecture of the system proposed herein which consists of five interrelated components: (i) domain module, (ii) tutor module, (iii) student module, (iv) evaluation module and (v) interface module. Let us briefly look at what each module consists of and does.

The interface module is responsible for enabling two-way communication between students and the system as well as facilitating student-system interaction in learning activities. Thus, it must allow exercises and solutions to be entered into the system. Additionally, it needs to have the facilities to simulate parsing techniques graphically.

The tutor module is made up of different components: (i) the problem solver component, which is responsible for correcting exercises; (ii) the exercises corrector component, whose task is to check how good the solution developed by the students to the exercises proposed is; and (iii) the learning instructor component, whose purpose is to provide feedback, explaining the process for finding the solution, the corrections made, and on the basis of this and the student profile, recommending new assignments.

---

1. URL: http://webdiis.unizar.es/ezpeleta/ (last access on 25/06/2020).
2. URL: http://cgosorio.es/BURGRAM/ (last access on 25/06/2020).
3. URL: http://paginapersonales.deusto.es/josuka/parser/parser.html (last access on 25/06/2020).
4. URL: http://www.jflap.org/ (last access on 25/06/2020).
5. URL: http://labraj.feri.umd.edu/lisa/ (last access on 25/06/2020).
6. URL: http://mdaines.github.io/grammophone/ (last access on 25/06/2020).
7. URL: http://portal.esi.uclm.es/proletool/ (last access on 25/06/2020).
8. URL: http://www2.cs.tum.edu/projects/cup/ (last access on 25/06/2020).
The student module contains information about the students, with the most important being that which refers to what extent they have acquired knowledge and skills in predictive parsing learning. Moreover, it contains statistics about how students have used the tool.

The evaluation module enables the professor to access information about how students are using the system or concerning how they are progressing with their training. This information may be used for planning new learning activities.

The domain module possesses declarative, procedural and conditional knowledge needed to enable active learning as explained below:

1. Declarative knowledge, which refers to grammars or elements of solutions such as first and follow sets of non-terminal symbols, the set of prediction symbols for each production and the parsing tables.
2. Procedural knowledge, which is made up of those algorithms needed for solving the exercises and for correcting the solutions which are proposed by the students (see Table 1).
3. Conditional knowledge, which is used for recommending the next action the students should take on the basis of the knowledge they have and the results obtained in the last exercises carried out.

In addition, examples or case studies have been incorporated into the domain module. This has been done using a repository of examples, which will be used for reinforcing learning. These examples are of great educational value and will be used for guiding the student’s work.

The system designed and developed has been deployed on the web and can be accessed by a web browser to be tested (registration is necessary).9

In the following sections, the interface and tutor modules will be explained in detail. These are the most important and innovative modules, as the interface module is that which gives freedom to students and the tutor module is that which implements the most up-to-date functionalities.

### 3.2. ITS Interface module

The interface module of the ITS proposed herein has been designed on the premise of providing students as much freedom as possible. Therefore, communication between the tool and the student is carried out with a DSL which has been specifically designed for this purpose. Using this language, students may upload the exercises they want to learn how to do as well as the solutions (which may or may not be correct) they have proposed for them to the ITS.

Due to space limitations, details of how the vocabulary, syntax and semantics of the designed language have been established are not provided in this paper. This DSL has been designed in a similar way to [47]. Despite this, these language elements have been specified using the metalanguage Extended BackusNaur Form (EBNF) and can be consulted on ProlesTool 3.0 web (see footer link).

An exercise consists in defining a grammar by specifying the four elements the tuple contains, that is, \((N,T,P,S)\) (see Section 2.2), and indicating the type of analyzer to be built for it (LL1, SLR1, LR1 and/or LALR1). The set of terminal symbols for the grammar \((T)\) can be given by the lexical analyzer which recognizes them (see lines 1–12 and 16 in Listing 1) or by specifying the terminal symbols (e.g. in the sentence terminal int, char, real, id; in line 16 instead of sentence use globalLexer and remove lines 1–12 in Listing 1). The type of analyzer is indicated by the directive analysys LL1, SLR1; (see line 15 in Listing 1).

---

9 Proletool 3.0 is available at https://portal.esi.uclm.es/proletool3/ (last access on 25/06/2020).
The solutions to the exercises the tools solve involves the following steps:

- Identifying nullable symbols.
- Specifying FIRST(X) and FOLLOW(X) sets for all grammar symbols X in N (i.e. ∀X ∈ N).
- For LL1 exercises:
The interface module uses a DSL processor for processing the documents given as input to the system. Meanwhile, it will check if these are well formed (in terms of vocabulary, syntax and semantic restrictions). Subsequently, it will obtain the necessary information from them, that is, the four elements that define the grammar (N, T, P, S), the type of parser the student want to build for it (LL1, SLR1, LR1 or LALR1) and the solution he or she proposes to it (nullable symbols, first and follow sets for each non-terminal symbol, lookahead symbols for each production and parser tables). The latter could be considered as a belief which the student has as to what the solution is for the exercise loaded into the system. All this information will be used in other modules of this ITS.

### Table 1
Procedural knowledge for solving exercises in Proltool 3.0.

| Operation | Description |
|-----------|-------------|
| Nullable(G) | It obtains the non-terminal symbols in the grammar G that can be derived into the empty string (λ). |
| First(x) | It obtains the set of terminal symbols that begin the strings derivable from x. |
| Follow(A) | It obtains the set of terminal symbols that can appear immediately to the right of A in some partial derivation. |
| Checking_LL1.Condition(G) | It checks whether the grammar G is LL(1). |
| Lookahead(p) | It obtains the set of terminal symbols which help decide whether to use the rule p. |
| Parsing_Table_LL1(G) | It constructs the predictive parsing table LL(1) for the grammar G. |
| VPR.SLR1(G) | It constructs the finite-state control of the bottom-up or shift-reduce parser SLR(1) for the grammar G, i.e. the viable prefix recognizer. |
| Parsing_Table_SLR1(G) | It constructs the predictive parsing table SLR(1) for the grammar G. |
| VPR.LR1(G) | It constructs the finite-state control of the bottom-up or shift-reduce parser LR(1) for the grammar G, i.e. the viable prefix recognizer. |
| Parsing_Table_LR1(G) | It constructs the predictive parsing table LR(1) for the grammar G. |
| VPR.LALR1(G) | It constructs the finite-state control of the bottom-up or shift-reduce parser LALR(1) for the grammar G, i.e. the viable prefix recognizer. |
| Parsing_Table_LALR1(G) | It constructs the predictive parsing table LALR(1) for the grammar G. |

- Specifying LOOKAHEAD(pj) for each grammar production rule pj in P (i.e. ∀pj ∈ P).
- Specifying the LL(1) parsing table whose dimensions will be |N| × (|T| + 1), where the rows are labelled with the non-terminal symbols and the columns with the terminal symbols plus the end of string delimiter (‘$’). The row-column intersection will contain the productions applicable in this situation. The analysis table LL(1) for the exercise in Listing 1 is shown in Fig. 3.

