VAST: a visualization-based educational tool for language processors courses

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes y J. Ángel Velázquez-Iturbide

ITiCSE 2009 - 14th annual ACM SIGCSE conference on Innovation and technology in computer science education, 342.

DOI: http://doi.acm.org/10.1145/1562877.1562983

VAST - A Visualization-based Educational Tool for Language Processors Courses

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes, J. Ángel Velázquez-Iturbide
Departamento de Lenguajes y Sistemas Informáticos I
Escuela Superior de Ingeniería Informática, Universidad Rey Juan Carlos
C/ Tulipán s/n, 28933 Móstoles (Madrid), Spain
{francisco.almeida,jaime.urquiza,angel.velazquez}@urjc.es

ABSTRACT

In this demonstration we present VAST, a visualization tool to support teaching language processors. On the one hand, VAST provides an API that allows generating visualizations of syntax trees independently of the parser generator. On the other hand, VAST provides a GUI with multiple views: the source code, the stack and the syntax tree.

Categories and Subject Descriptors

D.3.4 [**Processors**]: Compilers; K.3.1 [**Computer Uses in Education**]: Computer-assisted instruction (CAI); K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Human Factors, Languages

Keywords

Syntax trees, Visualization

1. INTRODUCTION

Many concepts of a typical language processors course are based on the formal languages theory. When students are writing a syntax directed translator, they must think about the construction process of the corresponding syntax tree.

Teaching parsers is usually supported by visualization tools drawing the syntax tree. But most of these tools are focused on the formal language theory, e.g. JFlap [4]. Parser generators are useful in the language processor courses. But if one wants to visualize the syntax tree structure, she must choose between: develop her own visualization, or use a parser generator with a built-in visualization system e.g. [1, 2, 3].

2. USE AND STRUCTURE OF VAST

The main objective of VAST is allow generating and manipulating syntax trees independent from the parser generator. Adapting parser specifications to generate the visualizations of the syntax trees is easy with VAST. There exist many parser generators, but all of them have the same structure, semantic rules associated to grammar rules. VAST provides an API (VASTAPI) that can be used from the semantic rules. Thus, its use is only limited by the implementation language, which is Java. Students only need to include in the semantic rules associated to each grammar rule an API call like `xxx.addProduction("LHS","RHS");` and fix the axiom production with an API call like `xxx.setRoot();`.

Then the students generate the parser using the parser specification annotated with the VASTAPI calls. Next, the syntax tree visualization data is produced as a side effect of the parser execution.

Finally, the syntax tree visualization can be manipulated with VASTVIEW, the GUI supplied by VAST. VASTVIEW provides three synchronized views: the input stream, the stack and the syntax tree. The visualization can be static –showing a specific processing state– or dynamic –playing the syntax tree construction process–. The graphical properties of the visualization can be customized with a configuration utility. Usually, the syntax trees produced in language processor projects are huge. We have designed VASTVIEW to cope with this problem. The syntax tree view offers both a global and a zoomed view. Students can navigate through the tree using both views. The global view highlights the current part of the tree visualized in the zoomed view, and the zoom factor is controlled by the students. Furthermore, we allow the students to hide/show subtrees adapting the visualization to their needs.

VAST is available at: http://www.lite.etsii.urjc.es/vast/.

3. ACKNOWLEDGMENTS

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

4. REFERENCES

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

[2] A. Kaplan and D. Shoup. CUPV–a visualization tool for generated parsers. In *SIGCSE '00: Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 11–15, New York, NY, USA, 2000. ACM.

[3] M. Mernik and V. Zumer. An educational tool for teaching compiler construction. *IEEE Transactions on Education*, 46(1):61–68, Feb. 2003.

[4] S. Rodger. Learning automata and formal languages interactively with JFLAP. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 360–360, New York, NY, USA, 2006. ACM.

ITiCSE'09, July 6–9, 2009, Paris, France.
ACM 978-1-60558-381-5/09/07.