Journal of Network and Computer Applications 41 (2014): 65-79

DOI: https://doi.org/10.1016/j.jnca.2013.10.011

Experience applying Language Processing techniques to develop educational software that allow active learning methodologies by advising students

Abstract

This paper is focused on those systems that allow students to build their own knowledge by providing them with feedback regarding their actions while performing a problem based learning activity or while making changes to problem statements, so that a higher order thinking skill can be achieved. This feedback is the consequence of an automatic assessment. Particularly, we propose a method that makes use of Language Processor techniques for developing these kinds of systems. This method could be applied in subjects in which problem statements and solutions can be formalised by mean of a formal language and the problems can be solved in an algorithmic way.

The method has been used to develop a number of tools that are partially described in this paper. Thus, we show that our approach is applicable in addressing the development of the aforementioned systems. One of these tools (a virtual laboratory for language processing) has been in use for several years in order to support home assignments. The data collected for these years are presented and analysed in this paper. The results of the analysis confirm that this tool is effective in facilitating the achievement of learning outcomes.

Keywords: Environment for Active Learning, Formal Languages Techniques, Automatic Assessment

1. Introduction

One of the main learning objectives of any teaching methodology is to allow students to achieve higher order thinking skills (Bloom (1956); Anderson and Krathwohl (2001)) while they acquire knowledge and develop corresponding intellectual capabilities. To achieve this objective, teachers, lecturers and professors can use several teaching-learning methodologies (Oser

and Baeriswyl (2001)). Education in the fields of Science and Engineering usually incorporate learner-centred education (Norman and Spohrer (1996)). The most widespread teaching-learning methodology usually merges theoretical lectures and practical laboratory sessions (or virtual laboratory sessions) in order to provide students an active learning environment (Bonwell and Eison (1991); McConnell (1996)), by applying a Problem Based Learning (PBL) approach (Dewey (1922)). Some researches have pointed out that this approach facilitates more effective and improved learning, whereby students are involved in learning activities that require problem solving (Eden et al. (1996); Makonnen (2000)). Home assignments are usual tasks to involve students in problem based activities. However, it has not been possible to give individual feedback given the current resources (Salmela and Tarhio (2004)).

Because Information Technologies can help to achieve the abovementioned higher order thinking skills (Churches (2008)), students can benefit from the use of Computer Aided Learning (CAL) environments. Our research will focus particularly on those systems that aid students in building their own knowledge by providing feedback regarding the consequences of their actions while engaged in a learning activity. In these systems, feedback is the foundation for the building of knowledge (Gordijn and Nijhof (2002)), as students can modify and improve the solutions they propose (Bravo et al. (2009)). In this regard, the studies of Kumar (2004) suggest that systems which provide any kind of feedback to students are more effective than those that do not, and they can be used as a supplement to classroom instruction (Fernandes and Kumar (2005)). In addition, Sanders and Hartman (1987) noticed that when learners observed the assessment of their assignments, it helped them to justify their choices when solving a problem by evaluating the advantages and disadvantages of each possible choice.

Furthermore, feedback regarding a student's assignments is not only useful for that student, but also for the system, so that it can guide the student's learning process. This is the case with Intelligent Tutoring Systems (ITS) (Murray (1999, 2003)) and Adaptive Hypermedia Systems (AHS) (Brusilovsky and Peylo (2003)), where the system sets the learning-activity flow as a consequence of the assessment of what students deliver on specific assignments. Therefore, in systems of this kind, feedback can be used: (1) to assess the degree in which the objectives are being achieved in the learning process; (2) to determine whether there is a need for replanning the learning activities; (3) to adapt the learning process to a student's specific cognitive

characteristics (Jurado et al. (2008)).

In addition, to allow students to test more alternatives, we hypothesise that they must be able to introduce their own changes to the statement of the problem, so that higher order thinking skills can be achieved. Thus, taking Bloom's Taxonomy (Bloom (1956)) into consideration, in which the thinking skills order is categorised as: knowledge; comprehension; application; analysis; synthesis; and evaluation; and allowing students to test as many alternatives as they wish, and even to introduce their own proposals of assignment in order to validate them, we could state that they are working in the higher order thinking skills: (1) in the application level, because they are able to use their knowledge to solve an assignment; (2) in the analisys and synthesis levels, due to the fact that they are able to create new assignment statements; (3) in the evaluation level, because they are be able to corroborate the accuracy of their evaluation for a specific solution.

Therefore, the aim of our research has been to identify a formal technique that facilitates the building of systems which allow students to specify problem statements, to introduce alternatives solutions to problems and to analyse the corresponding evaluation by mean automatic assessment. This evaluation will provide the corresponding feedback to students and, consequently, they will be able to identify possible mistakes in both the formalisation of the problem and the solution.

Thus, in this paper, we will present an approach that makes use of Language Processor techniques for designing and developing educational software tools that allow students to formalise problem statements, to express possible solutions and to evaluate them. These tools could be applied in those subjects where the problem statement and the solution can be formalised through a formal language, and where the problem can be solved by means of a tractable algorithm to efficiently compute the solution to the problem. In the designed formal languages, their semantic will be syntax-driven. Therefore, the semantic analysis will be carried out directly into the parsing process.

In order to have the corresponding application frameworks to test our approach, we have selected three different engineering courses: Formal Languages and Automata Theory, Language Processors, and Electronic Circuits. Therefore, we have enough scenarios to make a preliminary test of the versatility of our proposal. In this way, as example to clarify our explanation, we will expose our approach by showing the developed tools for the abovementioned courses.

This paper is organised as follows: firstly, an overview of related works is presented; secondly, our approach is described; followed by an example of application of the techniques, which will show how our approach is applied; subsequently the experimental results of the last six academic years are discussed; and finally, some conclusions are drawn.

2. Overview and Related Works

There are several approaches in addressing the problem of analysing the solutions to assignments provided by students in several specific learning domains. Thus, COALA (Jurado et al. (2009, 2012)), uses fuzzy logic to assess programming assignments by comparing the tutor's ideal solution with that delivered by the students, taking into account the imprecision while implementing it. Also, ViLLE Tool (Rajala et al. (2008)) includes techniques and mechanisms to provide automatic assessment and feedback. However, these tools are not used in user-defined domain courses. A good survey about techniques used in this kind of system can be found in (Ala-Mutka (2005)) and (Rahman and Nordin (2007)). They are only applied to programming learning courses.

Co-Lab (Bravo et al. (2009)) analyses similarities of objects, and the relationships between students' solutions and the ideal solution in a modelling process for System Dynamics. DomoSim-TPC (Bravo et al. (2006)) uses a meta-description for Domotics Designs in terms of types of objects, relationships between them, rules for model building, behavior of component model, etc. so that the students solutions meta-description is compared to the ideal meta-description. KERMIT (Suraweera and Mitrovic (2002)) assesses students solutions by using a knowledge base that consists of a set of constraints for conceptual database design. In He et al. (2009), we see an approach that makes a semantic assessment of summaries written by students, by merging Information Extraction and Natural Language Processing.

The main restriction in each of these approaches is that all are dependent on the inclusion of a database with predefined problems and their ideal solutions as specified by experts. These systems must then match the solution delivered by the student with the ideal solution for a specific problem. Each approach implements its specific matching technique in order to detect the differences and similarities between solutions in order to advise students of what they are doing well, and what they are doing wrong. This situation restricts each approach to specific domains, and limits them to a set of problem

assignments.

ACE is a system for automatically assessing assignments related to finite state automata and parsers (Salmela and Tarhio (2004)). This system includes a client to perform assignments as well as verifiers to check them. However, ACE is applied only to a specific domain. In addition, with ACE the assignments can not be defined by students.

So far, we can see that there is no one approach that can be used for several domains. Moreover, none of these approaches allow students to specify new assignment specifications by themselves, apart from the system's predefined assignments. Thus, in the next section we will present an approach that allows the specification of both the problem and the solution, by using Language Processing techniques. With this approach, students will be able to propose their own problems, as well as possible solutions to those problems, while the system gives advice regarding how the proposed solution fits the proposed problem.

3. What is the proposed approach?

In this section, we present our proposed technique for designing and developing educational software tools that allow students to specify problem statements, to introduce alternatives solutions to those problems, and to receive evaluation results from the system. This feedback is essential for students to assess how they are progressing, and it could be used by the tool to guide students through the learning process. Moreover, the tool could also be used by teachers as a classroom aid in order to present and illustrate lessons, to receive immediate feedback in lectures, and to allow examples to be modified accordingly so that it will help students understand them better. This promotes a new method of working in class which breaks with the classical pattern of classroom activity based mainly on one-way knowledge transmission/reception of pre-elaborated concepts.

In the next subsection, we will explain the suggested technique in detail; next we will present the architecture of the designed tools based on this technique; and finally, we will enumerate some guidelines that help to use the suggested technique to design and educational software tool.

3.1. Technique

The purpose here is to introduce what is needed to develop educational software tools as we have just described and, subsequently, to present the

technique suggested to design and build such systems.

First, we need a powerful communication mechanism that allows the users to precisely define problems or problems and their solutions. A mechanism is needed to establish how to specify both problems, and solutions provided by students. To determine how the tools input will be provided, we have considered several alternatives such as by means of forms (Cole et al. (1998)), by using diagrams (Rodger and Finley (2011); Tamagnini et al. (2011)), or by using an input language. The main advantage of input via forms is that it is a guided process which makes it ideal for users who are new to the process, but the drawbacks are that interaction with the tool is slow and the user has to work online. The diagram option is suitable for users who are not specialists in the area, but it shares the same negative aspects as input via forms. Moreover, it does not allow relating results between them, which impedes the experimentation with them. It is for this reason that we chose to use text written in a given formal language as the input of the tool. Although this alternative forces the student to learn a language, it has the major advantage of allowing students to formulate and build problems, and propose solutions to those problems quite easily, even at home without the need of computers. It is the most useful input method in helping students to formalise problems and their solutions.

