The Syntax Student's Companion: an eLearning Tool designed for (Computational) Linguistics Students

Aurélien Max
Groupe d'Etude pour la Traduction Automatique
(GETA-CLIPS)
Grenoble, France
aurelien.max@imag.fr

Abstract

This paper advocates the use of free and easily accessible computer programs in teaching. The motivating reasons for a particular program supporting the learning of syntax are given, and a first version of the program is presented and illustrated. Initial evaluation results led to additional specifications and to the development of a new version of the program that is introduced. Finally, several perspectives for such a support tool are drawn.

1 Introduction

Doing exercises to manipulate the concepts taught in a course is essential to both teachers and students. While the former want to ensure that their students have a good grasp of the material that they teach them, the latter often want to illustrate that material with some concrete practice. Linguistics or computational linguistics students who are introduced to the intricacies of grammar are no less concerned than any others. A typical exercise consists in asking students to analyze a sentence by means of its description as a syntactic tree. In introductory courses, either a context-free grammar is given to them before the exercise begins, or they have to build one of their own that can be used to analyze the sentence given. Obviously, the more exercises look like challenging "games" and the more they are easy to use and accessible, the more likely students are to invest time and effort in trying to do them (see e.g. (van Halteren, 2002; Gibbon and Carson-Berndsen, 1999)). If they spend a lot of time drawing, erasing parts of their trees, drawing them again or correcting them, and then waiting for minutes before their teaching assistant is available again, they may not find the whole exercise very captivating very long. But this type of exercise is essential to understand how the most basic of grammar formalism works and therefore to build a solid ground for the study of language analysis.

Computers play a growing role in education, as the number of workshops dedicated to eLearning and related domains shows. While many institutions experience financial cuts, often reflected in the reduction of the time devoted to supervised work, the use of computer support has also its roots in other reasons. It should be clear that computer tools are not meant to dispense entirely with teachers, but rather to have them concentrate on the pedagogical content. Machines are good at supporting well-defined tasks, and can therefore allow students to practise concepts that have been encoded into a well designed computer program. The issues of what type of practice can be done in a satisfactory manner with computers today and of the extent to which it can actually help students or assess their performance are open to debate and the object of research. Importantly to us, past projects have shown that the computer-assisted learning of syntax can produce a high level of engagement by students (e.g.(Larson, 1996)).

This paper concentrates more on the student's perspective, inspired from the author's own experience as a former computer science student taking courses in linguistics. The first section presents the motivating reasons for the creation of a computer program intended to support the practice of syntax exercises. The program is described and its use is illustrated by concrete examples. Preliminary elements of evaluation are inferred from the use of the program by university students and teachers, showing that this type of support yields promising results in spite of a few issues. We then present our current work by describing the design of a new version of the program, where modularity and extensibility play a central role. It is hoped that this new version will be

more suited to both students' and teachers' needs, and that this practical experience will contribute to the development of the field of computer-assisted learning. We finally propose several tracks for the evolution of this type of tool.

2 Motivating reasons for the creation of the program

Supervised time in university courses tends more to diminish than to augment. It is however particularly crucial in introductory courses that students can get a good grasp of the concepts by regular supervised practice. Exercise sheets are often found useful only if sufficient time in the classroom can be devoted to go throught all the different subtleties encoded in the exercises. It can therefore be advantageous to offer students a means to practise outside of the classroom, while still being able to ask their teachers for help. There are a number of criteria that should be taken into account when designing a computer program for supporting this kind of practice, including the following:

- **The program should be attractive to students**. It is well known in computer engineering that good programs can end up not being used if the user was not taken into consideration from the very beginning of the engineering process. Students are a particular kind of users who may not be willing to use programs that are tedious or overcomplicated to use, and seen as not helpful as a result.
- **Teachers should have the feeling that they can control what the program does.** Not only should it be simple for teachers to add new data conforming to predefined exercise types, but it should also be possible to extend the program.[1]
- **The program should provide useful feedback to students.** While it is probably the case that an asynchronous mode of practice whereby a student would do exercises on a computer and then send the results electronically to a teaching supervisor would yield good results in some contexts, students will expect the program to assess their answers and possibly provide feedback, and therefore support self-study to some extent.
- **The use of the program should be independent from place and time.** It is our own experience as teacher at the university level that a significant proportion of post-2000 students prefer to work from home when they have this possibility. Booking computer rooms for practice for specific courses may work for some students, but certainly not for all of them. This said, supervised sessions with computers may still be a fruitful option. This, of course, further implies that the program should not be too costly for both the university and the students, if not free.