- For SLR1, LR1 and LALR1 exercises:

- Specifying the SLR(1), LR(1) or LALR(1) parsing table. The syntax analysis table for the bottom-up or shift-reduce analyzers (SLR(1), LR(1), LALR(1)) is similar in structure although the way these are developed is different. They are obtained from a viable prefix recognizer (see Fig. 4). The analysis table consists of two tables: the action table and the reduce or goto table. The dimensions of the action table are \(|\text{number states of the viable prefix recognizer}| \times (|T| + 1)\), with the rows being the states the viable prefix recognizer are at and the columns being the terminal symbols plus the end of string delimiter (‘$’). Each row-column intersection will contain one of the following actions: (1) shift into state x, \(dx\); (2) reduce by rule \(p_i\), \(ri\); (3) accept input as a string of the language given; or (4) syntax error. The dimensions of the reduce or goto table are \(|\text{number states of the viable prefix recognizer}| \times |N|\), with the rows being the states the viable prefix recognizer are at and the columns being the non-terminal symbols. Each row-column intersection contains the change in state produced in the viable prefix recognizer when there is a reduction during analysis. The analysis table SLR(1) for the exercise in Listing 1 is shown in Fig. 5.

In Listing 2, an example of a correct and complete solution to the problems related to developing bottom-up analyzers (a LL(1) analyzer to be specific) is shown (see Listing 1). This solution is correct and complete and it could be compared with an incorrect and incomplete one shown in Listing 4.

In Listing 3, an example of a correct and complete solution to the problems about developing shift-reduce analyzers (specifically a SLR(1) analyzer) is shown (see Listing 1). There will be similar solutions to the problems associated with the LR(1) and LALR(1) types.

The interface module uses a DSL processor for processing the documents given as input to the system. Meanwhile, it will check if these are well formed (in terms of vocabulary, syntax and semantic restrictions). Subsequently, it will obtain the necessary information from them, that is, the four elements that define the grammar (N, T, P, S), the type of parser the student want to build for it (LL1, SLR1, LR1 or LALR1) and the solution he or she proposes to it (nullable symbols, first and follow sets for each non-terminal symbol, lookahead symbols for each production and parser tables). The latter could be considered as a belief which the student has as to what the solution is for the exercise loaded into the system. All this information will be used in other modules of this ITS.
The traditional model has been followed for designing and developing the DSL processor which is made up of four phases:

Phase 1. Lexical analysis: this will be responsible for analyzing the sequence of characters there are in the input and grouping them into elements belonging to language vocabulary (i.e. tokens).

Phase 2. Syntactic analysis: the purpose of this phase will be to check that the sequence of tokens obtained in the lexical phase has a valid structure according to the language syntax.

Phase 3. Semantic checking: this will be entrusted with checking the semantic restrictions on the syntactically well-formed structures and starting the information extraction process.

Phase 4. Code generation: in this phase information will be changed into declarative knowledge which will make up part of the domain module.

The details of how the DSL processor was developed will not be given as this has been deemed to be beyond the scope of this paper.

The interface module also has mechanisms for enabling interactions with the solutions developed in order to streamline navigation of the generated solution, the solutions corrections, feedback and suggestions made by the system. Moreover, this module enables visual and interactive simulations of strings analysis to be carried out using the parsers developed (see Fig. 6). With this students can experience and understand how the types of parsers studied work.
3.3. Tutor module

3.3.1. Fundamental ideas

Just as stated earlier, the purpose of this module is to encourage students to actively and self-sufficiently learn predictive parsing techniques. Intended learning outcomes (ILOs) for the ITS proposed herein are as follows:

- \( o_1 \): To know how to determine the \textit{NULLABLE} symbols.
- \( o_2 \): To know how to calculate the \textit{FIRST} sets.
- \( o_3 \): To know how to calculate the \textit{FOLLOW} sets.
- \( o_4 \): To know how to calculate the \textit{LOOKAHEAD} sets.
- \( o_5 \): To know how to develop the top-down analysis \textit{LL}(1).
- \( o_6 \): To know how to develop the bottom-up analysis \textit{SLR}(1), \textit{LR}(1), \textit{LALR}(1).

The main purpose of the ITS proposed in this paper is to get students to engage in learning activities conducive to achieving these ILOs. When deciding on the best learning activities in our situation, it should be remembered that what the students do is actually more important than what the professor does \[49\]. Thus, the teaching/learning activities (TALs) used by the ITS proposed here will be as follows:

- Exampling: students are asked to suggest and do an exercise which brings out a specific concept, problem or situation that they should explore in depth.
- Individual exercises: students are asked to do an exercise from the repository, the purpose of which is to study a specific concept, problem or situation in depth.
- Active review: students are asked to review a specific concept or algorithm in order to identify what they do not understand.

In order to know to what extent students understand ILOs, assessment tasks (ATs) that will be used in the ITS must be set. Scoring rubrics streamline assessment of students’ work. Moreover they are teaching tools that support student learning and promote the development of sophisticated thinking skills. Instructional rubrics that the ITS proposed here uses for grading and evaluating the students’ work (Tables 2–4), as well as the errors they can make before achieving each ILO, will be outlined below (Tables 5–10). This is public information and helps them focus their efforts and self-assess their own work. Furthermore, rubrics and errors provide students with specific feedback enabling them to reflect on how they are performing with a view to improving.

The raw rubric score is calculated as the sum of all criteria grades. The final grade is calculated by comparing actual scores with the worst/best possible score.