Next, we need a mechanism to understand the user input in order to acquire the useful information, resolve the problems given and explain how the solutions are reached. Moreover, these solutions should be used to check whether the solutions submitted by users are correct. Then, having taken into account the result of this evaluation, it is necessary to recommend actions to the student. Moreover, the student should be allowed to interact with the output during the explanation process. In this way, we promote active learning, where the student is guided by the system.

The suggested technique to design and build educational software tools with feedback capability consists of:

- A formal specification language to specify the problem or exercise for which the student wants to obtain an expert resolution. Moreover, it must allow students to specify their own proposed solutions. At first, the designer of this language must define its syntactic rules by using context-free rules and then associate each syntactic rule with its corresponding semantic rule.

- A language processor for processing that language and acquiring the

information needed to resolve the problem and to evaluate the student's solution.

- A problem solver (that we will call Resolutor) to provide a solution to each problem and to explain that solution. To achieve this, it generates all the information needed to understand the system's solution.

- A corrector to correct the exercises completed by students, give them feedback, and point out what they have learned or failed to learn.

- An instructor to guide students in their learning process and to assign learning activities. To achieve this, it is necessary to know whether the students have grasped the concepts involved in the learning activity.

- A presenter to create the most appropriate view of the system output, with the aim of allowing students to interact with the problem. The tool could have animation and simulation capabilities to facilitate comprehension of the subject.

Below, we describe the architecture that arises from the suggested technique.

3.2. Architecture of the System

Figure 1 illustrates the suggested system architecture for building educational software based on the aforementioned technique.

The architecture suggested focuses on modular capabilities and emphasises how the interactions between modules are organised. Here we present each module of our application:

- Input Processor. This module receives the exercises (statements and their solutions, optionally) proposed by the user (student or tutor). Unlike the other modules, it has access to the given input. The aim of this module is to check that the input is correct, that is, the input is free of syntax errors (e.g. malformed strings), and semantic errors (e.g. violation of restrictions on the content, for instance, two input exercises with the same associate ID). If there are errors, the module must communicate this to the user, giving as much information as possible in order to help the user solve the problem (e.g. line where the error is located, or guidance on what is occurring) and stop the

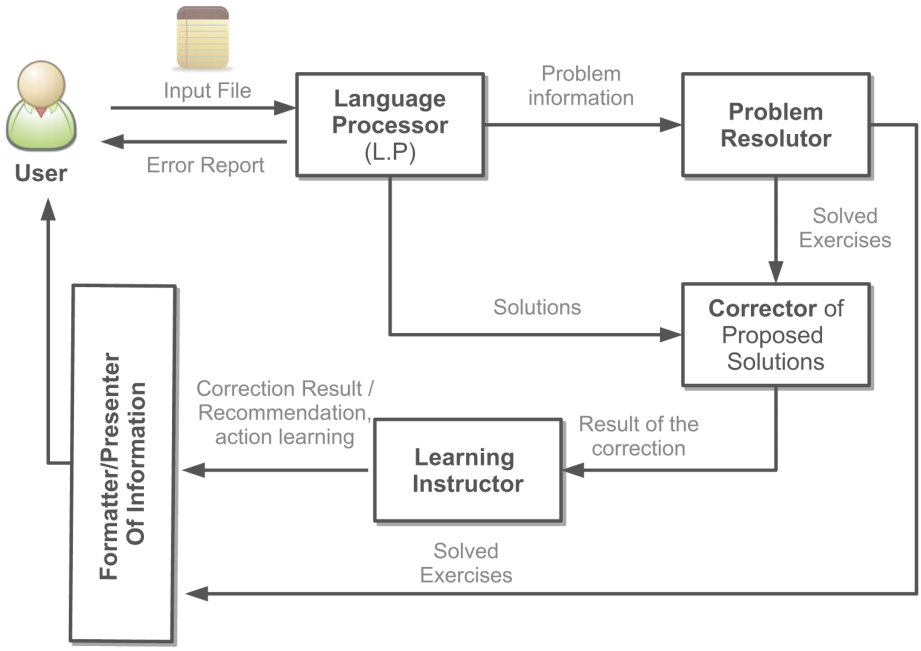

Figure 1: Functional system architecture.

process. On the contrary, if there are no errors, the module should extract useful information for other modules. This information can be sent by means of information flows or stored in data structures for later use. On the other hand, the extracted information relates to exercises to be solved and must be sent to the Problem Resolutor. Moreover, if the user includes a proposed solution, that must also be sent to the Corrector of Proposed Solutions. Finally, the module is based on a Language Processor of a Formal Language which allows inputs for the system to be defined.

- Problem Resolutor. The aim of this module is to invoke the algorithms that provide solutions by using the information gathered by the Input Processor module. Each kind of exercise has an associated algorithm to resolve it. It is important to note that this module always generates an answer at this point in the architecture. The output of this module consists of the steps and temporal information needed to understand the process to be followed in order to solve the problem. We recall that one of the main objectives of the tool is to assist in the learning process. The output information could be stored as an intermediate form of representation to be processed (e.g. XML document) or be stored in data structures for later use.

- Corrector of Proposed Solutions. This module analyses the solutions provided by the user, which have been previously obtained by the Input Processor. A second objective of this module is to compare these solutions with those generated by the Problem Resolutor. Problems and their proposed solutions must correspond, and this association is normally made by means of IDs. Secondly, the module generates a report with the result of the correction process. The report shows students errors and where they occurred. Subsequently, this information is sent to the Learning Instructor.

- Learning Instructor. This module works as a student's personal tutor and as a guide in the learning process. According to the information provided by the previous module, it should recommend learning activities, for instance, to study a particular concept or lesson, or to complete more exercises of different kinds. In order to develop this module, a mechanism is necessary to store a student's status during the learning process and the learning method to apply following a learning design

(IMS-GLC (2003); Koper and Tattersall (2005)). Moreover, the module stores in a database what the student has completed and learned, and the point in the global learning process in which the student is currently located. Thus, the system can guide the student on subsequent occasions when he/she uses the tool. There must also be an element to establish relationships between frequent errors and the learning actions specifically intended to help students avoid those errors. Accordingly, a problem repository and a manager to recommend exercises to students could be included.

- Formatter/Presenter of Information. The aim of this module is to present the information generated for the user in the best way possible. The main idea is to choose the best possible representation in order to promote the use of the system as a learning tool. Thus, the information is represented as a text document, as a graphic, or in any other way that facilitates interaction. To develop this module, we use XML, HTML, figures, custom software or, indeed, interpreters with an associated language, to facilitate interaction between users and generated information.

This architecture could run through a web browser without any need for local installation or automatically-installed plug-ins. Web-based design must be the trend in educational software design, since it simplifies installation and platform compatibility issues, and it provides immediate access (Boroni et al. (1998)). Moreover, it could collect reports and statistics that would allow us to analyse work completed by students.

3.3. Applying the Technique: Guidelines

The following guidelines have been developed (diagrammatically illustrated in Fig. 2) without including phase 0) to show how to design a tool based on the suggested technique. The entire process consists of the following key phases:

- Phase 0. Decide whether the domain fulfills the restrictions imposed for the application of the proposed technique.
- Phase 1. Decide what types of exercises must be resolved by the system.
- Phase 2. Specify the solutions to the types of exercises identified in the previous phase.

- Phase 3. Create a Domain-Specific formal language capable of expressing the exercises and their respective solutions.
- Phase 4. Build a language processor to parse and check the correctness of the student's input and to acquire the meaning of the input text. Its design may be split into the following three independent steps:
  - Step 1. Transform character stream into token stream.
  - Step 2. Parse input token stream, signalling acceptance or rejection.
  - Step 3. Map an input-language sentence into a formal representation of its meaning.

  We recall that the semantic is syntax-driven, that is, each semantic rule will be represented by actions depending on the problem. So, given a syntactic rule, it will perform the mapping of a state into a result or it will do semantic verifications.
- Phase 5. Design and implement algorithms to solve the problems.
- Phase 6. Establish which algorithms should be invoked by formal representations of input-language sentence meaning.
- Phase 7. Design and implement an automatic correction module to mark and correct the exercises/problems whose solutions are provided by students in accordance with the solutions obtained in Phase 5. They must been solved as though on paper.
- Phase 8. Design and implement a module to guide the student's learning process. This module must be related to the automatic correction module.
- Phase 9. Design and implement the output formatter to create output from any of the system's modules, so that it can be used in the most advantageous way possible to improve student learning (and its use by tutors in the classroom).

It should be noted that this technique cannot be applied in all cases, only in those cases in which: (1) we can use a formal language as a tool for representing exercises or problems and their solutions, and (2) there are algorithms to efficiently compute the solutions to those exercises.