When we first worked on the development of a program that would support the practice of syntax exercises, back in 1999, there were already programs in this area. Trees 2 [2], developed at the University of Pennsylvania, allowed students to visually build syntactic trees, but in such a way that they could only be valid relative to the grammar used. Moreover, at the time the program could only be run locally on Macintosh computers and required the purchase of a licence. Syntactica[3], developed at Sunny Brooks University, allowed students to build grammars and then ask the program to build the syntactic tree for them, which they could subsequently modify. Again, at the time the program only existed for NeXT computers. The free Java applet from the University of Bangor, The Syntax Tutor[4], permitted a student to enter a set of context-free rules and to ask the system to parse a sentence with it.

Except for the case of the The Syntax Tutor, these programs had to be bought, and could only be run on specific computer families. Nevertheless, their existence shows that there was a very promising trend, supported by encouraging evaluation (see e.g. (Larson, 1996; Phillips, 1998)), to offer students computer programs for the study of syntax.

[1] It is not expected that teachers would write computer code themselves, but the program could be extended by means of predefined building bricks or by the addition of code by a computer engineer with access to a clear application programming interface (API).

[2] http://www.ling.upenn.edu/ kroch/Trees.html

[3] http://semlab2.sbs.sunysb.edu/Users/rlarson/Syntactica/syntactica.html

[4] http://www.bangor.ac.uk/ling/java/lt/LingTutor.html. This link has been down for some time.

3 Program design considerations

When designing the program, we had two types of considerations in mind, pedagogical and technical. The basic idea was to let students build syntactic trees in a simple way, and to edit or consult the underlying grammars. What seemed very important was to let the students the possibility to make errors, considering that trial and error, providing appropriate feedback is given, can be part of a sound learning process. Therefore, students should be able to draw syntactic trees that are not valid relative to a given grammar, which was given to them or was build by them, and was accessible and modifiable or hidden. The syntactic theory used would initially be the X' theory[5], and the types of exercises would include the drawing of ambiguous sentences based on some data, and the modification of existing trees to illustrate syntactic transformations.

Technical considerations included the fact that the program should be runnable anywhere and on any computer family. The Java programming language (Sun Microsystems, 1995) was the obvious choice, as it was already quite mature and could be run over the Internet on any platform that had a Java virtual machine. Furthermore, a Java program can exist in two flavors, as an *application* that can be installed and run locally on a personal computer, and as an *applet* that can be downloaded at execution time over the Internet and run by the virtual machine of a web browser installed on computers of a university department without any installation nor maintenance.

Furthermore, exercises and resources for the program had to be modifiable. For a local use with the application version, the user should be able to create new exercises using a simple description language. For a distributed use with the applet version, the administrator of the website where the applet is hosted should be able to add resources that would be immediately accessible to all the remote users. Modifiable resources include grammars, trees, exercise definitions, and language resource files for running the program in the language of the user. XML (W3C, 2000) was chosen as the format for most of the resources, and a simple schema was designed to allow the creation of new resources. It was initially believed that this provided a simple way of creating new resources and modifying existing ones.

[5]This choice was based on a particular introductory course taught at McGill University, which used (O'Grady and Dobrovolsky, 1996) as its coursebook.