---

\[ \]

**Fig. 5.** Exercise1 SLR(1) parsing table (see Listing 1).
Table 2  
Rubric for NULLABLE determination criterion. 

| Assessment | Description |
|------------|-------------|
| 0          | The student does not try to calculate the nullable elements. |
| 1          | The student correctly calculates less than 10% of the nullable elements. |
| 2          | The student correctly calculates between 10% and 40% of the nullable elements. |
| 3          | The student correctly calculates between 40% and 80% of the nullable elements. |
| 4          | The student correctly calculates more than 80% of the nullable elements, but not all of them. |
| 5          | The student correctly calculates all the nullable elements. |

Table 3  
Rubric for FIRST, FOLLOW and LOOKAHEAD sets calculation criteria. 

| Assessment | Description |
|------------|-------------|
| 0          | The student does not try to calculate the elements of the set. |
| 1          | The student correctly calculates less than 10% of the elements of the set. |
| 2          | The student correctly calculates between 10% and 40% of the elements of the set. |
| 3          | The student correctly calculates between 40% and 80% of the elements of the set. |
| 4          | The student correctly calculates more than 80% of the elements of the set, but not the full set. |
| 5          | The student correctly calculates all the elements of the set. |

Fig. 6. Graphical representation and interaction capacity of the proposed ITS.
This module consists of three components which will provide students with the necessary guidance, support and resources for them to succeed, which are as follows (see Fig. 2): (i) Problem solver, (ii) Exercises corrector, and (iii) Learning instructor. These are described in greater detail below.
Errors when developing the SLR(1), LR(1) or LALR(1) tables.

Table 8

| Error ID | Description |
|----------|-------------|
| e_{15}   | Does not include a terminal symbol α in the LOOKAHEAD(p_i) set, when the right-hand side of the production rule p_i begins with this terminal symbol (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha x \) such that \( \alpha \in (N \cup T)^* \) and \( x \in T \)). |
| e_{16}   | Does not include a terminal symbol α in the LOOKAHEAD(p_i) set, when the right-hand side of the production rule p_i begins with a non-terminal symbol B and α belongs to the FIRST(B) set, (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow Bx \) such that \( \alpha \in (N \cup T)^* \) and \( B \in N \) and \( x \in FIRST(B) \)). |
| e_{17}   | Does not include a terminal symbol α in the LOOKAHEAD(p_i) set, when the right-hand side of the production rule p_i begins with a list of non-terminal and nullable symbols, \( \beta \), and the symbol α is on its right, (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha \beta x \) such that \( \alpha \in (N \cup T)^* \) and \( \beta \in N^* \) and \( \beta \Rightarrow \lambda \)). |
| e_{18}   | Does not include a terminal symbol α in the LOOKAHEAD(p_i) set, when the right-hand side of the production rule p_i has a non-terminal symbol C with all symbols on its left nullable and α belongs to C initials, (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha C \beta x \) such that \( \alpha \in (N \cup T)^* \) and \( \beta \in N^* \) and \( C \in N \) and \( \beta \Rightarrow \lambda \) and \( x \in FIRST(C) \)). |
| e_{19}   | Does not include a terminal symbol α in the LOOKAHEAD(p_i) set, when the right-hand side of the production rule p_i is nullable and the terminal symbol α belongs to FOLLOW(X) (i.e. \( \exists \pi \in P \), with \( p_i : X \rightarrow \alpha x \) such that \( \alpha \in (N \cup T)^* \) and \( x \Rightarrow \lambda \) and \( x \in FOLLOW(X) \)). |
| e_{20}   | Includes terminal symbols which do not belong to LOOKAHEAD(p_i). |

Table 9

| Error ID | Description |
|----------|-------------|
| e_{21}   | Wrongly fills the table when working with productions p_i whose right-hand side begins with a terminal symbol \( \alpha \) (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha x \) such that \( \alpha \in (N \cup T)^* \) and \( x \in T \)) and does not make \( LL1_{Table}[A, \alpha] = \{p_i\} \). |
| e_{22}   | Wrongly fills the table when working with productions p_i whose right-hand side begins with a non-terminal symbol B (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow Bx \) such that \( \alpha \in (N \cup T)^* \) and \( B \in N \) and does not make \( LL1_{Table}[A, B] = \{p_i\} \) \( \forall \alpha \in FIRST(B) \)). |
| e_{23}   | Wrongly fills the table when working with productions p_i whose right-hand side begins with a list of non-terminal and nullable symbols, \( \beta \), followed by a terminal symbol α (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha \beta x \) such that \( \alpha \in (N \cup T)^* \) and \( \beta \in N^* \) and \( \beta \Rightarrow \lambda \)) and does not make \( LL1_{Table}[A, \beta] = \{p_i\} \) \( \forall \alpha \in FIRST(C) \)). |
| e_{24}   | Wrongly fills the table when working with productions p_i where on its right-hand side there is a non-terminal symbol C and all the symbols on its left are nullable (i.e. \( \exists \pi \in P \), with \( p_i : A \rightarrow \alpha C \beta \) such that \( \alpha \in (N \cup T)^* \) and \( \beta \in N^* \) and \( C \in N \) and \( \beta \Rightarrow \lambda \)) and does not make \( LL1_{Table}[A, C] = \{p_i\} \) \( \forall \alpha \in FIRST(C) \)). |
| e_{25}   | Wrongly fills the table when working with productions p_i whose right-hand side is nullable (i.e. \( \exists \pi \in P \), with \( p_i : X \rightarrow \beta \) such that \( \beta \in N^* \) and \( \beta \Rightarrow \lambda \)) and does not make \( LL1_{Table}[A, \beta] = \{p_i\} \) \( \forall \alpha \in FOLLOW(X) \)). |
| e_{26}   | Fills the cells which must be empty or fills them wrongly (with wrong productions). |