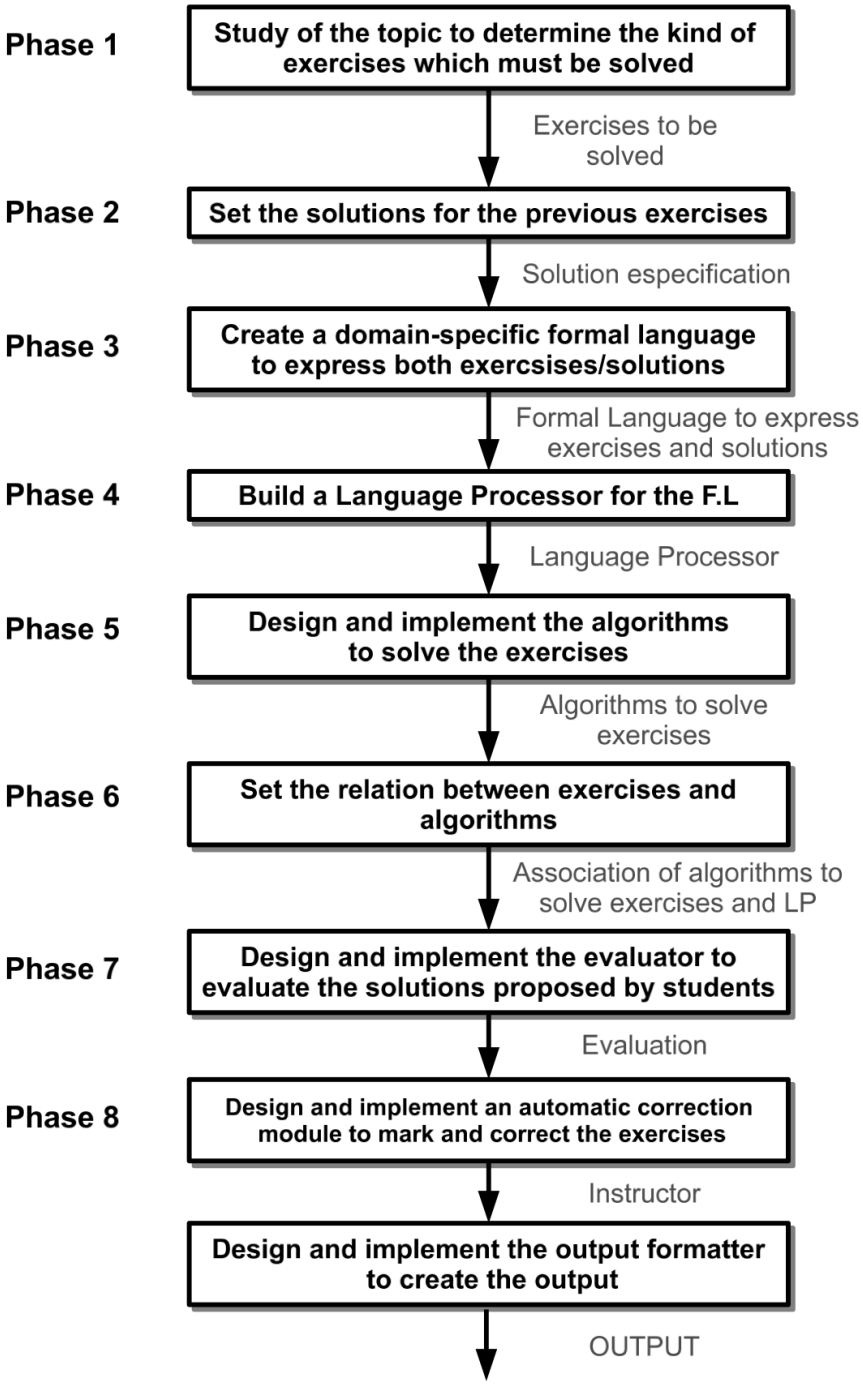

Figure 2: Guidelines to build educational software based on the suggested technique.

4. Example of application

This section examines in detail how the suggested idea is used in the design of an educational software tool to be used in the subject of Language Processors. This is just a case of study in order to show how to apply our approach. The aim of this tool is to make it easier both to teach and to learn the principal concepts that comprise this subject. Additionally, two further implemented applications are also briefly shown, with the aim of verifying the validity of the aforementioned idea. In these last two examples, we have kept the discussion to a minimum in order to give a description of how the idea is used rather than to provided a detail description of how the solution has been reached and implemented.

4.1. PROLETOOL: A tool to be utilised both in teaching and learning Language Processors

One concern that must exist in the teaching profession is the continuing effort to improve the quality of training. To meet this goal, we encounter the problem of the design and development of materials and applications that will help to achieve improvements in training. In this regard, *Proletool* is an educational software tool designed to be used both by teachers and students for improving teaching and learning activities in courses pertaining to the construction of language processors (Castro-Schez et al. (2011c)).

The purpose of *Proletool* is to complement materials that currently exist (for example, see books by Aho et al. (2006); Grune et al. (2000); Cooper and Torczon (2011)), in order to assist in the understanding of the relationship between lexical and syntactic analysis stages, and to facilitate the teaching and learning of top-down parsing techniques: LL1; and bottom-up parsing techniques: SLR1, LR1 and LALR1; which are discussed in language processor courses for graduate students of Computer Science and Information Technology.

4.1.1. Issues Proletool faces

The main aims and objectives of the subject Language Processors (LP) are to introduce students to:

- the concepts underlying the design and implementation of language processors;

- the problems that arise during the construction of language processors, translators, compilers and interpreters for programming languages, or more generally, to introduce students to any software program in which a formal language is the communication mechanism available;
- the algorithms for the implementation of mechanisms used in providing the intended functionality of these systems to their users, in particular, lexical and syntax analysis, context handling, and code generation and optimisation.

To build these types of systems, knowledge of grammar and formal languages and machines for processing them is an essential requirement. The contents of an introductory LP course are organised around the following thematic units:

- Introduction. This unit introduces the topic of the subject, presents the basic terminology of the field and shows how processing can be divided into four principal phases: lexical analysis, syntactic analysis, contextual analysis, and code generation.
- Formal Languages and Grammars Review. Since, a language processor works with formal languages, and a grammar is meant to describe all lexical and syntactic constraints of a formal language, a review of formal languages and grammars is given in this unit. Moreover, formal language theory has led to tools that automate the production of scanners and parsers.
- Lexical Analysis. This unit focuses on the lexical analysis stage, showing how to group input characters into tokens which will be used by the parser or syntax analyser. Additionally, other key tasks of the lexical stage are studied.
- Syntax Analysis. This unit is concerned with the study of syntactic analysis, showing how a parser can be constructed from the source language's syntactic specification. Top-down and bottom-up parsing techniques are studied in detail.
  - Top-Down Parsing Techniques (LL1).
  - Bottom-Up Parsing Techniques (SLR1, LR1, LALR1).

- Semantic Analysis or Context Sensitive Analysis. This unit concentrates on correctness and meaning of sentences which are well formed. This is done at the level of checking parts of speech against grammatical rules.

Special importance is given to the study of the most commonly used techniques in developing a lexical analyser, and the most commonly used techniques of syntactic analysis. This is because many subsequent stages depend on these elements. *Proletool* will be used both in teaching and learning these essential concepts and techniques.

4.1.2. Proletool description

The main functionalities that the tool must offer, from the point of view of teachers and students alike, are the following:

1. It may be used as a classroom aid to help tutors to improve instruction. Tutors develop examples to explain subject concepts and algorithms. These examples will be given as input to *Proletool* and its output projected onto a whiteboard at the front of the class. The use of examples to teach concepts and algorithms could be decisive in helping students understand them better. Moreover, the tool allows tutors to receive immediate feedback in lectures and to modify examples accordingly. Consequently, teaching activities are improved.
2. It may be used as a personal tutor to help students experiment with the concepts and their relationships in order to fully develop a solid understanding of them. Usually, the completion of a large number of exercises helps students to achieve significant learning and knowledge of the concepts with respect to the aforementioned topics and their relationships. However, this cannot be accomplished in lectures, due to the extent of the set of themes to be learned and the restricted time available for lectures. Using the tool, students develop exercises which are then processed by the tool before an output is generated, providing solutions to those initial exercises. There are no limitations regarding the number of exercises that can be submitted to the tool. Furthermore, students could relate the subjects concepts by correlating the output associated with one exercise with the input associated with another exercise. As a result, the learning activity is more active which helps students in developing their knowledge.

3. It may be possible to become a LP instructor. The instructor must have the ability to identify what the student has learned or failed to learn, so that he/she can plan for future programmes. This information must then be used to guide students through the process of learning.

In order to achieve this, it will be necessary, firstly, to have a mechanism that allows students to present exercises and their respective solutions to the tool. Secondly, it will be necessary to have a mechanism with the capacity to understand and resolve these exercises appropriately, and to check the accuracy of solutions provided by students. Finally, a mechanism whereby feedback is presented to students regarding exercises submitted should be incorporated, so that students can interact with the exercises to further develop their knowledge.

An exercise is the specification of a grammar, $G = (V, T, P, S)$, where V is the set of nonterminal symbols, T is the set of terminal symbols, P is the set of production rules that are used for transforming strings, and S is a distinguished symbol from N ($S \in N$) which is the start symbol, to which one or several kinds of Syntax Analysis Techniques (LL1, SLR1, LR1, LALR1) are associated. Moreover, T could be explicitly specified, or be given implicitly using a previously defined student's lexer.

A solution is the generation of an appropriate analysis table based on the Syntax Analysis Techniques associated with the exercise. Moreover, additional information will be used to generate the solution.

The educational software tool *Proletool* consist of:

- A formal language that encompasses both the specification of exercises and their solutions. Therefore, it has a language processor to analyse and understand this language and to extract the information that is necessary to invoke the algorithms that resolve the exercises. Once these solutions have been generated by the tool, it checks whether they are the same as those provided by the student.

- A database to collect statistics which will then be used to determine the stage at which the student is in the learning process. This is very important as it will be used to make subsequent recommendations to the student, for example, "*You must study the way FIRST set is calculated*" or "*You must do the following exercise ...*".