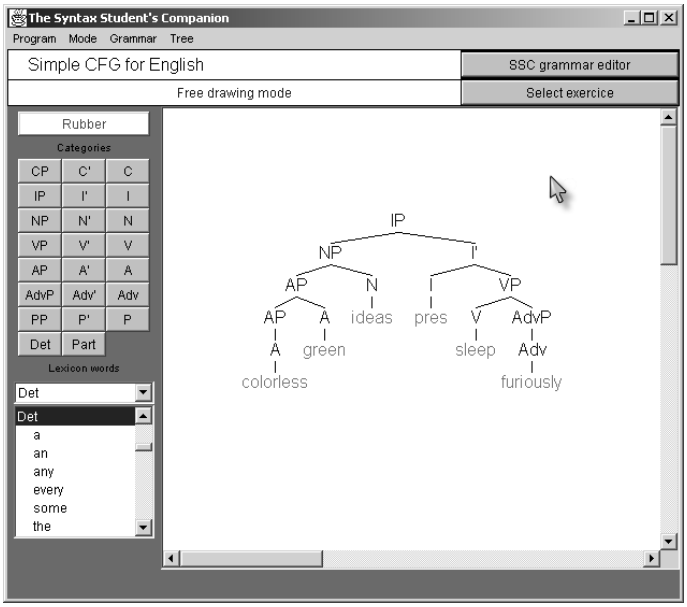

Figure 1: The main window of the program

4 Presentation of the program

Our program is called the *Syntax Student's Companion*. Figure 1 shows its main interface running in English.[6] The top panel contains the active grammar (Simple CFG for English in the example), a button to launch the grammar editor, the active mode (Free drawing mode) and a button to switch to the exercise mode. The panel on the left contains buttons for all the nonterminal and terminal categories of the active grammar, and a list for the words in the lexicon. The main panel is a scrollable zone called the *workspace* where trees can be drawn. Menus contain commands relative to the customization of the program, user modes, grammars, and trees.

Clicking on a syntactic category or on a lexicon word allows dropping it onto the workspace at a chosen location.[7] Trees are built by combining subtrees, as illustrated in

[6]The program can be run in 7 languages thanks to localized resource files contributed by various people.

[7]The Trees program proposes to drop on the workspace subtrees corresponding to partial structures described in the grammar used. We plan to add this feature in the next version of the program, as it allows students to concentrate on more advanced notions.

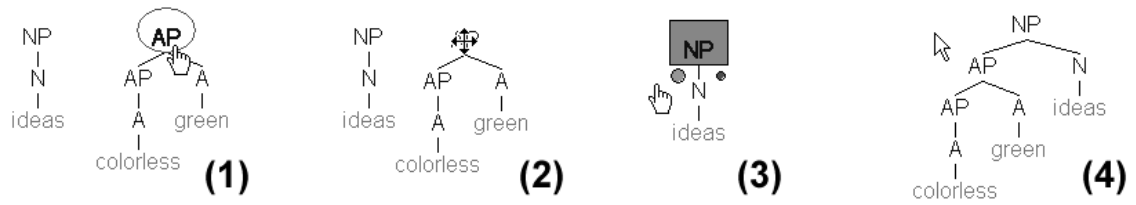

Figure 2: Steps for attaching a subtree to a node

figure 2. First, the root node of the tree that will become a subtree of another tree should be selected with the mouse (1), and dragged onto the node that will become its mother (2). If that node has not any children yet, then the attachment is done. Otherwise, the user has to select the position of the new subtree among the daughters of its mother (3). When the position has been chosen, the attachment is done, and the new layout of the tree is produced (4), so as to ensure that the trees are always well-balanced.[8] Alternatively, categories and words can be directly dropped onto the workspaces as children of existing nodes. Trees or subtrees can be copied and pasted onto the workspace, allowing faster construction. To detach a subtree, the root of the subtree should simply be dragged away from its parent tree. Trees and subtrees can also be removed from the workspace by using the rubber tool.

All these adjunction operations can be done regardless of the rules defined in the active grammar. Therefore, students may make errors and be aware of them only after they try to validate their trees with the active grammar. Indeed, contexts where students could use a tree drawing application with grammars designed in such a way that irrelevant errors were not possible revealed in some cases that the students had become too dependent on the helping hand of the program and were not able to perform as well without it (Phillips, 1998).