Table 10

| Error ID | Description |
|----------|-------------|
| e_{27}   | Wrongly builds the tables, with a wrong number of rows in the action table and reduce or goto table (i.e. incorrect number of states for the viable prefix recognizer). |
| e_{28}   | Does not correctly indicate the symbols that produce actions in the action table (i.e. wrong number of columns in the action table). |
| e_{29}   | Does not correctly handle the end of string delimiter '$' in the action table (i.e. there are no actions associated with it when there should be). |
| e_{30}   | Does not correctly calculate the first state of the viable prefix recognizer (i.e. errors in the first row of the action and reduce or goto table). |
| e_{31}   | Wrongly fills the actions in the action table. |
| e_{32}   | Does not indicate shifts of state in the action table. |
| e_{33}   | Does not indicate a reduction in the action table. |
| e_{34}   | Does not indicate the situation in which the analysis must successfully end. |
| e_{35}   | Does not correctly indicate in the reduce or goto table the symbols that produce a change of state (i.e. wrong number of columns in the reduce or goto table). |
| e_{36}   | Wrongly fills the change of state in the reduce or goto table. |

3.3.2. Problem solver component

The problem solver component just solves exercises uploaded to the system, and for this purpose, it uses declarative knowledge acquired by the interface module and procedural knowledge available in the domain module (see Section 3.1). Furthermore, it will show how to reach the solution and, in doing so, it must perform the following actions:

- It first understands the exercise, converting the information obtained by the interface module (i.e. four elements that define the grammar \((N, T, P, S)\) and the type of parser that students wants to build for it \((LL1, SLR1, LR1\) or \(LALR1))\) in declarative knowledge.
It generates the solution applying the appropriate procedural knowledge to the declarative knowledge to build the desired parser (LL1, SLR1, LR1 or LALR1). Moreover, it will provide all information needed for understanding how the solution is reached.

It accepts the solution given as valid, storing each element of the solution (nullable symbols, firsts and follows of each non-terminal symbol, lookahead symbols of each production and parser tables) in the domain module as declarative knowledge.

As a result of the work done by this module, students will know what the solution to an exercise is and how to reach it. However, they will not know what they have done right or where they have gone wrong in the solution they have given. So, they must check manually if the solution provided by the tool matches that they have obtained beforehand.

3.3.3. Exercises corrector component

The main goal of the exercises corrector component is to compare the solution obtained by the problem solver component with that the student has provided for the exercise given as input to the ITS. That is, this component must act as a professor in its capacity as exercises corrector, its mission is to detect the correct answers as well as where students may go wrong. It must also inform of any information which students have not specified in the solution that they set out. To do so, it carries out the following actions:

- It corrects the solutions students have given for the exercise, comparing the declarative knowledge containing the solution obtained by the problem solver component with that which contains the solution given by students. This is carried out using procedural knowledge that has the necessary mechanisms to do such comparison. Additionally, it detects whether there are errors in the student exercise solution and if so, which ones.
- It evaluates the solution students provide for the exercise, using the aforementioned rubrics score (see Tables 2–4). The score for each ILO will be stored in a variable called recent_scores whose value ranges from 0 to 5, inclusive. Moreover, they will be used to modify the extent to which students have mastered each ILO. It will be stored in a variable called level_mastery, also defined in the range [0, 5].
- It gives feedback, informing whether the solution students have given for the exercise contains errors (see Tables 5–10). This feedback also provides a justification for the grade given.
- It determines how student learning is progressing, informing to what extent the ILOs are being reached.

The exercises corrector component creates a report with the results of the correction process as output. This report contains information about what students have done well, errors they have made and that data which although they have not been specified in the input, the solution should contain them. In Tables 11 and 12 there is an example of the report created for the input shown in Listing 4. This is provided as a solution to exercise1 shown in Listing 1, in which errors are made.

The exercises corrector component uses fuzzy rules to determine how much students have learnt for each ILO. With these rules, it can be known whether an ILO has been reached or not. The rules use information about the mastery levels of each student in the ILO (level_mastery) and the errors/correct answers given in relation to the ILO in the last exercise completed (recent_scores). Each ILO will be associated with an achievement indicator (achieved, in progress or not achieved), which determines if the student know how to do it (achieved), do not know how to do it (not achieved) or if he or she is at an intermediate phase (in progress).

The set of fuzzy rules used by the component are as follows:

- $R_1$: IF recent_scores are Very Bad THEN achievement_level is not achieved.
- $R_2$: IF recent_scores are Bad AND level_mastery is NOT Very Good THEN achievement_level is not achieved.
- $R_3$: IF recent_scores are Bad AND level_mastery is Very good THEN achievement_level is in progress.
- $R_4$: IF recent_scores are Average AND level_mastery is NOT Very Bad THEN achievement_level is in progress.
- $R_5$: IF recent_scores are Average AND level_mastery is Very Bad THEN achievement_level is not achieved.
- $R_6$: IF recent_scores are Good OR Very Good AND level_mastery is Average OR Bad OR Very Bad THEN achievement_level is in progress.
- $R_7$: IF recent_scores are Good OR Very Good AND level_mastery is Good OR Very Good THEN achievement_level is achieved.

The fuzzy definition domains for the input variables recent_scores, level_mastery and for the output variable achievement_level are not shown, as the authors of this paper do not deem this to be important in order to understand the system presented herein.

However, we do think that justifying the use of fuzzy logic when designing this module would be beneficial. At this stage, there is no attempt to provide an accurate assessment of how much students have learnt, but rather to provide them with guidance that they can understand. In this respect, using linguistic variables makes it easier for them to interpret this information.
As a result of the work done by this module, students will receive detailed feedback on the work they have done, with an explanation of what areas they have done well, and what they may still need to improve. In addition, on a broader level, students will be informed as to whether they have truly mastered the subject or not.

3.3.4. Learning instructor component

The aim of the learning instructor component is to guide the student's progress towards a goal, which in our situation is reaching our ILOs (see Section 3.3.1). Now, it is time to guide the work of the students by assigning new exercises (individual exercises), by making suggestions about what study materials they should consult before they can move on to the next assignment (active review) or by asking them to reflect on what they have learnt which enables them to recognize errors and develop strategies for tackling weak points themselves (exampling). Guiding students is one of the most important sides of the work of a professor. For this purpose, the system proceeds as follows:

- It decides what the next step should be for reaching the ILOs. Suggestions must be as specific as possible.

The learning instructor component uses the following resources to work: a repository of examples, each one related to situations that could lead to the errors discussed in Tables 5–10; study materials including video lectures, course textbooks and lecture slides; and a set of rules for deciding on what to recommend students to do.