- A web-based architecture for enabling streamlined student-tool interaction throughout the learning process (see Fig. 3).

- An output interface with interactive animation and simulation capabilities to study how the techniques work, in addition to static pictures and text.

Below, a simple example of input in the *Proletool* input language is shown:

```
grammar gramatica
{
    analysis    SLR1, LR1;
    nonterminal S, T, L;
    terminal    int, char, id;

    S := T L ';';
    T := int      | char;
    L := id ',' L | id;
}
```

The tool's output with respect to this input is not shown here. In the same way, discussion with respect to the design of the tool has been reduced to a minimum due to space limitations and in the interests of clarifying the really important aspects of the ideas presented in this paper. The tool can be tested in Castro-Schez et al. (2011c) where it can be found these issues as well as an accurate description of *Proletool*'s input language.

4.2. SELFA-Pro: SoftwarE for Learning of Formal languages and Automata theory

In this section, we briefly present the *SELFA-Pro* project (Castro-Schez et al. (2011b)), an educational software tool designed to improve both the teaching and learning of the main concepts of the subject Automata Theory and Formal Languages (ATFL). This subject has great importance in the formation of Computer Science engineers, because it provides the theoretical basis of computation. It helps foster a better understanding of computer science and its mathematical origins, as well as its strengths, limitations and potential (Carrol and Long (1989); Harrison (1978); Hopcroft et al. (2001); Kelley (1998); Lewis and Papadimitriou (1997); Martin (2003)). Moreover, from a computational point of view, it is useful to present models of computation that could be used in the resolution of real problems.

The main aim of the ATFL subject is that students acquire knowledge and develop abilities with respect to automata, formal grammars and languages, which will allow them to analyse, understand and solve problems. An ATFL course usually involves the following topics:

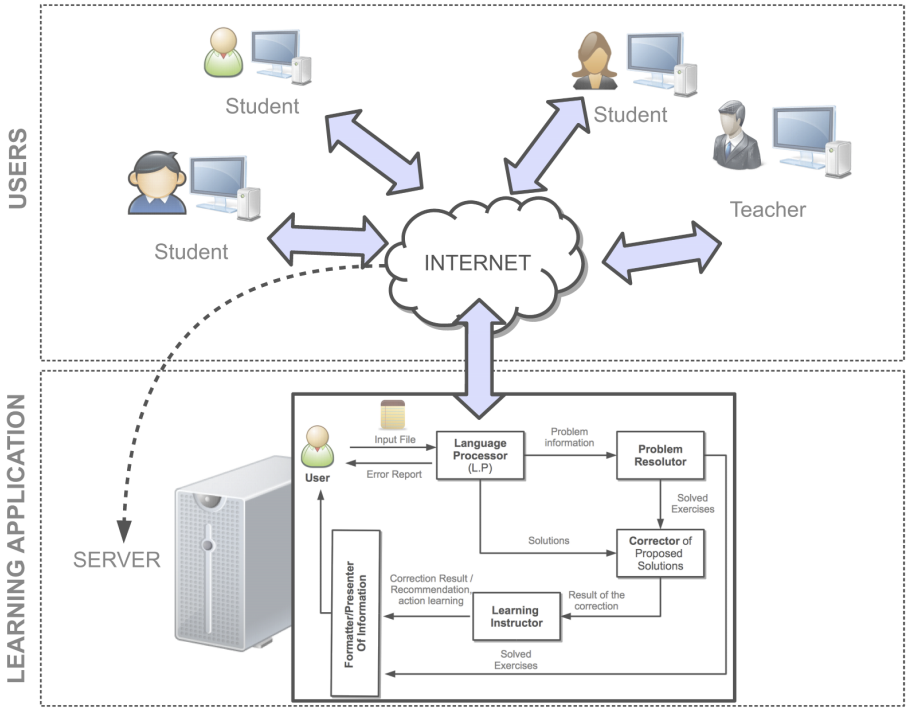

Figure 3: Deploying web-based architecture of *Proletool*.

- Introduction to abstract machines, formal languages and grammars.
- Regular languages and grammars, Finite Automata and regular expressions.
- Context-free languages and grammars and pushdown automata.
- Unrestricted and context-sensitive languages and grammars and Turing Machines.

SELFA-Pro is intended to be used by two categories of user: computer science students and university lecturers. The aim of this tool is to facilitate both the teaching and learning of the main concepts of the ATFL subject, whose level of abstraction makes both activities difficult. To accomplish this, the tool allows a set of objects (automata, grammars and regular expressions) to be defined, manipulated and used by means of algorithms. The tool also allows previous results to be used in subsequent work. The exercises that *SELFA-Pro* solves can be put into the following four groups:

- exercises on finite-state automata (e.g. to convert a non-deterministic automaton with null transitions ($\lambda-$NDFA) into non-deterministic finite automaton (NDFA), to convert a finite automaton ($\lambda-$NDFA or NDFA) into a deterministic finite automaton (DFA), to obtain the minimal deterministic automaton equivalent to a given DFA,);
- exercises on pushdown automata (e.g. to obtain a context-free grammar from a pushdown automaton,);
- exercises on grammars (regular and context-free) (e.g. grammar cleaning algorithms such as removing lambda productions or useless symbols and productions, to convert a context-free grammar to Chomsky Normal Form,. . .);
- exercises on regular expressions (e.g. to convert a regular expression to a $\lambda-$NDFA using Thompsons algorithm).

Each exercise solved requires the design and implementation of several algorithms.

The main features of *SELFA-Pro* relevant to the ideas presented in this work are the following:

- It uses Client/Server architecture.
- It is based on a formal specification language and its associated processor. This language will be used to define the automata and grammars, and to manipulate and relate them (e.g. it allows two regular languages to be defined by means of two regular expressions, it then allows two non-deterministic automata that recognise the same languages to be obtained from previously defined regular expressions, and then it checks if they are equivalents, that is, if they recognise the same language).
- It uses images and visual representations to display results so that they are easier to understand and interact with (e.g. to show how a non-deterministic automaton with null transitions works).
- It has ability to solve exercises, explaining how the solution is reached (e.g. to answer how a deterministic finite automaton is obtained from a regular expression).

The main advantages of *SELFA-Pro* in comparison to other software tools designed with the same purpose (Devedzic et al. (2000); Garcia-Osorio et al. (2011); Rodger and Finley (2011); Tamagnini et al. (2011); Tran (2007)) are the followings:

- It uses web technology to make it accessible by means of standard web browsers.
- It has a text mode input that allows the user to interact quickly with the tool.

Below, we demonstrate how the tool could be used by a tutor to explain how to test the equivalence of Regular Languages. Hopcroft and Karp (1971) presented an algorithm for testing the equivalence of two deterministic finite automata (DFA). Considering the two merged DFAs as a single DFA, they study if their respective start states are equivalent. The tutor could prepare an example to explain this to the class, in which it is demonstrated that for each regular expression it is necessary to obtain its equivalent nondeterministic finite automaton (NFA). It could subsequently be demonstrated how the equivalent deterministic finite automaton (DFA) is obtained for each NFA. Finally, it could be demonstrated how the Hopcroft and Karp algorithm works. This example in *SELFA-pro* language is as follows:

```
/* Deciding equivalence of regular expressions exp1 and exp2 */

regexp exp1{ 1(01)*0 }
regexp exp2{ (10)+ }

/* Every regular expression has an equivalent nondeterministic
   finite automaton(NFA), they match the same strings */

automata1=REtoFA(exp1);
automata2=REtoFA(exp2);

/* Every NFA has an equivalent deterministic finite
   automaton(DFA), they match the same strings */

automata3=FAtoDFA(automata1);
automata4=FAtoDFA(automata2);

/* Graphical representation of each DFA is shown  */

print(automata3);
print(automata4);

/* Hopcroft and Karp algorithm is applied */

equals(automata3,automata4);
```

Moreover, if a tutor wants to explain to students how an automaton works, it would be helpful to make use of graphical representation and animation so that the audience might better understand the concept. In this regard, *SELFA-Pro* offers operations for visualisation and simulation (see Figs. 4,5). The following copied text should be added to the end of the previously typed text:

```
/* Invoking order to show in a visual way how
   automata1 works with the input 1 0 1 0,
   See Fig. 1 */

visualrecognize(automata1,1 0 1 0);

/* Invoking order to show in a visual way how
   automata3 works with the input 1 0 1 0, See Fig. 2 */

visualrecognize(automata3,1 0 1 0);
```

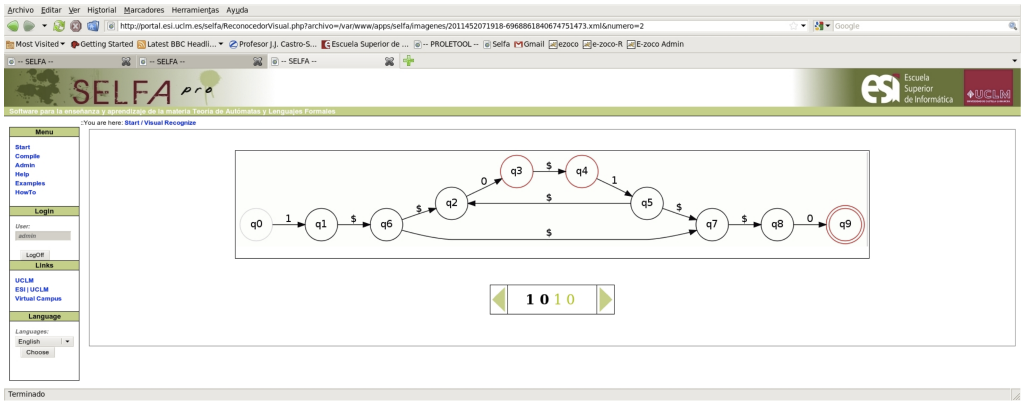

Figure 4: Graphical representation and animation capacity for `visualrecognize(automata1,1 0 1 0);`.

