Syntax Trees Visualization in Language Processing Courses

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes

2009 Ninth IEEE International Conference on Advanced Learning Technologies, 597-601

DOI: http://dx.doi.org/10.1109/ICALT.2009.158

2009 Ninth IEEE International Conference on Advanced Learning Technologies

Syntax Trees Visualization in Language Processing Courses

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes
LITE - Laboratory of Information Technologies in Education, Rey Juan Carlos University, Spain
{francisco.almeida,jaime.urquiza}@urjc.es

Abstract

This paper describes the educational tool VAST. We designed VAST to be used in compiler and language processing courses. The current version allows generating and visualizing syntax trees and their construction process. The main advantages of VAST follow: it is designed to be as independent from the parser generator as possible, it allows students to visualize the behaviour of parsers they develop, and it has an interface designed to easily handle huge syntax trees. We describe different ways of using VAST in educational settings as well as a usability evaluation.

1. Introduction

Language Processors (LP) and *Compilers* are often perceived by students as some of the most complex subjects. Typical topics covered by these subjects are: the scanning and parsing phases, syntax directed translation and, if the subject is compilers, symbol tables, semantic analysis, and intermediate and object code generation. The scanning and parsing phases are clearly based on formal languages theory. Syntax directed translation and its use in compilers —semantic analysis and intermediate code generation— do not have such a clear binding with formal languages theory, but they require a clear understanding of the underlying syntax structure.

Automatic parser generators have assisted in reducing the complexity of LP design. These tools are used in educational contexts, but they are professional tools rather than educational tools. Acquiring expertise in these tools is an additional advantage, but they are not easy to learn. Thus, the complexity of LP courses is increased by the use of these tools. Furthermore, there exist many parser generators, but they have not an homogeneous way to specify a LP, the most notable differences being the notation and organization of syntactic and lexical specifications.

The scanning phase exhibits a close relationship between the theoretical foundations and its corresponding generation tools. Consequently, its learning curve is smooth. The case of the parsing phase is different. The relationship between the theoretical foundations –stack automaton and context free grammars– and its corresponding generation tools –that associate actions to grammar rules– seems to be close again. However, there are other topics in the subject, closely related to syntax but without specific support from generator tools, e.g. the error recovery process or the syntax trees (ST). The former require the assistance of an expert on the tool; the latter is not supported by these tools. Notice that the understanding of STs generated and their building process is very important for the most complex part of the subject, namely syntax directed translation.

In this paper we present VAST, an educational system designed to visualize STs. With VAST, students are capable to watch the ST generated by their own parsers, filling a gap between theory and practice of LP design. This is not a novel approach, but existing solutions are partial or too specific. VAST solves this problem from a more sound and generic approach.

The rest of the article is structured as follows. In the section 2 we describe related work. In section 3 we explain the implementation of VAST. Then, we show the different visualizations generated by VAST in section 4. In sections 5 and 6 we describe its educational use and a usability evaluation. Finally, we state our conclusions and future work in the section 7.

2. Related work

As we have previously mentioned, there exist tools focused on filling the gap between theory and practice of syntax analysis, but they do it in a partial or too particular way. We survey these tools in this section.

On one side of the gap we have found the educational tool JFlap [16]. This tool represents a valid approach for the theoretical foundations, even its design has a clear educational aim. However, this system does not allow the students to generate their own parsers.

On the other side of the gap, we have found tools that visualize the matching process from a more practical point of view. We have found nine tools. Four of them [2,7,14,18] do not allow to generate the parser.

978-0-7695-3711-5/09 $25.00 © 2009 IEEE
DOI 10.1109/ICALT.2009.158

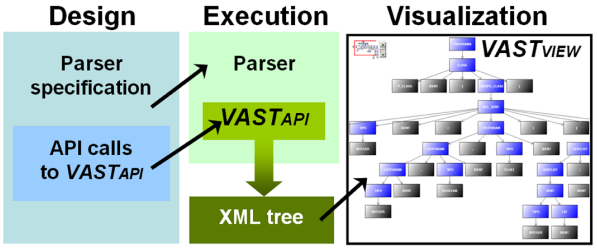

Figure 1. Structure and use of VAST

Another tool, VCOCO [15], allows to generate user specified parsers, it uses the tool COCO/R [12]. But it represents a debug approach for experts rather than an educational tool for students.