Let us see how the rules have been set out. We have defined three categories of rules, which are the following:

- Type 1: IF achievement_level in ILO o<sub>i</sub> is achieved THEN Challenge the student to propose a new exercise that works on o<sub>i</sub>.
- Type 2: IF achievement_level in ILO o<sub>i</sub> is in progress AND the error made is e<sub>j</sub> THEN Assign the student an exercise from the repository in which the context situation related to error e<sub>j</sub> is produced.
- Type 3: IF achievement_level in ILO o<sub>i</sub> is not achieved AND the error made is e<sub>j</sub> THEN Review concept or algorithm X AND Recommend an exercise from the repository be done in which the situation related to error e<sub>j</sub> is produced.

As a result of the work done by this module, students will receive detailed feedback on the work they have done, with an explanation of what areas they have done well, and what they may still need to improve. In addition, on a broader level, students will be informed as to whether they have truly mastered the subject or not.

Table 11
Report of the correction of NULLABLE, FIRST, FOLLOW and LOOKAHEAD.

| Block       | Item | Success | Failure | Not specifically indicated |
|-------------|------|---------|---------|---------------------------|
| NULLABLE    | R    |         |         |                           |
|             | L    |         |         |                           |
| FIRST       | S    | int, real, char | id | int, real, char |
|             | T    | int, real, char | | |
|             | L    | id | | |
|             | R    | | | |
| FOLLOW      | S    | | | $ |
|             | T    | id | | |
|             | L    | | | |
|             | R    | | | |
| LOOKAHEAD   | p<sub>1</sub> | int, real, char | | |
|             | p<sub>2</sub> | | |
|             | p<sub>3</sub> | char | | |
|             | p<sub>4</sub> | real | | |
|             | p<sub>5</sub> | id | | |
|             | p<sub>6</sub> | | | |
|             | p<sub>7</sub> | id | | |

Table 12
Report of the correction of table LL(1) definition, where [p<sub>i</sub>], using brackets, denotes Not specifically indicated and [p<sub>i</sub>], using crossing-outs, denotes an error.

| Block | Item | Success | Failure | Not specifically indicated |
|-------|------|---------|---------|---------------------------|
| S     | | $ \rightarrow TL : | | |
| T     | | $ \rightarrow TL : | | |
| L     | | L \rightarrow idR | | |

As a result of the work done by this module, students will receive detailed feedback on the work they have done, with an explanation of what areas they have done well, and what they may still need to improve. In addition, on a broader level, students will be informed as to whether they have truly mastered the subject or not.
connected with it. Here, the learning instructor component assumes students have the necessary theoretical knowledge and practical skills, but must keep on practicing exercises. For this reason, it recommends they do some exercises from the repository in which the situation whereby the error was made is recreated. Type 3 rules (36 rules, one for every possible error) are used when students have not reached a high enough achievement level for a learning goal and also make errors linked to this. Here, the learning instructor component assumes students do not have the necessary theoretical knowledge and practical skills. Therefore, it recommends they study these as well as doing exercises from the repository which recreates the situation in which they went wrong.

The rule base consists of 78 rules which depend on the achievement level of each ILO and the errors discussed in Section 3.3.1. These may be grouped in the following way: (i) ILO o1: 7 rules (R1 – R7); (ii) ILO o2: 11 rules (R8 – R28); (iii) ILO o3: 13 rules (R29 – R41); (iv) ILO o4: 17 rules (R42 – R58); (v) ILO o5: 13 rules (R59 – R71); (vi) ILO o6: 21 rules (R72 – R93). Full details of all the rules used by the learning instructor component will not be given, but, by way of example, just a few rules belonging to each of the groupings and types specified above will be shown:

- (Type 3) R2: IF achievement_level in ILO o1 is not achieved AND the error made is e2 THEN review concept of nullable symbol AND do exercise select_exercise(e2).
- (Type 2) R2: IF achievement_level in ILO o1 is in progress AND the error made is e3 THEN do exercise select_exercise(e3).
- (Type 2) R3: IF achievement_level in ILO o2 is in progress AND the error made is e4 THEN do exercise select_exercise(e4).
- (Type 3) R4: IF achievement_level in ILO o3 is not achieved AND the error made is e11 THEN review concept of follows AND do exercise select_exercise(e11).
- (Type 2) R5: IF achievement_level in ILO o4 is in progress AND the error made is e19 THEN do exercise select_exercise(e19).
- (Type 1) R6: IF achievement_level in ILO o5 is achieved THEN propose a LL(1) table development exercise.
- (Type 3) R5: IF achievement_level in ILO o6 is not achieved AND the error made is e27 THEN review the structure of the bottom-up analysis table AND do exercise select_exercise(e27).

where select_exercise(e_i) takes an exercise from the repository in which there are the same circumstances whereby the error e_i occurred (see Tables 5–10). For R2 and R4, this means choosing an exercise which contains a NULLABLE symbol A and \( \beta p_i \) with \( p_i : A \rightarrow \beta \). For R5, this means recommending an exercise from the repository which contains the production \( p_i \in P \), with \( p_i : A \rightarrow \beta \), such that \( \beta \in N^* \) and \( \alpha \in (N \cup T)^* \) and \( C \in N \) and \( \beta \Rightarrow \lambda \) and \( a \in \text{FIRST}(C) \). For R21, this means selecting an exercise from the repository which contains the production \( p_i \in P \), with \( p_i : X \rightarrow \alpha \beta \), such that \( \alpha \in (N \cup T)^* \) and \( \beta \Rightarrow \lambda \). For R27, it selects an exercise from the repository which contains the production \( p_i \in P \), with \( p_i : X \rightarrow \lambda \), such that \( \alpha \in (N \cup T)^* \) and \( \beta \Rightarrow \lambda \) and \( a \in \text{FOLLOW}(X) \). For R58, it selects an exercise from the repository for developing bottom-up SLR(1) syntax analyzers.

To round off this section, it is worth mentioning that each exercise in the repository is related to errors or rather to situations that can lead to such errors. This association is automatically made every time the professor uploads an exercise to the repository. The DSL processor will analyze it, extract any important information there is and associate it with types of errors. It must be stressed that every example may be associated with various error situations.