The current version only supports simple context-free grammars. Grammars can either come from a remote or a local file, or they can be created from scratch by the student. The grammar editor (see figure 3) allows the

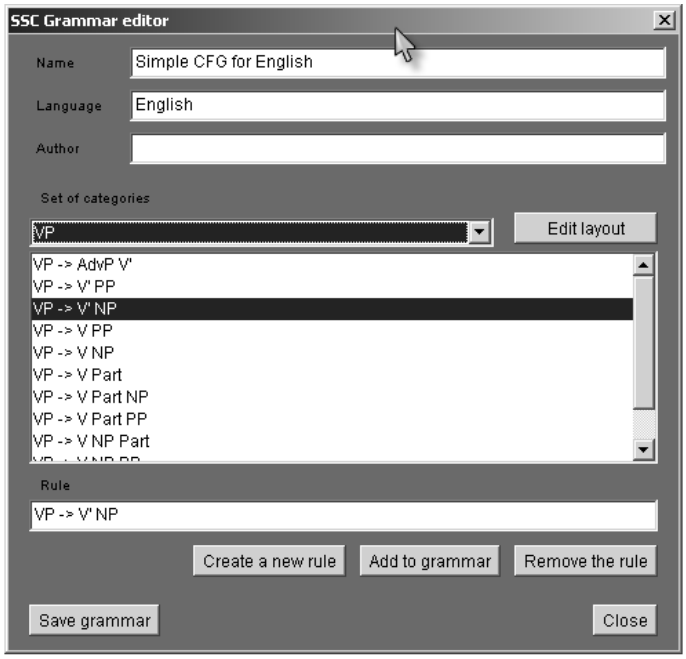

Figure 3: The dialog box of the editor for context-free grammars

consultation and modification of the current grammar. It shows all the derivation rules corresponding to a given nonterminal category[9], and allows specifying of how they are presented on the window of the main interface.

Once students have built trees, they can ask the program to check their validity according to the active grammar. If the active grammar is modifiable, they can modify it so as to ensure that the coverage of the grammar include their trees. If the active grammar is hidden (i.e. not accessible), the validation of their trees indicates whether they conform to an

[8] We are aware that some textbooks use trees with upright lefthand branches and sloping righthand branches, so we will add this possibility as a new parameter. Likewise, we will allow trees to be built bottom up, with all the words of a sentence aligned horizontally.

[9] In the presented implementation, lexical categories appear as just any other nonterminal categories in the grammar editor dialog box, but that may be confusing for students. We therefore think that the lexicon should be distinguished from the grammar itself, as it is done on the left panel of the main interface (see figure 1).

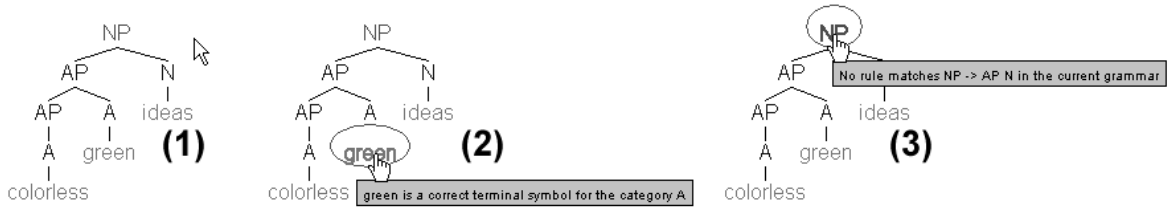

Figure 4: Checking of the validity of a tree relatively to the current grammar

implicit grammar specification (such as one that would have been described during lecture sessions). Tree nodes that violate the rules of the grammar are shown in red, and passing the mouse cursor over them displays a message indicating the nature of the error, as illustrated in figure 4, subfigure (3).

Three modes of exercises have been defined and can be encoded in XML resource files. The drawing of non-ambiguous trees requires the student to draw the tree for a given sentence using a given grammar, whereby the analysis of the sentence is unambiguous. An example of such an exercise encoded into XML format is given in figure 5 for the Spanish phrase *convocatoria de proyectos de innovación educativa*. Figure 6 illustrates the ambiguous tree drawing exercise type. The student is asked to draw the syntactic tree for a sentence (*Time flies like an arrow* in this case) given several data that permit to disambiguate the sentence and find the correct syntactic derivation. The last type of exercise asks students to modify trees (see figure 7) to reflect syntactic transformations. Instead of asking the student to draw the syntactic tree for the sentence (in the example, *Who will come tomorrow?*), she is provided with a base tree (in the example, the tree for the sentence *Bobby-Joe will come tomorrow*), in order to better illustrate the transformations that take place.