This example can be tested in (Castro-Schez et al. (2011b)). An accurate description of *SELFA-Pro*s input language could be obtained from (Castro-Schez et al. (2011b)). For a more detailed description *SELFA-Pro*, its design, and how it is used, please see Castro-Schez et al. (2009).

4.3. ELECTRONICS: A tool for both designing and answering questions about electrical circuits

The main goal of this educational software, called *Electronics* (Castro-Schez et al. (2011a)), is to provide a tool for both teaching and learning the fundamentals of electrical circuits. It could be incorporated into lectures and laboratory sessions of introductory courses in Circuit Analysis for Electrical and Computer Engineering, with the aim of teaching and learning to predict the voltages and currents in electrical circuits. It introduces Mesh Analysis, which is a powerful method for circuit analysis. The *Electronics* software package has been designed to be accessed by all students in computer laboratories.

The main features of *Electronics* relevant to the ideas presented in this work are the following:

- It has a formal language to design electrical circuits, that is, to specify the interconnection of analogue electrical elements such as resistors, inductors and capacitors. Therefore, it has a language processor for processing and understanding the circuit description.

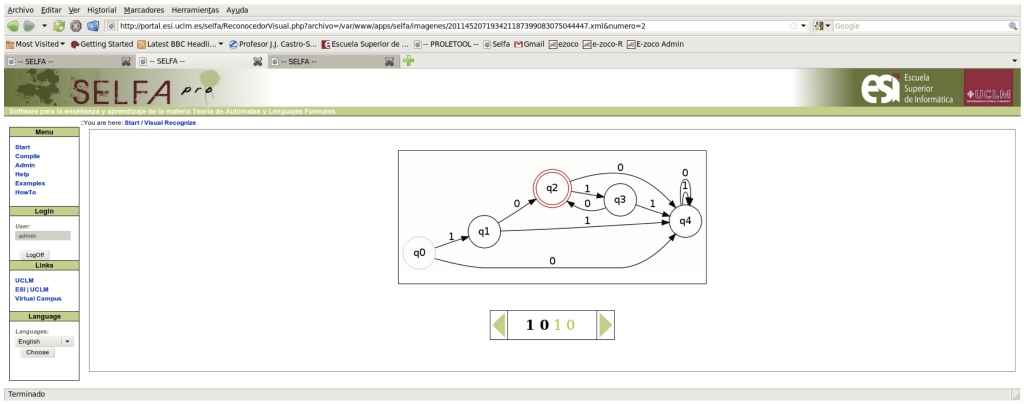

Figure 5: Graphical representation and animation capacity for `visualrecognize(automata3,1 0 1 0);`.

- It has a formal language to make queries regarding voltages and currents between nodes with respect to previously defined circuits. Therefore, it has a language processor that could be considered as an interpreter. The interpreter reads each query from the user and it is answered on the fly. This happens every time the user submits a question.

To use the system, we must first define the electronic circuit consisting of a series of nodes connected to form edges (branches), which in turn can hold a maximum of three components, that is, sources, resistors and capacitors. To accomplish this, we must describe the electronic circuit using a formal language. Once this is done, *Electronics* uses a compiler to understand the electronic circuit and gives a graphical representation of the circuit diagram.

Below, we show an example of an electronic circuit description using the *Electronics* input language:

```
program example_circuit {

/* First, the nodes of the circuit are defined. Node refers
   to any point on a circuit where two or more circuit
   elements meet */

  nodo a (0,0);
  nodo b (130,0);
  nodo c (250,120);
```

```
  nodo d (370,0);
  nodo e (500,0);
  nodo f (0,240);
  nodo g (250,240);
  nodo h (500,240);

/* Second, the nodes are connected according to the design */

  arista AB (a,b);
  arista BC (b,c);
  arista CD (c,d);
  arista DE (d,e);
  arista GC (g,c);
  arista FG (f,g);
  arista GH (g,h);

/* Third, a mesh is define. A mesh is a circuit loop. */

  malla l (AB,BC,GC,FG);


/* Last, individual electronic components, such as resistors,
   capacitors are defined. Moreover, mesh currents are given. */

  fuente fuent (c,b,0.2);
  resistencia resis (FG,2.4);
  condensador cond (AB,3);
  fuente fuent1 (c,d,2.2);
  resistencia resis1 (GC,0.2);
  condensador cond1 (DE,4.4);
  fuente fuent2 (h,g,3.6);
  resistencia resis2 (AB,2.2);
}
```

After, the compilation process of the electronic circuit given above, the system creates the graphical representation show in Fig. 6.

Once *Electronics* has loaded and compiled the circuit, we could then apply the Mesh Analysis. The user could ask for calculations of potential and intensity with regard to that circuit and an interpreter would process the user's queries.

Fig. 7 shows an example of a potential operation on the mesh defined in the circuit. To invoke this operation, the user must specify the intensities passing through each one of the edges that define the mesh. Then the interpreter analyses the question and detects if there is a mesh defined between

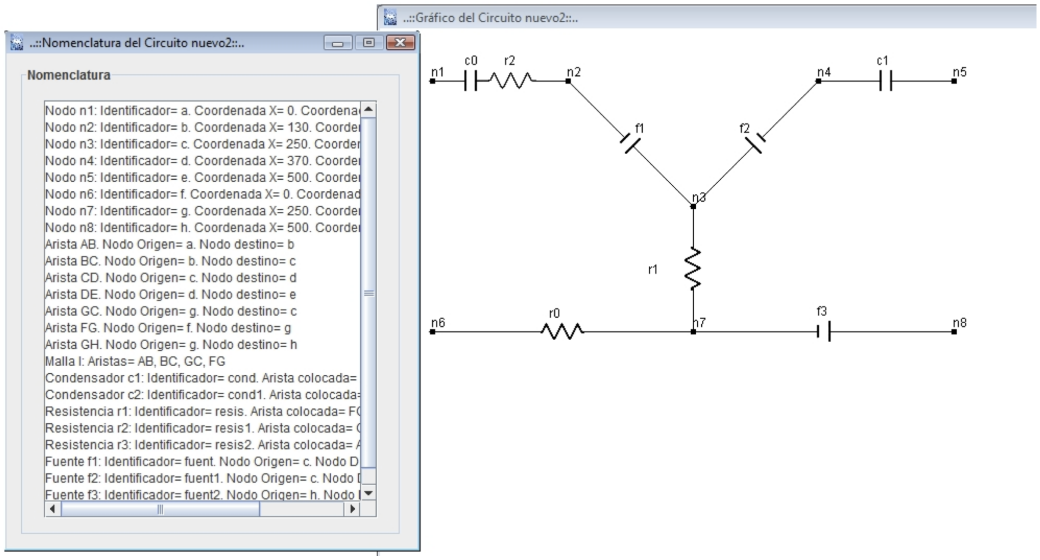

Figure 6: Viewing the circuit and nomenclature.

the source and destination of the calculation of potential, and also if the intensity is specified for each branch of the mesh.

An example of calculation of intensity can be seen in 8. Firstly, the user specifies the potential of each one of the branches between source and destination nodes of the operation, and then the system calculates the current through these branches.

Though still at the prototype stage (see *Electronics* Project Home Page (Castro-Schez et al. (2011a)), this tool shows how the ideas presented in this work are useful in defining circuits and answering questions about them.

4.4. Summing up the Examples

In order to sum up what the previous tools have in common and what are their main distinct features, we show the Table 1.

The features we have considered just have been described in previous sections. Thus, we focus on highlighting that *Selfa-Pro* can management most complex statements. However, Proletool is the most complete and it is being used in courses of Computer Science at our faculty for seven years. Since its inception in 2005, *Proletool* has been in continuous operation and has been improved and upgraded between 2005 and 2012 until reaching the status described in this paper (see Section 3).

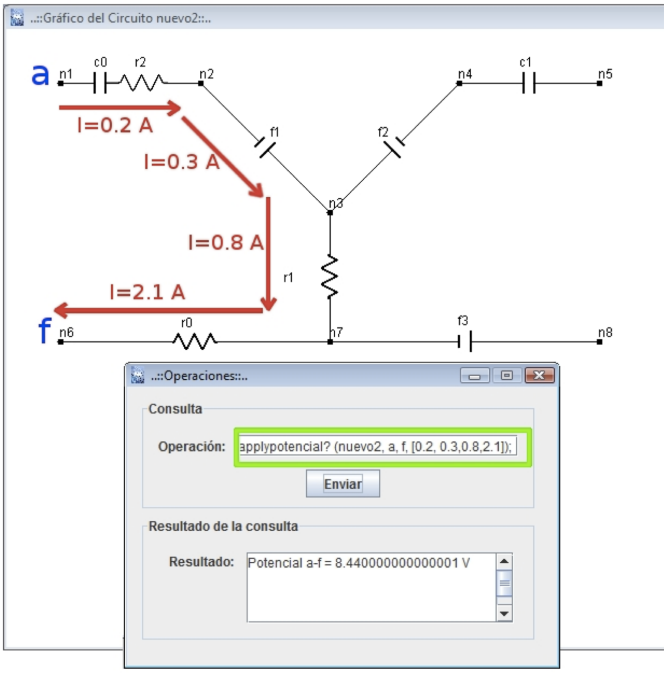

Figure 7: Example of calculation of potential.