The last four tools [3,6,8,9] give more elaborated visualizations and allow to generate user specified parsers. Only one of them, CUPV, does not visualize the ST. However, all of them are highly dependent from an own notation or the generation system used. This approach restricts its educational use because the parser and the visualization system are totally dependent.

There is not any tool that covers all the parsing algorithms and visualizes all the dimensions –algorithmic behavior and ST–, therefore it is possible that a teacher has to use more than one, switching between different notations, organizations and visualizations. In this context, the students have to learn how to use different tools: specification notation, construction process, interpretation of output messages –conflict reports, transitions matrix or items sets–. Furthermore, the teacher has to dedicate time to become familiar with the different environments, and to plan their integration in the course. This could make more difficult their use in educational environments [13].

Finally, we have not found any tool covering the sintactic error recovery. We think that this topic is complex enough to require some visual support.

Therefore, we will focus our efforts on filling the gap between theory and practice. We will visualize the ST and its building process with a visual interface specially designed for this task. We will keep as independent as possible the parser specification and the ST visualization.

3. The implementation of VAST

Our main concern for the implementation of VAST was to separate the ST visualization and its building process. To this aim, VAST offers an API, VASTapi designed to be used when the parser is building the ST, and a graphical interface, VASTview to visualize the ST generated. VASTapi translates grammar rule application events to XML-based information that will be displayed by VASTview. This structure ensures a high degree of independence between VAST and the parser generator used. See Figure 1 for a schematic view of the design of VAST and its use.

4. Working with VAST

In this section, we explain the basic use of VAST. First, the user has to annotate her/his parser specification using methods from VASTapi. Once the parser has been generated, its execution will produce the information used by VASTview to visualize the ST and animate its construction process.

4.1. Generating the visualizations with VAST

The main aim of VAST is to allow the generation and manipulation of the ST independently from the parser generator. Due to the API calls, the execution of the parser generates an intermediate XML based representation of the ST. This XML information is the data source of VASTview.

VASTapi require some information to work properly, this information must be provided by the user. API calls are inserted as part of the typical actions associated to grammar rules. The information needed by the API is just the grammar rule applied –using the method `addProduction`– and the axiom, reduction or derivation, with the `setRoot` method. Now VASTapi can build the XML intermediate representation of the ST that will be visualized by VASTview.

4.1. Visualizing the ST with VASTview

The graphical representation of the ST is the hierarchical structure resulting from the grammar rules application, which is a tree. We allow to give different representations to terminal nodes (T), non-terminal nodes (NT) and error nodes. The *T* nodes are the leaves of the tree, the *NT* nodes are internal ones, and error nodes represent a place where the parser has recovered from a syntactic error. The error nodes only appear if the user has included error recovery inside the parser specification. With the visualization of the error nodes, the user can see the exact error recovery place and the amount of the input stream correctly processed.

We have designed VAST thinking about parsers designed by students. Probably, the ST produced by these parsers are big and without any fixed structure (symmetry, with or height). Therefore, we have developed VASTview, a graphical interface specially

designed to cope with such trees. This interface is made up of a global view and a detailed view of the ST, together with zoom actions, sub-tree aggregation and animation of the construction process of the ST, see the Figure 2.

The global and detailed views allow the user to easily manipulate the ST. The global view shows the whole ST highlighting its visible part in detailed view. The detailed view facilitates give the students a closer inspection of the ST with zooming, aggregating and scrolling, all of them synchronized with the global view. Zooming actions on the detailed view allow the students to focus their attention on specific parts of the ST adjusting the desired level of detail. Sub-tree aggregation –by means of expand/collapse actions– allow the students to maintain a representation of the ST where only interesting parts of it are visible. Finally, scrolling allows students to watch every node in the tree changing the portion of the tree visible in the detailed view. Scrolling can be performed with the scroll bars of the detailed view and directly moving the highlighted area in the global view.