5 Initial evaluation

As we are not ourselves involved in syntax teaching[10], we have not been able to perform any formal evaluation of the presented version of the program. It is however crucial to be able to assess the effectiveness of such a tool, both in terms of the type of help it gives to the student and the support it provides to the teacher. The initial evaluation elements we have been able to gather from emails sent to us via the website of the project constitute the

[10]Our initial motivation was to offer such a program to fellow students.

```
<?xml version="1.0"?>

<exercices type="" author="">
  <exercice name="convocatoria de
    proyectos de innovacion
    educativa"
  language="espanol"
  type="Unambiguous tree drawing">
    <sentence>convocatoria de proyectos
      de innovacion educativa</sentence>
    <grammar name="" type="" author="">
      <rules>
        fsust -> nucleo mod;
        ncleo -> sust;
        mod -> fprep;
        fprep -> director termino;
        director -> prep;
        termino -> fsust;
        mod -> adj;
        sust -> convocatoria;
        prep -> de;
        sust -> proyectos;
        sust -> innovacion;
        adj -> educativa;
      </rules>
      <categories_display>
        <row>fsust nucleo</row>
        <row>mod fprep sust</row>
        <row>adj prep termino</row>
        <row>director</row>
      </categories_display>
    </grammar>
  </exercice>
</exercices>
```

Figure 5: Sample exercise definition for unambiguous tree drawing

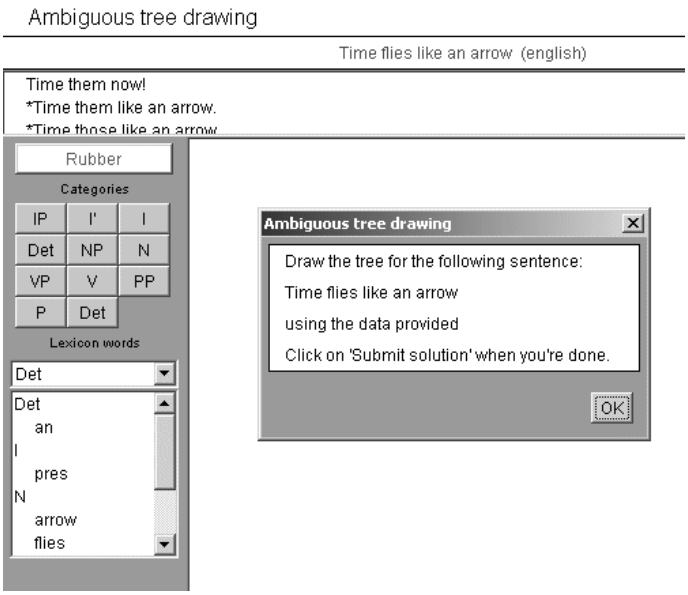

Figure 6: Ambiguous tree drawing exercise

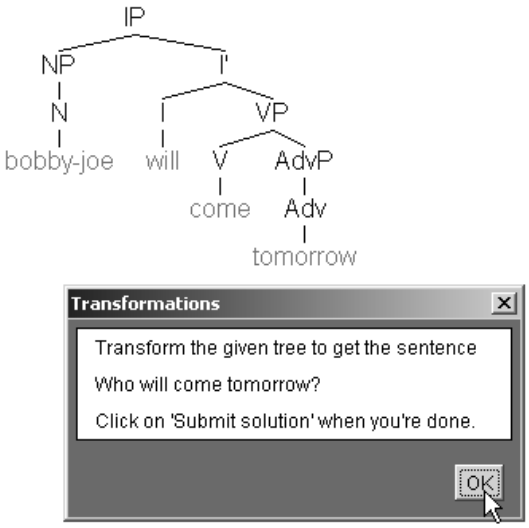

Figure 7: Tree transformation exercise

basis for an updated specification for the new version of the program that we will introduce in the next section.