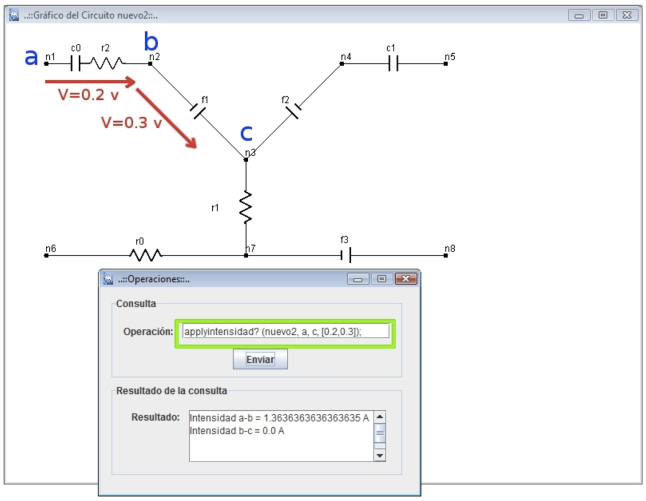

Figure 8: Example of calculation of intensity.

For this reason, next section describes the registered data using *Proletool* during these years/courses and draws some conclusions that we can extract analysing them. Nevertheless, we recall that with the description of these tools we just intend to show how to apply our approach and the result we obtained.

Table 1: Main features of Proletool, Selfa-Pro and Electronics

	Tools		
Features	Electronics	Selfa-Pro	Proletool
Input based on a domain-specific language (DSL)	X	X	X
DSL processed by means of a specific language processor (interpreter/compiler)	X	X	X
DSL allows to user specify new statements/assignments	X	X	X
Support statements/assignments resolution	X	X	X
Complexity of the problems supported by the tool	low	high	medium
DSL allows to user specify solutions and alternative solutions to the statements/assignments			X
Support automatic correction of specified solutions			X
Returns students assignments, with detailed notes and grade			X
Database to collect statistics in the use		X	X
Database to collect statistics in the learning process			X
Automatic assessment of the interactions with the tool to guide the learning process			X
Provides feedback to students about their learning process			X
Provides feedback to professors		X	X
Formatter/Presenter of Information	X	X	X
Implementation based on a Client/Server architecture		X	X

Table 2: Some statistics extracted from the use of *Proletool*

	Course 05/06	Course 06/07	Course 07/08	Course 08/09	Course 09/10	Course 10/11	Course 11/12
Nr. Submits	3717	2142	3262	1916	1082	1126	2719
Compiled Grammars	3884	2242	3384	1942	1086	1147	2747
Total Analysis	5185	2670	4027	2264	1249	1626	3851
Analysis LL1	1751(33,8%)	1037(38,8%)	1971(48,9%)	1057(46,7%)	469(37,6%)	564(34,7%)	1059 (27,5%)
Analysis SLR1	1466(28,3%)	557(20,9%)	707(17,6%)	339(15,0%)	322(25,8%)	468(28,8%)	1060 (27,5%)
Analysis LR1	1349(26%)	724(27,1%)	757(18,8%)	465(20,5%)	180(14,4%)	250(15,4%)	964 (25,0%)
Analysis LALR1	619(11,9%)	352(13,2%)	592(14,7%)	403(17,8%)	278(22,3%)	344(21,2%)	768 (19,9%)
Nr. Simulations	440	686	560	789	677	253	652
Simulations LL1	188(42,7%)	358(52%)	317(56,6%)	325(41,2%)	501(74,0%)	129(51,0%)	228 (35,0%)
Simulations SLR1	127(28,9%)	62(9,0%)	79(14,1%)	62(7,9%)	29(4,3%)	36(14,2%)	167 (25,6%)
Simulations LR1	68(15,5%)	216(31,5%)	68(12,1%)	70(8,9%)	10(1,5%)	61(24,1%)	104 (16,0%)
Simulations LALR1	57(13,0%)	50(7,3%)	96(17,1%)	332(42,1%)	137(20,2%)	27(10,7%)	153 (23,5%)

5. Showing results using Proletool between 2005 to 2012

The purpose of this section is to present the data collected during the period 2005-2012 showing the use of the *Proletool*. This tool has been used by the students to learn concepts and techniques from the Language Processors course at the Computer Science and Engineering Faculty of the University of Castilla-La Mancha (Spain). Moreover, the tool was used freely by students who considered it useful to study the subject. That is, no experiments were designed (not an experimentation itself) and the free use of the tool could be seen as an indicator of its utility.

In this section, we also show some data of students' academic results during the period 2000-2012 in the course where has been used the tool. This course was taught by the same teacher in all years of the period and assessment and evaluation were similar. In addition, we will analyse the impact of the Proletool usage on the academic results obtained by the students and its influence in the students' scores. This analysis comprises the academic years 2005 to 2011, when the tool is used, against the academic years from 2000 to 2004 when the tool was not available.

First, in Table 2 we show some data regarding the use of the tool which are related to the concepts and techniques of the subject: submissions and compiled grammars (problem specifications and solutions to those problems make at home and how many times they compiled them), as well as the number of analyses performed and simulations subdivided by the type of analysis. Additionally, Table 3 shows the anonymous usage data (unregistered user). As it can see, a large amount of activity has been performed anonymously. That means that the tool is used not only by the professor and the students of the subject where the tool is primarily used, but also by others that are interested in the tool. Perhaps, the consider it useful.

Table 4 shows the degree of satisfaction of users with the tool. In Proletool has been added a form where users can provide feedback regarding the perceived utility (The tool has been useful in learning the subject concepts) and usability (The tool is easy to use) of the tool by means of a Likert scale. The format of the five-level Likert item used is: 1) strongly disagree; 2) disagree; 3) neither agree nor disagree; 4) agree; 5) strongly agree.

Tables 2, 3 and 4 show not only that students used the tool over a number of years and still use it, but they also think that it is easy to use and useful. Therefore, students consider that it is easy to specify new learning activities statements and that the system is able to provide the corresponding feedback

regarding the relevant solutions delivered.

Table 3: Some statistics collected regarding the use of *Proletool* for the anonymous user

	Course 05/06	Course 06/07	Course 07/08	Course 08/09	Course 09/10	Course 10/11	Course 11/12
Nr. Compiled Grammars	1832	1348	2256	1590	886	626	1911
Nr. Simulations	256	573	473	754	658	213	600

Table 4: *Proletool*'s utility and usability perceived by the students

	Course 05/06	Course 06/07	Course 07/08	Course 08/09	Course 09/10	Course 10/11	Course 11/12
Nr. Received Comments	42	51	56	21	16	34	29
Avg. Utility (1-5)	4,3	4,4	4,6	4,6	4,8	4,8	4,6
Avg. Usability (1-5)	4,0	4,0	4,6	4,5	4,6	4,4	4,5

Proletool provides to the students with the mechanisms to solve any exercise that they could propose and it also allow to evaluate their respective solutions if they are present. As a consequence, it allows to work on high order cognitive skills levels and provides an active learning environment. But also raises a question of general significance: has the use of the tool some influence on the students' academic results? This can be evaluated using the data presented in Tables 5, 6 and 7. Table 5 shows the percentages of students that obtained a given score for the Language Processors subject. We can see from the data registered since the 2005 academic year, differences between those students that used Proletool in order to prepare the subject and those that did not: Students which used Proletool for their homework obtained higher scores in the exams. On the other hand, in Table 6 we can see the percentages of students per academic year that passed the subject before the tool was available, and Table 7 exhibits the percentages of students per academic year that passed the subject after the tool was available. This last table also collects information about the academic results of those who used the tool when it was available and from those who did not it. Graphically, this information is shown in Figures 9 and 10.

In order to analyze if the scores for the students that passed the subject by using *Proletool* is significantly better than the others, we have applied

Table 5: Percentages of students that obtained a given score (in a 0 to 10 scale) for the Language Processors subject

		Score					
Course	Tool Use	$5 \leq x < 6$	$6 \leq x < 7$	$7 \leq x < 8$	$8 \leq x < 9$	$9 \leq x \leq 10$	Pass rate
05/06	Yes (41)	36,59%	34,15%	7,32%	9,76%	2,44%	90,24%
	No (61)	47,62%	14,29%	14,29%	0,00%	0,00%	76,19%
06/07	Yes (24)	54,17%	20,83%	0,00%	8,33%	8,33%	91,67%
	No (78)	48,00%	16,00%	4,00%	0,00%	0,00%	68,00%
07/08	Yes (20)	35,00%	20,00%	15,00%	10,00%	5,00%	85,00%
	No (58)	31,25%	25,00%	6,25%	0,00%	0,00%	62,50%
08/09	Yes (19)	52,63%	26,32%	5,26%	5,26%	5,26%	94,74%
	No (36)	50,00%	11,76%	5,88%	2,94%	2,94%	73,53%
09/10	Yes (8)	37,50%	25,00%	12,50%	0,00%	25,00%	100,00%
	No (39)	39,13%	8,70%	8,70%	13,04%	0,00%	69,57%
10/11	Yes (21)	27,27%	31,82%	18,18%	9,09%	9,09%	95,45%
	No (27)	69,23%	7,69%	0,00%	7,69%	0,00%	84,62%
11/12	Yes (22)	50,00%	36,36%	4,55%	4,55%	0,00%	95,45%
	No (13)	28,57%	42,86%	14,29%	0,00%	0,00%	85,71%

