Parsing @ IDE

Vadim Zaytsev
Raincode Labs
Brussels, Belgium
vadim@grammarware.net

ACM Reference format:
Vadim Zaytsev. 2017. Parsing @ IDE. In *Proceedings of the Fifth Annual Workshop on Parsing Programming Languages, Vancouver, Canada, October 2017 (Parsing@SLE'17)*, 1 pages.

Position Presentation Proposal

It is widely known that most grammars are domain-specific, in a sense that they are created for a narrow purpose: to parse programs that are to be executed, or to analyse specific parts of programs, or to document the structure of a language, or to guide a pretty-printer, etc. One commonly overlooked purpose is IDE support, which would be nice to discuss at the workshop.

Typical IDE-supported features include: syntax highlighting of otherwise monotone text, word selection for scope visualisation, code folding for hierarchical program blocks, visual editing of naturally graphic elements, debugging executable programs, discovering and running tests, performing dependency analysis, suggesting refactorings, displaying violations of coding conventions, providing code navigation to allow programmers to quickly jump between definitions and uses or to follow a call trace, configuring a build, displaying tooltips with documentation, and many others. Some of them are possible to implement based on a parse tree or an AST, but many fall into one or more of the following problematic categories: (1) need to work on partially incorrect programs (e.g., code completion); (2) must work significantly faster than a complete parser (e.g., syntax highlighting); (3) have no sufficiently advanced parser available or require noticeably more information than the parser provides (e.g., detecting missing dependencies).

Most research done on this topic is limited to getting basic IDE support like syntax highlighting by either tweaking a grammar by adding ad hoc manually written code (e.g., to assign colours and implement name suggestion strategies [4]) or by enhancing the grammar with annotations that carry enough information for the underlying universal algorithms to work (e.g., to recover from errors [2, 3]). For getting faster towards a sufficiently detailed parse result, in the industry it is common (see SublimeText, TextMate, Cloud9, MakePad, CodeMirror, Raincode, ...) to use ad hoc combinations of simplistic parsing algorithms (e.g., recursive descent or parsing expression grammars) and regular expressions to perform some form of approximate/island/robust parsing collectively known as "semiparsing" [6]. For some families of languages even their representation in a "grammar" is still in its infancy, which is the case for at least spreadsheet-based [1] and pattern languages [7]. Beyond all that, it is still true that "support for debugging and testing a program written in a DSL is often nonexistent" [5].

In practical language/compiler development, IDE integration is an important part of DSL deployment and is often crucial to gain clients' acceptance. What exactly are all the properties specific or even exclusive to IDE grammars; how to address the challenges of rapid language prototyping; which methods to use to create coarse-grained IDE-specific language definitions and to refine them incrementally to support more advanced features; is still unknown and demands further work and investigation.

References

[1] Sorin Adam and Ulrik Pagh Schultz. 2015. Towards Tool Support for Spreadsheet-based Domain-specific Languages. In *Proceedings of the 14th International Conference on Generative Programming: Concepts and Experiences (GPCE)*. ACM, 95–98. https://doi.org/10.1145/2814204.2814215

[2] Maartje de Jonge, Lennart C. L. Kats, Eelco Visser, and Emma Söderberg. 2012. Natural and Flexible Error Recovery for Generated Modular Language Environments. *ACM Transactions on Programming Languages and Systems (ToPLaS)* 34, 4, Article 15 (Dec. 2012), 50 pages. https://doi.org/10.1145/2400676.2400678

[3] Lennart C. L. Kats, Maartje de Jonge, Emma Nilsson-Nyman, and Eelco Visser. 2009. Providing Rapid Feedback in Generated Modular Language Environments: Adding Error Recovery to Scannerless Generalized-LR Parsing. In *Proceedings of the 24th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM, 445–464. https://doi.org/10.1145/1640089.1640122

[4] Federico Tomasetti. 2017. Kanvas: generating a simple IDE from your ANTLR grammar. Online. (2017). https://tomassetti.me/kanvas-generating-simple-ide-antlr-grammar/

[5] Hui Wu and Jeff Gray. 2005. Automated Generation of Testing Tools for Domain-specific Languages. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 436–439. https://doi.org/10.1145/1101908.1101993

[6] Vadim Zaytsev. 2014. Formal Foundations for Semi-parsing. In *Proceedings of the Software Evolution Week (IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering), Early Research Achievements Track (CSMR-WCRE 2014 ERA)*, Serge Demeyer, Dave Binkley, and Filippo Ricca (Eds.). IEEE, 313–317. https://doi.org/10.1109/CSMR-WCRE.2014.6747184

[7] Vadim Zaytsev. 2017. Parser Generation by Example for Legacy Pattern Languages. In *Proceedings of the 16th International Conference on Generative Programming: Concepts and Experience (GPCE)*, Matthew Flatt and Sebastian Erdweg (Eds.). ACM, 212–218. https://doi.org/10.1145/3136040.3136058