4.1. Animating the construction of the ST

We animate the construction process of the ST using the different intermediate stages. The animation of the construction process help students to see how the input stream is matched by means of shifts –terminal node creation– and reductions –connection of existing nodes with a new non-terminal node–, together with the error recovery. Playing an animation is as easy as using typical VCR controls, together with a slide bar allowing fast location of specific stages of the matching process.

The ST changes its shape, area and contents during its construction process. Thus the interface could adapt to each stage using a best-fit policy changing the location of the nodes and other properties of the graphical representation. We have decided to maintain these properties unless the student changes them. All the nodes keep their location from their creation to the end of the process. This prevents students from distracting, e.g. while searching for the new location of existing nodes.

5. Educational use of VAST

VAST allows users –teachers and students– to view and manipulate a ST and its construction process. From the teacher's point of view, VAST can be used as a

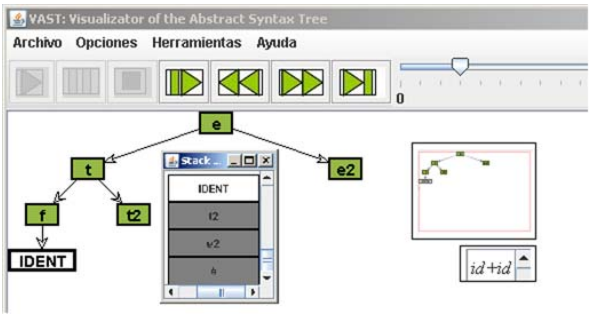

Figure 2. The VASTview interface

demonstration tool for classroom sessions.

Due to its independence from the parser generator, it can be used with all typical parsers in the curricula, brute force, top-down or bottom-up. Thus, the effort dedicated by the teacher to learn how to build visualizations with VAST is applicable to all the course. Also, graphical representations and the interface of ST is the same during the course, preventing the students from learning new ones for each different parsing technique, which is the current situation.

From the students' point of view, in addition to see the behavior of parsers developed by the teacher, they can develop their own visualizations and test the behavior of their own developed parsers. In a related field, literature about algorithm visualization with educational aims [4] has shown that active use of visualizations by students improves their learning process. Thus, visualizations become a part of the student educational experience rather than the main element of this experience. The following section describes an evaluation of VAST in this context.

6. Evaluation of VAST

VAST has passed both an heuristic and an informal observational evaluation. Before conducting an educational evaluation, we want to formally test the usability of VAST. In this section we describe this evaluation, but a detailed report is available at [1].

6.1. Subjects, tasks and experimental method

59 students took part in the evaluation; the participation was incentive-based. They were enrolled in a languages processors course. As the educational design is made up of active tasks with visualization technologies we surveyed the subjects' learning styles, they were mostly active/sensing/visual learners.

The tasks were three exercises about LL(1) parsing where the solution had to be an electronic document with the result of the exercise, together with

visualizations and textual explanations supporting that result. The first exercise asked the students to modify a given grammar of arithmetic expressions so the operators precedence is changed. The second exercise asked students to produce the typical LL(1) syntax errors, namely starting symbols and expected terminals. The third exercise asked students about the panic mode error recovery strategy, they were given how the error recovery must occur and were asked to produce the associated syntax error situation.

We designed this evaluation as a controlled experiment plus an observational study [10]. Also, as we are testing an educational tool we have used some features of true experimental educational studies [11]. We divided students in two balanced groups using a knowledge test. Thus, we decided to use two different visualization tools VAST (the treatment group) and ANTLRWorks (the control group). ANTLRWorks is a visualization extension of the well-known parser generator ANTLR (http://www.antlr.org/). Each group solved the same tasks using the assigned tool. After completing the tasks, we surveyed the students' opinion with a questionnaire. The aspects considered in this questionnaire were: ease of use, learning improvement, quality of the tools, students' satisfaction and personal opinion.