Table 6: Students who pass before the availability of *Proletool*

	Course 00/01	Course 01/02	Course 02/03	Course 03/04	Course 04/05
Students enrolled	49	55	70	103	106
Pass	34,69%	54,55%	47,14%	43,69%	34,91%
Not pass	65,31%	45,45%	52,86%	56,31%	65,09%

Table 7: Students who pass after the availability of *Proletool*

	Course 05/06	Course 06/07	Course 07/08	Course 08/09	Course 09/10	Course 10/11	Course 11/12
Students enrolled	102	102	78	55	47	48	35
Pass	52,94%	40,20%	53,85%	76,36%	51,06%	77,08%	77,14%
Not pass	47,06%	59,80%	46,15%	23,64%	48,94%	22,92%	22,86%
Students whom use	41	24	20	19	8	21	22
Use and pass	90,24%	91,67%	85,00%	94,74%	100,00%	95,45%	95,45%
Not use and pass	76,19%	68,00%	62,50%	73,53%	69,57%	84,62%	85,71%

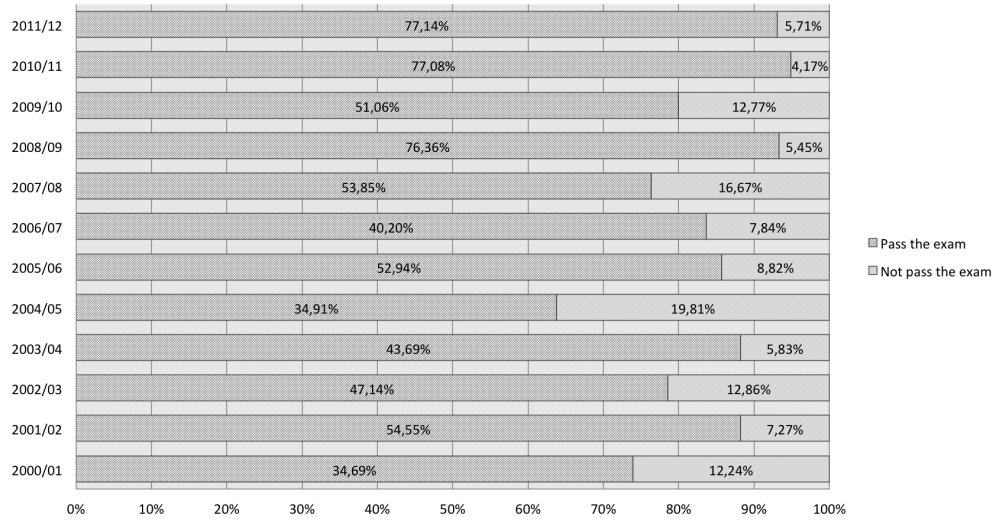

Figure 9: Percentage of students pass and not pass the exam before the use of *Proletool* and while using it

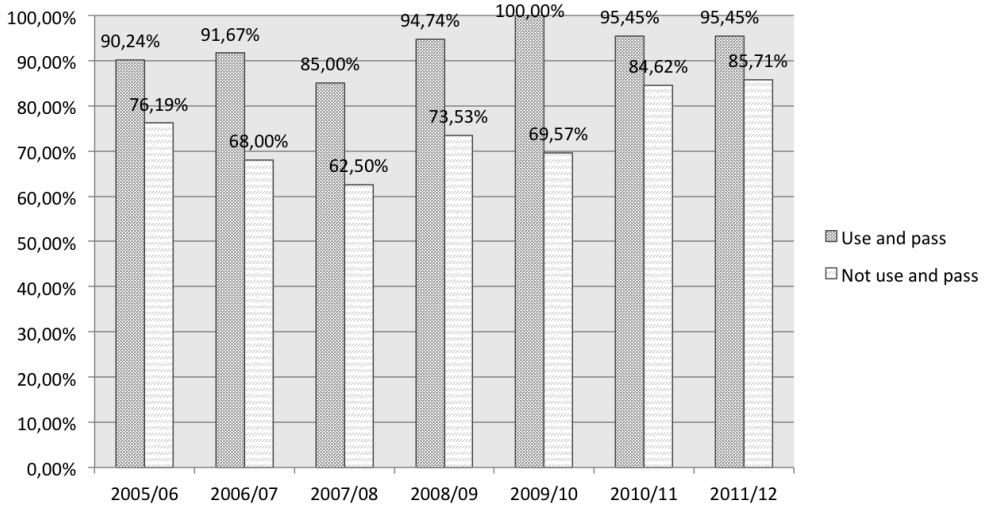

Figure 10: Percentage of students that use *Proletool* and pass and those who do not use it and pass

Table 8: Some statistics obtained from the samples of scores

	Before	Using	Not using
Sample size	250	156	139
Sample mean	4.82	6.07	5.13
Sample std. deviation	2.06	1.40	1.56

the corresponding t-test. It will be used to compare whether the average difference between the groups of students is really significant or if it is due instead to random chance. Thus, Table 8 shows the number of items for the sample, as well as the means and standard deviations for the scores of that students before the availability of Proletool (labeled before), and that who use it (labeled use) and do not use it (labeled do not use) after its availability. Furthermore, Figure 11 shows the box plot for those samples. The obtained results for the corresponding two-sample t-test with a 0.05 (or 5%) of significance are:

1. T-test for "use" against "not use": $t_{value} = 5.44$, $p_{value} = 1.125 \cdot 10^{-07}$. Then, it can be concluded that the null hypothesis can be rejected. Thus, the means for the scores of that students who "use" *Proletool* against "do not use" it are different and slightly higher for those who "use" it.
2. T-test for "before" against "use": $t_{value} = -6.667$, $p_{value} = 8.592 \cdot 10^{-11}$. So, the null hypothesis can be rejected. Therefore, the means for the scores of that students who "do not used" *Proletool* before its availability and that "use" it now are different and a little higher for those who use it now.
3. T-test for "before" against "do not use": $t_{value} = -1.547$, $p_{value} = 0.1227$. Therefore, the null hypothesis can not be rejected. The means for the scores of that students who "do not used" it before the availability of *Proletool* against those who "do not use" it now, have not changed significantly in order to think that they have different distributions.

Regardless, what we can see with all this analysis is that the students who pass the exam with better score usually use *Proletool*. This could be because: a) the students are excited about to have found in Proletool the way in order to practice without limits and without require support from the

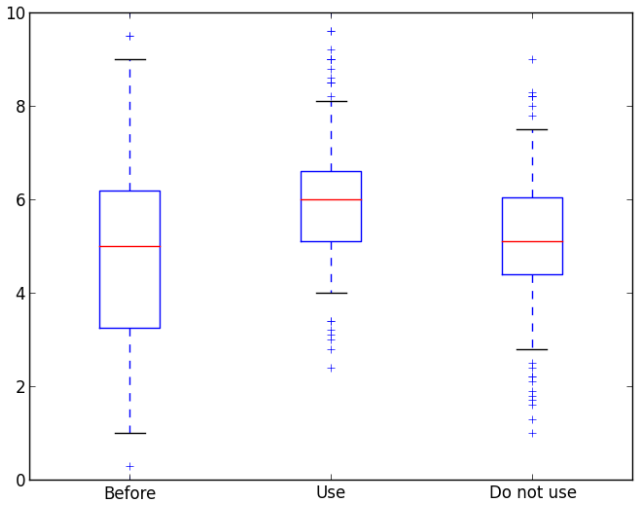

Figure 11: Boxplots comparing the scores for students before *Proletool*, and for those who use it and for those who do not use it

teacher, which use to be one of the goals in tutoring system; b) the tool helps them a lot and therefore they have passed and, because students typically want to pass, there are many who decide to use it in order to pass the course.

Nevertheless, these data allow to us to conclude that the tool has already been successfully used in teaching and in learning the Language Processors subject at the Computer Science and Engineering Faculty of the University of Castilla-La Mancha (Spain). These collected data suggest that our approach is satisfactory, especially when comparing the scores for students who used *Proletool* with those who do not.

6. Concluding remarks

As previously mentioned, we have focused our research on those systems that aim to assist students in developing their own knowledge by providing them with feedback regarding the learning activity they are performing for their homework. Thus, they can build their knowledge by means of modifying and improving their solutions. We have started introducing and analysing some related works with regard to the automatic evaluation of learning activities, showing that there are no integral approaches that can be

used independently of the learning domain or which allow students to specify their own learning activities. To solve this issue in some learning domains, we have described a proposal that allows students to specify their own assignments, introduce alternative solutions to those assignments and view an automatic evaluation provided by the system in order to check the validity of the solutions provided. Our proposal makes use of Formal Language and Automata Theory techniques, both in the definition of formal languages for the assignment statement description and the respective solutions specification, and in the development of computational models to solve the problem and to analyse the solutions so that feedback may be provided.

In order to clarify the concepts related with the proposal, as well as to show its versatility, we have described three different tools that we have developed for three different domains, namely: *Proletool*, for language processors related subjects; *SELFA-pro*, for formal languages and automata theory; and *Electronics*, for electronics circuits. Furthermore, with the aim of showing the viability and validity of our proposal we have presented and commented on some statistics extracted from *Proletool*, which is the most complete of our tools and the most used by the students since 2005.

Together with the description regarding the development and use of these applications, we believe the versatility of our proposal has been proved in this paper. We have also documented the types of scenarios where our proposal can be applied, that is, all those scenarios where problems can be expressed by a formal language and whose domain provides deterministic algorithms to solve them. Of course, the pedagogical validity of the system will depend on multiple end factors, like the usability of the user interface, the availability of proper simulation and visualisation capabilities, etc.