After this module has completed its work, students will know what to do next to keep on learning.

Lastly, it must be noted that instructor learning component just makes recommendations, which students are free to accept or decline. In any event, a record is made of the exercises recommended and done from the repository to prevent duplicating recommendations to the student.

4. Initial assessment of Proletool 3.0

In order to evaluate the proposed system, Proletool 3.0 has been used in a real situation by fourth year Computer Science Engineering students to learn concepts and techniques related to predictive parsing from the Language Processors course at the Computer Science and Engineering Faculty of the University of Castilla-La Mancha (Spain). In this section, we are going to introduce some statistics of use of the tool, analyzing its influence on the learning process and the grade obtained by the students. On the other hand, the results of an empirical study in which we collect the opinion the students had concerning how useful and effective the suggested system was are presented.

In total, 68 students had the opportunity to use the tool Proletool 3.0 to study the subject (i.e. the number of students enrolled in the course). The tool was freely used only by students who considered it useful to study the subject. In other words, no experimentation was designed (not an experiment itself). Moreover, its use was neither rewarded nor penalized. Therefore, the free use of the tool might already be considered as an indicator of its perceived usefulness.

The tool was regularly used by 35 students (those who did more than 20 exercises with it), occasionally by 6 students (who did less than 15 exercises) and 27 students never used it (among whom were the 5 students who did not attend classes). Out of the group that used the tool regularly, 51% used the new functions that Proletool 3.0 offered (18 students), as opposed to 49% who did not (17 students). In total, the students completed 1.210 exercises using the tool.

The usefulness and effectiveness of the tool was evaluated by comparing the exam grades of students who used the tool as opposed to those who did not (see Table 13). The average grade obtained by the students was 5.87 points out of 10, while
Table 13
Use of the tool and academic assessment, where $S_j$ is the student’s identifier.

| Student | Exercises | Type          | Final grade |
|---------|-----------|---------------|-------------|
| $S_{11}$ | 30        | Proletool 3.0 | 10.0        |
| $S_{38}$ | 29        | Proletool 3.0 | 10.0        |
| $S_{44}$ | 28        | Proletool 3.0 | 9.7         |
| $S_{11}$ | 35        | Proletool 3.0 | 9.5         |
| $S_{37}$ | 31        | Proletool 3.0 | 9.5         |
| $S_{48}$ | 20        | Proletool 3.0 | 9.2         |
| $S_{32}$ | 30        | Proletool 3.0 | 9.0         |
| $S_{17}$ | 20        | Proletool 3.0 | 8.8         |
| $S_{24}$ | 32        | Proletool 3.0 | 8.5         |
| $S_{12}$ | 33        | Proletool 3.0 | 8.0         |
| $S_{17}$ | 20        | Proletool 3.0 | 7.8         |
| $S_{12}$ | 45        | Proletool 2.0 | 7.6         |
| $S_{14}$ | 30        | Proletool 3.0 | 7.5         |
| $S_{21}$ | 28        | Proletool 3.0 | 7.5         |
| $S_{23}$ | 25        | Proletool 3.0 | 7.5         |
| $S_{25}$ | 35        | Proletool 3.0 | 7.5         |
| $S_{47}$ | 20        | Proletool 3.0 | 7.5         |
| $S_{40}$ | 37        | Proletool 3.0 | 7.5         |
| $S_{13}$ | 30        | Proletool 2.0 | 7.3         |
| $S_{30}$ | 45        | Proletool 2.0 | 7.3         |
| $S_{12}$ | 40        | Proletool 2.0 | 7.3         |
| $S_{28}$ | 35        | Proletool 2.0 | 7.3         |
| $S_{32}$ | 25        | Proletool 2.0 | 7.5         |
| $S_{18}$ | 30        | Proletool 2.0 | 6.5         |
| $S_{26}$ | 0         | Proletool 2.0 | 6.5         |
| $S_{36}$ | 45        | Proletool 2.0 | 5.8         |
| $S_{35}$ | 20        | Proletool 2.0 | 5.0         |
| $S_{37}$ | 43        | Proletool 2.0 | 5.0         |
| $S_{19}$ | 0         | Proletool 2.0 | 5.0         |
| $S_{36}$ | 45        | Proletool 2.0 | 5.8         |
| $S_{35}$ | 20        | Proletool 2.0 | 5.0         |
| $S_{17}$ | 43        | Proletool 2.0 | 5.0         |
| $S_{15}$ | 0         | Proletool 2.0 | 5.0         |
| $S_{36}$ | 35        | Proletool 2.0 | 5.0         |
| $S_{46}$ | 11        | Proletool 2.0 | 5.0         |
| $S_{44}$ | 0         | Proletool 2.0 | 5.0         |
| $S_{36}$ | 11        | Proletool 2.0 | 5.0         |
| $S_{27}$ | 0         | Proletool 2.0 | 5.0         |
| $S_{31}$ | 36        | Proletool 2.0 | 5.0         |
| $S_{49}$ | 35        | Proletool 2.0 | 5.0         |
| $S_{33}$ | 0         | Proletool 2.0 | 4.5         |
| $S_{38}$ | 0         | Proletool 2.0 | 4.5         |
| $S_{30}$ | 0         | Proletool 2.0 | 4.5         |
| $S_{32}$ | 0         | Proletool 2.0 | 4.5         |
| $S_{21}$ | 0         | Proletool 2.0 | 4.5         |
| $S_{15}$ | 12        | Proletool 2.0 | 4.0         |
| $S_{12}$ | 0         | Proletool 2.0 | 4.0         |
| $S_{15}$ | 10        | Proletool 2.0 | 4.0         |
| $S_{18}$ | 13        | Proletool 2.0 | 4.0         |
| $S_{21}$ | 8         | Proletool 2.0 | 4.0         |
| $S_{39}$ | 0         | Proletool 2.0 | 4.0         |
| $S_{43}$ | 0         | Proletool 2.0 | 4.0         |
| $S_{45}$ | 0         | Proletool 2.0 | 3.9         |
| $S_{41}$ | 0         | Proletool 2.0 | 3.7         |
| $S_{16}$ | 0         | Proletool 2.0 | 3.0         |
| $S_{23}$ | 0         | Proletool 2.0 | 3.0         |
| $S_{14}$ | 0         | Proletool 2.0 | 3.0         |
| $S_{34}$ | 0         | Proletool 2.0 | 2.9         |
| $S_{39}$ | 0         | Proletool 2.0 | 2.4         |
| $S_{10}$ | 0         | Proletool 2.0 | 2.0         |
| $S_{42}$ | 0         | Proletool 2.0 | 1.9         |
| $S_{41}$ | 0         | Proletool 2.0 | 1.7         |
| $S_{14}$ | 0         | Proletool 2.0 | 1.7         |
| $S_{29}$ | 0         | Proletool 2.0 | 1.7         |
| $S_{24}$ | 0         | Proletool 2.0 | 1.7         |
| $S_{33}$ | 0         | Proletool 2.0 | 1.7         |
| $S_{41}$ | 0         | Proletool 2.0 | 1.7         |
the percentage of students who passed the subject was 61.76% (i.e. 42 students). Analyzing the obtained academic results in relation to using the tool, we checked that 100% of the students who regularly used it passed the subject and obtained an average grade of 7.37 points out of 10, whereas only 10.29% who used it occasionally, or never, passed, and they obtained an average grade of 5.5 points out of 10. There were significant differences between the academic results of those that used it just with the Proletool 2.0 functions and those who used the new Proletool 3.0 ones, the former obtained an average grade of 6.22 compared to 8.44 for the latter. Moreover, the top ten grades were obtained by students that used Proletool 3.0. These two groups also differ when the number of exercises carried out by each one of them is taken into account. Students who used the new functions that came with Proletool 3.0 did fewer exercises, 508 in total. On average every student from this group did 28 exercises. Those who did not use them performed 637 exercises in total, which means, on average, that 37 exercises were done per student. This may show that their learning was more guided by the learning instructor component, since fewer exercises were needed to achieve better grades.