The main aim of this evaluation is not a comparison between VAST and ANTLRWorks. We want to test the usability of VAST, but also we investigate useful features of a professional tool like ANTLRWorks in an educational context.

6.2. Results of the evaluation

During the experiment, instructors observed how the students worked with the tools. Due to some problems with the computers of the lab eleven students abandoned the evaluation. Instructors observed that students in treatment group were enthusiastic with VASTview, but got some confused with the variety of windows simultaneously open: VASTview, editor for grammar and input stream and console. Students in the control group liked the grammar editor of ANTLRWorks but got confused with some compulsory selections –line ending platform, starting rule– and the difference between the interpreter and the debugger.

We compared students' opinions about ANTLRWorks and VAST using the Mann-Whitney test. Most of them were quite similar ($p>.05$) and around 4 (in a five values likert scale). We just detected differences in opinion about the support for learning the stack behavior, the ease of use and the students' satisfaction, see Table 1 for details.

Table 1. The analysis of students' opinion

Student's opinion	ANTLR Works	VASTapi/ VASTview	Significant differences
Learn stack	2.38	3.46 (view)	$U=163.5$, $p<.05$
Ease of use	4.36	3.15 (api)	$U=92.5$, $p<.05$
Satisfaction	4.09	3.45 (api)	$U=138.5$, $p<.05$

7. Conclusions and future work

We have presented the educational tool VAST, aimed at the visualization of syntax trees. We have identified a large gap existing between concepts taught in theory and generation tools used in practice. We feel that this gap can be filled by the visualization of STs and their construction process. Moreover, visualization of STs may assist in learning/teaching syntax directed translation. We have surveyed relevant, related tools. Tools that allow users to generate their own parsers either demand high expertise or are tightly coupled to a given environment. Actually, every tool has its own way to specify the parser, report errors or show transition tables.

We have created VAST to solve these problems. Visualizing a ST and its construction process is almost independent from the parser generator adopted. Thus, a teacher can choose a parser generator based on her/his own criteria –parsing algorithm, specification format– and then use VAST to visualize STs. We want to highlight that VAST was developed so that two parts are isolated: the generation API (VASTapi) and the visual interface (VASTview). VASTapi was designed to build STs, its output being an XML file. VASTview interprets such an XML file to visualize the ST and its construction process. Therefore we have two independence levels: one between the parser generator and VASTapi, and the other between VASTapi and VASTview. At the moment we have developed VASTapi with Java, so we are not totally independent from the parser generator. However, just porting our API to other language will enable to use VAST in other development environments.

We have evaluated the usability of VAST in a real-use environment. Results for the visualization interface, VASTview, are positive. In general, the students are satisfied with VASTview, also they think that: it is easy to use, it supports them in the learning process and it has a good quality. Also we observed that students liked the visualization and animation capabilities of VASTview. ANTLRWorks obtained similar results. But we observed that students got confused because of some professional features as choosing the end of line platform and the starting grammar rule, or having two different visualization tools –the interpreter and the debugger–.

Students' opinion about VASTapi is two-fold. On the one hand, students think that the API is quite

simple. On the other hand, the grades for ease of use and students' satisfaction are worse than those for VASTview and ANTLRWorks.

Our future lines of work are based on these results. They give us hints about how a syntax tree visualization tool with educational aims should be designed. On the one hand, the visualization interface is suitable for students avoiding advanced professional features. But it should incorporate other features as the grammar specification, textual explanations of the different actions performed and advanced navigation through the parsing process, allowing students to select pieces of the input stream and showing the corresponding state in the parsing process.

On the other hand, we have detected that the generation process of visualizations is made up of many separate steps: grammar edition, grammar annotation, parser generation, parser compilation, input stream edition, parser execution and visualization. Although they should not be a problem for the teachers, their integration will improve the interaction of students with the tool. Thus we plan a global integration based on two functional integrations: annotation-generation-compilation and execution-visualization. The former will automatically annotate grammar specifications, generate the parser source code and compile it. The later will allow students to edit the input stream, execute the parser and visualize the ST using the same interface. Thus, parser visualizations will adapt to the typical parser development process of specification, generation and execution.