In order to expose the first impression we have obtained from the experience, we have analysed the results showing that students who have used the tool for six academic years think that it is both possible and easy to specify new learning activities statements and that the system is able to provide appropriate and useful feedback for the solutions provided to solve those activities. That is, students have a mechanism to test as many alternatives and variations of problem statements and their respective solutions as desired. In addition, the results show that students who used the tool obtained better scores than those who did not.

Taking this into consideration, we think that it is possible to build systems that allow students to specify problem statements, to introduce alternative solutions for those problems, and to analyse the corresponding evaluation,

that is, the corresponding feedback provided to the students. Thus, students are able to identify their mistakes in both the formalisation of the problem or in the solution. Therefore, they are able to introduce their own changes to the problem statements, so that higher order thinking skills can be achieved.

Thinking in terms of Blooms Taxonomy, this means that the students are working on the highest thinking skills order. They are able to use their knowledge to solve an assignment (application level), to create new assignment statements (synthesis level), and to corroborate the accuracy of their evaluation for a specific solution (evaluation level). Furthermore, the system provides an active learning environment, and allows them to obtain high scores because they are involved in learning activities that require problems to be solved.

Presently, our work is focused on obtaining data pertaining to experimental studies in other domains which have been presented in this paper and these data will contribute to obtaining a more exact conclusion.

References

Aho, A., Lam, M., Sethi, R., Ullman, J., 2006. Compilers: Principles, Techniques, and Tools, 2nd Edition. Addison Wesley.

Ala-Mutka, K., 2005. A survey of automated assessment approaches for programming assignments. Computer Science Education 15 (2), 83–102.

Anderson, L., Krathwohl, D. (Eds.), 2001. A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman.

Bloom, B. S., 1956. Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain. New York: David McKay Co Inc.

Bonwell, C., Eison, J., 1991. Active learning: Creating excitement in the classroom. Tech. rep., The George Washington University, School of Education and Human Development.

Boroni, C. M., Goosey, F. W., Grinder, M. T., Ross, R. J., 1998. A paradigm shift! the internet, the web, browsers, java and the future of computer science education. In: 29th ACM SIGCSE Technical Symposium on Computer Science Education. pp. 145–152.

Bravo, C., Redondo, M. A., Ortega, M., Verdejo, M. F., 2006. Collaborative environments for the learning of design: A model and a case study in domotics. Computers & Education 46(2), 152–173.

Bravo, C., van Joolingen, W. R., de Jong, T., 2009. Using co-lab to build system dynamics models: Students' actions and on-line tutorial advice. Comput. Educ. 53 (2), 243–251.

Brusilovsky, P., Peylo, C., 2003. Adaptive and intelligent web-based educational systems. International Journal of Artificial Intelligence in Education 13, 156–169.

Carrol, J., Long, D., 1989. Theory of Finite Automata with an Introduction to Formal Languages. Prentice Hall.

Castro-Schez, J. J., del Castillo, E., Hortolano, J., Rodriguez, A., 2009. Designing and using software tools for educational purposes: Flat, a case study. IEEE Transactions on Education 52 (1), 66–74.

Castro-Schez, J. J., Garcia-Bermejo, D., Garcia-Bermejo, R., 2011a. Electronics project home page.
URL `http://www.esi.uclm.es/jjcastro/electronics`

Castro-Schez, J. J., Hortolano, J., Rodriguez, A., 2011b. Selfa project home page.
URL `http://portal.esi.uclm.es/selfa/`

Castro-Schez, J. J., Santos, P. A., Santos, J., 2011c. Proletool project home page.
URL `http://portal.esi.uclm.es/proletool/`

Churches, A., 2008. Bloom's taxonomy blooms digitally. Educators' eZine.
URL `:http://www.techlearning.com/article/8670`

Cole, D., Wainwright, R., Schoenefeld, D., 1998. Using java to develop web based tutorials. In: 29th ACM SIGCSE Technical Symposium on Computer Science Education. pp. 92–96.

Cooper, K., Torczon, L., 2011. Engineering a Compiler, 2nd Edition. Morgan Kaufmann.

Devedzic, V., Debenham, J., Popovic, D., 2000. Teaching formal languages by an intelligent tutoring system. J. Educational Technology & Society 3 (2), 36–49.

Dewey, J., 1922. How We Think, A Restatement of the Relation of Reflective Thinking to the Educative Process. D.C. Heath and company, New York.

Eden, H., Einsenberg, M., Fischer, G., Repening, A., 1996. Making learning a part of life. Commun. ACM 39 (4), 40–42.

Fernandes, E., Kumar, A., 2005. A tutor on subprogram implementation. J. Comput. Small Coll. 20 (5), 36–46.

Garcia-Osorio, C., Arnaiz-Moreno, A., Arnaiz-Gonzalez, A., 2011. Thoth project home page.
URL `http://pisuerga.inf.ubu.es/cgosorio/THOTH/`

Gordijn, J., Nijhof, W. J., 2002. Effects of complex feedback on computer-assisted modular instruction. Computers & Education 39 (2), 183–200.

Grune, D., Bal, H., Jacobs, C., Langendoen, K., 2000. Modern Compiler Design. Wiley & Sons.

Harrison, M., 1978. Introduction to Formal Language Theory. Addison-Wesley.

He, Y., Hui, S. C., Quan, T. T., 2009. Automatic summary assessment for intelligent tutoring systems. Computers & Education 53, 890–899.

Hopcroft, J. E., Karp, R. M., 1971. A linear algorithm for testing equivalence of finite automata. Tech. rep., Cornell University.
URL `http://hdl.handle.net/1813/5958`

Hopcroft, J. E., Motwani, R., Ullman, J. D., 2001. Introduction to automata theory, languages and computation. Addison Wesley.

IMS-GLC, 2003. Ims learning design. Online.
URL `http://www.imsglobal.org/learningdesign/index.cfm`

Jurado, F., Molina, A. I., Redondo, M. A., Ortega, M., Giemza, A., Bollen, L., Hoppe, H. U., 2009. Learning to program with coala, a distributed computer assisted environment. Journal of Universal Computer Science 15 (7), 1472–1485.

Jurado, F., Redondo, M. A., Ortega, M., 2012. Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice. Journal of Network and Computer Applications 35 (2), 695–712.

Jurado, F., Santos, O. C., Redondo, M. A., Boticario, J. G., Ortega, M., September 24-26 2008. Providing dynamic instructional adaptation in programming learning. In: Corchado, E., Abraham, A., Pedrycz, W. (Eds.), Proceeding of the Hybrid Artificial Intelligence Systems (HAIS2008). Vol. 5271/2008 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Burgos (Spain), pp. 329–336.

Kelley, D., 1998. Automata and Formal Languages: An Introduction. Prentice Hall.

Koper, R., Tattersall, C. (Eds.), 2005. Learning design: A handbook on modelling and delivering networked education and training. Springer.

Kumar, A., 2004. Using online tutors for learning - what do students think? In: Proceedings of Frontiers in Education Conference (FIE 2004). IEEE, pp. 524–528.

Lewis, H. R., Papadimitriou, C. H., 1997. Elements of theory computation. Prentice Hall.

Makonnen, P., 2000. Do www-based presentations support better (constructiristics) learning in the basics of informatics? In: 33rd Hawaii Int. Conf. System, Sciences.

Martin, J., 2003. Introduction to Languages and Theory of Computation. McGraw-Hill.

McConnell, J. J., 1996. Active learning and its use in computer science. SIGCUE Outlook 24 (1-3), 52–54.

Murray, T., 1999. Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education 10, 98–129.

Murray, T., 2003. An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state ofthe art. Kluwer Publishers, Ch. 17, pp. 491–544.

Norman, D. A., Spohrer, J. C., April 1996. Learner-centered education. Communications of the ACM 39 (4), 24–27.

Oser, F. K., Baeriswyl, F. J., 2001. AERA's Handbook of Research on Teaching, 4th Edition. Washington: American Educational Research Association (AERA), Ch. Choreographies of Teaching: Bridging Instruction to Learning, pp. 1031–1065.

Rahman, K. A., Nordin, M. J., December 2007. A review on the static analysis approach in the automated programming assessment systems. In: National Conference on Programming 07.

Rajala, T., Laakso, M.-J., Kaila, E., Salakoski, T., 2008. Effectiveness of program visualization: A case study with the ville tool. Journal of Information Technology Education: Innovations in Practice 7, 15–32.

Rodger, S. H., Finley, T. W., 2011. Jflap project home page.
URL `www.jflap.org`

Salmela, L., Tarhio, J., 2004. Ace: Automated compiler exercises. In: Proceedings of the 4th Finnish/Baltic Sea Conference on Computer Science Education. pp. 131–135.

Sanders, D., Hartman, J., 1987. Assessing the quality of programs: a topic for the cs2 course. In: SIGCSE '87: Proceedings of the eighteenth SIGCSE technical symposium on Computer science education. ACM, New York, NY, USA, pp. 92–96.

Suraweera, P., Mitrovic, A., 2002. Kermit: A constraint-based tutor for database modeling. In: S. A. Cerri, G. G. . F. P. (Ed.), Intelligent tutoring systems. Berlin: Springer, pp. 671–680.

Tamagnini, J. J., Cavadini, S. V., Berdaguer, P. L., Cheda, D. A., Pachado, F., Petersen, M., 2011. Sepa! project home page.
URL `http://www.ucse.edu.ar/fma/sepa/`

Tran, Q.-N., 2007. Interactive symbolic software for teaching formal languages, automata and beyond. J. Computing Sciences in Colleges 22 (4), 129–136.