In the light of these data, it could be stated that the tool has influenced the academic results of the students. However, one may argue that because using it was voluntary, only the best students used it. Even if that were true, the fact that the tool had been used by this type of student is a good sign as they may have considered it useful for studying this subject or that it was highly motivating for that purpose.

In order to confirm this hypothesis, we asked the students their opinion concerning Proletool 3.0. This was done by means of an online survey in which questions were asked on a range of issues (see Table 14). The survey, in which a total of 25 students took part out of the 35 who used the tool, was anonymous and voluntary. They had to answer dichotomous questions about the tool features. We use this kind of questions to elicit a choice from respondents on specific issues. They are used for unambiguously distinguishing the opinion of each respondent. Table 14 shows the questions and the answers respondents gave to them.

From the values obtained in this survey, some conclusions can be drawn: the tool is user-friendly (Q1), the learning curve for the tool is quite short (Q2), the information it provides is understandable and its simulation capacity helps understand the concepts (Q3 and Q4). The responses to questions Q5, Q6, Q7 and Q8 show the students found the tool useful for learning this subject. The response to question Q9 shows students did not find the tool motivating.

5. Conclusions and future work

In this paper, an Intelligent Tutoring System has been designed and developed to enable student learning through experimentation, working at the highest cognitive level of Bloom’s Taxonomy, encouraging them to learn actively and to be self-sufficient. As such, the tool developed, called Proletool 3.0, contains the modules typically found in this type of systems (Domain module, Tutor module, Student’s module, Interface module and the Evaluation module). In addition, the system provides a web interface that enables it to be used as a learning object on an e-learning platform.

To allow the learning through experimentation, the tool enables students to create new exercises as well as give solutions to them on their own, to obtain instant evaluations and corrections automatically, and to provide suggestions that may guide them through their learning process. The freedom to work with any exercise the student can think of is the main novelty of the system suggested in this article as current systems are very limited in this respect. These only allow the student to work with the examples included in a repository or variants of these.

A specific domain language has been designed to create exercises and specify their solutions, and its associated processor has been developed to process them. The DSL has been designed with a high focus on syntactic and semantic specifications and restrictions. Its processor has been implemented according to the traditional model, in which the input text is processed, through four sequential phases (lexical analysis, syntactic analysis, semantic checking and code generation), and its meaning is processed to produce declarative knowledge.

10 Proletool 3.0 is free and available at http://portal.esi.uclm.es/proletool3/.

Table 14
Questions and results from the survey carried out by the students.

| ID | Question                                                                 | Yes % | No % |
|----|-------------------------------------------------------------------------|-------|------|
| Q1 | Have you found it difficult to use Proletool 3.0?                        |       | 100  |
| Q2 | Have you found it difficult to learn the Proletool 3.0 language?         | 8     | 92   |
| Q3 | Do you think the information provided by Proletool 3.0 is understandable?| 92    | 8    |
| Q4 | Has the visual and interactive simulations of Proletool 3.0 helped you to understand the syntax analysis methods? | 88    | 12   |
| Q5 | Has Proletool 3.0 been useful for understanding the syntax analysis methods? | 100  |      |
| Q6 | Do you think using Proletool 3.0 influences the obtained grade?         | 88    | 12   |
| Q7 | If you could go back in time, would you use Proletool 3.0 to prepare the subject? | 100  |      |
| Q8 | Would you recommend using Proletool 3.0 to other students in the next academic year? | 100  |      |
| Q9 | Has Proletool 3.0 motivated you to study more?                           | 32    | 68   |
In addition, the developed system contains the components that allow to solve the exercises proposed by the student, to correct the solutions that he/she presents and to guide his/her learning. The design of these components is a complex task due to the freedom given to the student to define their own exercises. For any exercise given as input, the exercises corrector component compare the solutions students suggest with those obtained by the problem solver component. Knowledge about the subject and potential errors that may be made has been modeled and included in the system, in order to provide accurate information about (i) the correction results (success, failure and not specifically indicated) and (ii) ILOs achievement level (to know which have been achieved, which have not been achieved and which are in progress).

The learning instructor component is the responsible for giving students recommendations in order to guide their learning, so that the ILOs will be reached. To carry out this task, a knowledge base is used, in which the different achievement indicators for each of the learning goals are linked to each of the errors that may be made in the subject in order to recommend the next action to be taken. This may be (i) challenging the student to propose an exercise (type 1), (ii) suggesting doing an exercise from the repository (type 2), or (iii) recommending studying a concept or technique (type 3).