Several teachers have reported that they had used the program at some point in their teaching, but we suspect that in most cases the program was demonstrated to students (for example, using a data projector in the classroom), hoping that they would use it for self-study. The most important limitation user feedback told us was the difficulty to add new exercises for teachers. Only few people contributed exercises in XML format[11], suggesting that this way of specifying resources was probably not adequate for linguistics teachers. Although the program can support any grammar theory based on context-free grammars, the default grammars made some users think that only the X' theory could be used, and some users had difficulty to see that the grammars could in fact be edited and totally new sets of categories defined. Unsurprisingly, some teachers said they were interested in the support of feature structures.

A not-so-expected use of the program was for producing graphical trees for inclusion into documents. This, corroborated with several user testimonies, seems to indicate that the program is considered easy to use. Its simplicity was in fact often mentionned as one of the preferred characteristics by students who used the program without any prior recommandation from a teacher. We also think that the availability of the program and its online user manual in several languages may have contributed to this.[12]

Some technical issues were also reported. Most users of the program, who are not supposed to be computer scientists, found it difficult to set up the Java program and run it as an application. Moreover, some web browsers did not run the applet perfectly. The existing version of the program is based on the Java technology that existed in 1999, and the language is now more mature and better supported, so it is now simpler to set up a Java virtual machine on one's computer and to run Java programs, and support for Java in web browsers is much better than it used to be.

As regards the evaluation we would like to be able to conduct, we believe that user questionnaires and logging of student activity would be good indicators of its effectiveness. Also, it would be interesting to see if the use

[11] Some people may have written exercises of which we are not aware.

[12] Evaluation results for the Syntactica grammar workbench revealed that the use of this kind of computer-assisted instruction surprisingly increased the need for instructor support (Larson, 1996). We assume that this was partly due first to the number of functions of the program, as well as the fact that at the time linguistics students were for the most part new to the use of computers.

of the program can make significant differences in the evaluation of the performance of student groups.

6 Current work

We have specified a new version of the program that will be partly developed by two Masters students during a computer engineering project. We present the main changes from the existing version in this section, and we conclude with some perspectives in the next section.

First of all, the main lesson we can draw from user feedback is that no matter how much time is spent on specification, not all features that would be useful to users could be imagined. Therefore, it seems a good idea that such a non-commercial program be extensible by other contributors who would like to add new features such as new exercise types, or support for other grammatical theories. The new version will have an OpenSource licence, which implies that we pay a particular attention to the genericity, modularity and documentation of the source code, and that the program will continue to be free to use, which seems essential to us.

A bottleneck to a more widespread use of the program is certainly the difficulty to create new resources, mainly exercises. A particular mode for the definition of exercises will be integrated into the program. This mode will allow a teacher to describe an exercise and its solution in a way as similar as possible to the exercise mode itself. We also want to support the description of possible errors and their appropriate corrections and comments, in order to provide better feedback to students. Once the exercises are defined, it would be possible to submit them to a repository on a web server, on a collaborative mode.[13]

A novel use of the applet version will allow using it inline in web pages, instead of as a separate application window. This will not only allow the dynamic drawing of tree descriptions specified as parameters to the Java applet (and possibly tree animations), but also the insertion of exercises within online course material. We plan to use this for the tutorial of the program.

On the content side, several ideas have been submitted and will be implemented depending on time. Notably, it seems particularly interesting to provide actual linguistic data from corpora to students from which grammars can be inferred, as in (Borin and Dahllof, 1999). A new exercise type will ask students to write a grammar accounting for a given small corpus, which could already be morphologically annotated or not. Lexicons will be separated from grammars, in order to make them reusable when possible. Feature structures will also be supported, both for the edition of grammars and for the validation of syntactic derivations.

A number of new features concern the graphical display of trees. Notably, it will be possible to collapse or expand subtrees (using the triangle notation), and to draw trees top-down with the terminal symbols immediately under the non-terminal that dominates them, or bottom-up with the terminal symbols aligned horizontally.[14] It will also be possible to specify display properties (such as font and color) at the level of nodes and subtrees, and to export trees as bitmap files for easy inclusion into documents like assignments and course notes.