Automatic annotation will be reached with specific developments for each parser generator. From the students' point of view, making the annotation step transparent to students is much more important than loosing parser generator independence. From the teachers' point of view, we keep independent from the parser generator, because they can still annotate manually parser specifications. Finally, we will extend VAST to support visualization of syntax directed translation concepts.

8. Acknowledgements

This project is supported by project TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.

9. References

[1] F.J. Almeida-Martínez, and J. Urquiza-Fuentes "Teaching LL(1) parsers with VAST – A usability evaluation", *DLSI-I Technical Report 2009-01*, Dept. Of Computer Science Languages and Systems, Rey Juan Carlos University, Spain, 2009, pp. 1-14. http://www.dlsi1.etsii.urjc.es/doc/DLSI1-URJC_2009-01.pdf

[2] Andrews, K., Henry, R.R, and Yamamoto, W.K.: "Design and implementation of the UW illustrated compiler"; *SIGPLAN* Not. 23, 7 (June 1988), 105-114.

[3] Bovet, J.:"ANTLRWorks: The ANTLR GUI development environment". (2009) http://www.antlr.org/works/index.html

[4] Hundhausen, C., Douglas, S. and Stasko, J.: A meta-study of algorithm visualization effectiveness"; *J. of Vis. Lang. and Comp*. 13, 3 (June 2002), 259-290.

[5] Hudson, S., Flannery, F. and Ananian, C.S.: "Cup LALR parser generator for java". (2008) http://www2.cs.tum.edu/projects/cup/

[6] Kaplan, A. and Shoup, D.: "CUPV a visualization tool for generated parsers"; *SIGCSE* Bull 32, 1 (March 2000), 11-15.

[7] Khuri, S. and Sugono, Y.: "Animating parsing algorithms"; *SIGCSE* Bull. 30, 1 (March 1998), 232-

[8] Lovato, M.E. and Kleyn, M.F.: "Parser visualizations for developing grammars with Yacc"; *SIGCSE* Bull 27, 1 (March 1995), 345-349.

[9] Mernik, M. and Zumer, V.: "An educational tool for teaching compiler construction"; IEEE T. Educ. 46, 1 (Feb 2003), 61-68.

[10] O. Kulyk, R. Kosara, J. Urquiza-Fuentes & I. Wassinki "Human-Centered Aspects". In: A. Kerren, A. Ebert & J. Meyer (eds.) *Human-Centered Visualization Environments*. Springer-Verlag, Germany, 2007, pp. 13-75.

[11]. L. Cohen, L. Manion and K. Morrison *Research Methods in Education*. Routledge Falmer, USA, 2001.

[12] Mössenböck, H.: "A generator for production quality compilers"; *Proc. Intl. W. Compiler Compilers CC'90*, Lec. Notes Comp. Sci. 477, Springer-Verlag, New York (1990).

[13] Naps, T., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velázquez-Iturbide, J.: "Iticse 2002 working group report: Exploring the role of visualization and engagement in computer science education"; SIGCSE Bull. 35, 2 (June 2002), 131-152.

[14] Resler, D.: "VisiCLANG–a visible compiler for CLANG"; *SIGPLAN* Not. 25, 8 (August 1990), 120-123.

[15] Resler, R.D. and Deaver, D.~M.: "VCOCO: a visualization tool for teaching compilers"; SIGCSE Bull. 30, 3 (September 1998), 199-202.

[16] Rodger, S.: "Learning automata and formal languages interactively with JFLAP"; SIGCSE Bull. 38, 3 (September 2006), 360-360.

[18] Vegdahl, S.R.: "Using visualization tools to teach compiler design"; *J. Comput.* Small Coll. 16, 2 (January 2001), 72-83.