The system put forward in this paper has been used on a real learning situation. It has been used by the students during four academic years in a Language Processors course at the Computer Science and Engineering Faculty of the University of Castilla-La Mancha (Spain). Thus, an initial evaluation of the usefulness and effectiveness of the system has been made. The data obtained show how promising the system is as a useful learning instrument for student training, since it has had influence on the marks obtained, and it has been perceived by them as a good learning instrument.

As a future line of research a component has been planned to be incorporated into the tutor module which can assess the knowledge students have gained. The aim of this is to detect those who are at risk of dropping out and to set up activities which will encourage to join the group again. There are also plans to integrate this system into an e-learning platform, where it will coexist with a range of materials and activities: lessons, readings, pictures, audios, videos. in order to study what influence it has.

Listing 1: Grammar definition with lexical analyzer specification

```plaintext
lexer globalLexer
{
    tokens
    {
        int  := "int";
        real := "real";
        char := "char";
        id   := [a-zA-Z][a-zA-Z0-9]*;
        nulls := [ \r\n\t ];
    }
    pass nulls;
}
grammar exercise1
{
    analysis LL1, SLR1;
    use globalLexer;
    nonterminal S, T, L, R;
    S := T L ';';
    T := int | real | char;
    L := id R;
    R := ';', id R | ;
}
```
Listing 2: Solution of type LL1 for the grammar given in Listing 1

```plaintext
solution exercise1 {
  nullables{R;}
  firsts {
    S={int, real, char};
    T={int, real, char};
    L={id};
    R={',','};
  }
  followers {
    S={$};
    T={id};
    L={';','};
    R={';','};
  }
  lookahead {
    1={int, real, char};
    2={int};
    3={char};
    4={real};
    5={id};
    6={'','};
    7={';','};
  }
  lll_parsing_table {
    [S, int]=1;
    [S, real]=1;
    [S, char]=1;
    [T, int]=2;
    [T, char]=3;
    [T, real]=4;
    [L, id]=5;
    [R, '']==6;
    [R, ';', ']=7;
  }
}
```
Listing 3: Solution of type SLR1 for the grammar given in Listing 1

```plaintext
solution exercise1 {
  nullables{R;}
  firsts{
    S={int, real, char};
    T={int, real, char};
    L={id};
    R={',','};
  }
  followers{
    S={$};
    T={id};
    L={';','};
    R={';','};
  }
  slr1_parsing_table{
    action_table{
      [0,char]=d4;
      [0,int]=d3;
      [0,real]=d5;
      [1,$]=acceptar;
      [2,id]=d7;
      [3,id]=r2;
      [4,id]=r3;
      [5,id]=r4;
      [6,';','']=d8;
      [7,';','']=d10;
      [7,';','']=r7;
      [8,$]=r1;
      [9,';','']=r5;
      [10,id]=d11;
      [11,',','']=d10;
      [11,';','']=r7;
      [12,';','']=r6;
    }
    goto_table{
      [0,S]=1;
      [0,T]=2;
      [2,L]=6;
      [7,R]=9;
      [11,R]=12;
    }
  }
}
```
Listing 4: Solution of type LL1 with errors for the grammar given in Listing 1

```plaintext
solution exercise1 {
    nullables{R,L;}
    firsts{
        S={id};
        T={int, real, char};
        L={id};
        R={' ',','};
    }
    followers{
        T={id};
        L={' ',','};
        R={id};
    }
    lookaheads{
        1={id};
        2={int};
        3={char};
        4={real};
        5={id};
        6={' ',','};
        7={id};
    }
    ll1_parsing_table {
        [T, int] = 2;
        [T, char] = 3;
        [T, real] = 4;
        [L, id] = 5;
        [R, ' ', '] = 6;
        [R, id] = 7;
    }
}
```

CRediT authorship contribution statement

J.J. Castro-Schez: Conceptualization, Methodology, Writing - original draft, Validation, Supervision. C. Glez-Morcillo: Data curation, Writing - original draft, Visualization. J. Albusac: Software, Validation, Writing - review & editing. D. Vallejo: Methodology, Software, Validation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Funding: This research has been supported by the Spanish Ministry of Economy and Competitiveness within the context of the TIN2015-66731-C2-2-R project, by the Department of Technologies and Information Systems (Grant No. 00421372), and by the University of Castilla-La Mancha (AIR Research Group, Grant No. 01110G9180)

References

[1] C. Bonwell, J. Eison, Active Learning: Creating Excitement in the Classroom, ASHE Higher Education Report Series, Wiley, 1991.
[2] T. Murray, Authoring intelligent tutoring systems: an analysis of the state of the art, Int. J. Artif. Intell. Educ. 10 (1999) 98–129.
[44] D. Rodríguez-Cerezo, M. Gómez-Albarrán, J.-L. Sierra-Rodríguez, Interactive educational simulations for promoting the comprehension of basic compiler construction concepts, in: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE ’13, ACM, New York, NY, USA, 2013, pp. 28–33, https://doi.org/10.1145/2462476.2462498.

[45] R.D. Resler, D.M. Deaver, VCOCO: a visualisation tool for teaching compilers, SIGCSE Bull. 30 (3) (1998) 199–202, https://doi.org/10.1145/290320.283123.

[46] J.L. Sierra, A.M. Fernández-Pampillon, A. Fernández-Valmayor, An environment for supporting active learning in courses on language processing, SIGCSE Bull. 40 (3) (2008) 128–132, https://doi.org/10.1145/1597849.1384307.

[47] J.J. Castro-Schez, M.A. Redondo, F. Jurado, J. Albusac, Experience applying language processing techniques to develop educational software that allow active learning methodologies by advising students, J. Network Comput. Appl. 41 (2014) 65–79, https://doi.org/10.1016/j.jnca.2013.10.011.

[48] H. Mössenböck, A generator for production quality compilers, in: Proceedings of the Third International Workshop on Compiler Construction, Springer-Verlag, London, UK, 1991, pp. 42–55.

[49] T.J. Schuell, Cognitive conceptions of learning, Rev. Educ. Res. 56 (4) (1996) 411–436, https://doi.org/10.2307/1170340.