7 Perspectives and conclusions

One could think of many other features that would probably make the program even more useful for learning. We only mention a few and we hope that OpenSource contributions will extend the list.

A key aspect of this kind of support tool certainly lies in the nature of the feedback that is provided to students. We have already said that the mode for defining exercises will allow the teacher to specify possible wrong solutions and to associate them with an appropriate correction. An interesting extension would be a mode where students could send the results of their exercise session (possibly containing a series of coherent exercises) to a supervisor by

[13] Collaborative projects, such as the Papillon project for multilingual lexical resources, show that this approach can work if submitters can also benefit from the submissions of other contributors.

[14] In the latter case, it will be possible to specify that the trees be developed with an upright lefthand branch and sloping righthand ones, as this layout is used in some textbook and is therefore more familiar to students using them.

email from the program. Then, the annotated corrections of exercises could feed a database and be reused in subsequent unsupervised exercises. We think that there is indeed much to be gained from past corrections, as shown in the research on vicarious learning using past dialogues between learners and their teachers (Cox et al., 1999), which, incidentally, was also based on the teaching of syntax.

The range of topics covered by the program could be extended. The learning of syntax could probably be supported by the integration of parsers, which could be of particular interest to computational linguistics students (see e.g. (Meurers et al., 2002; van Halteren, 2002)). The integration of generators would also allow students to inspect the productions of their grammars to attempt to identify why they could overgenerate. Furthermore, we would like to reuse what already exists for the morphological analysis of words in terms of inflections and derivations, as well as for compositional semantic analysis.

The program we have presented puts a particular emphasis on its central users, who are students in (computational) linguistics. Initial evaluation has shown that this kind of support was very welcome by the learners' community, and we hope that it will be more widely adopted by the teachers' community in its new version that attempts to reduce known limitations. We look forward to new developments in the field of research in computer-assisted learning, and in particular on methodologies for the evaluation of systems.

Acknowledgements

Many thanks go to the people who have directly contributed to this unfunded project on a volunteer basis, in particular Séverine Gedzelman and Bénédicte Grizolle for their work on the new version of the program, and Àgnes Sandor, Su-Ying Hsiao, Tanja Hieber, Susana Sotelo Docío, Thierry van Steenberghe, Nicola Cancedda and Christophe Terrasson for their contribution. Many thanks also to Lisa Travis and Nathan Friedman from McGill University, and to all the students and teachers who have sent encouraging feedback on their use of the tool.

References

Lars Borin and Mats Dahllof. 1999. A Corpus-Based Grammar Tutor for Education in Language and Speech Technology. In *Proceedings of the workshop Computer and Internet supported education in language and speech technology, EACL'99, Bergen, Norway.*

Richard Cox, Jean McKendree, Richard Tobin, John Lee, and Terry Mayes. 1999. Vicarious learning from dialogue and discourse. *Journal of Instructional Science*, 27:431–458.

Dafydd Gibbon and Julie Carson-Berndsen. 1999. Web tools for introductory computational linguistics. In *Proceedings of the workshop Computer and Internet supported education in language and speech technology, EACL'99, Bergen, Norway.*

Richard K. Larson. 1996. Grammar as a laboratory science. In *Presented at the American Association for the Advancement of Science Meetings, Special Session "From Curiousity to Science Through Linguistic Inquiry" Baltimore, U.S.A.*

W. Detmar Meurers, Gerald Penn, and Frank Richter. 2002. A Web-based Instructional Platform for Contraint-Based Grammar Formalisms and Parsing. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, Philadelphia, U.S.A*, pages 19–26.

William O'Grady and Michael Dobrovolsky. 1996. *Contemporary Linguistic Analysis.* Copp Clarck, Toronto, 3rd edition.

Colin Phillips. 1998. Teaching Syntax with Trees. *GLOT International*, 3.7.

Sun Microsystems. 1995. The Java programming language. http://www.javasoft.com.

Hans van Halteren. 2002. Teaching NLP/CL through Games: the Case of Parsing. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, Philadelphia, U.S.A*, pages 1–9.

W3C. 2000. XML 1.0: The eXtensible Markup Language (2nd edition). October 2000 W3C recommandation, http://www.w3.org/TR/Rec-xml.