Reverse Engineering Grammar Relationships

Ralf Lämmel and Vadim Zaytsev

Software Languages Team
The University of Koblenz-Landau
Germany

zaytsev@uni-koblenz.de

1 Grammar consistency checking

Many software languages (and programming languages, in particular) are described simultaneously by multiple grammars that are found in different software artifacts. For instance, one grammar may reside in a language specification; another grammar may be encoded in a parser specification; yet another grammar may be present in an XML schema for tool-independent data exchange. Ideally, one would want to reliably establish and continuously maintain that all co-existing (potentially embedded) grammars describe the same intended language. Without such guarantee, grammar inconsistencies may go unnoticed, and grammar-based software artifacts may get brittle. Some ad hoc or brute force methods exist to address this problem, but ultimately grammar consistency checking is an open software engineering problem without a satisfying best practice.

2 Obstacles for consistency checking

For simplicity, let us assume that all the grammars of interest for the given language are BNFs. If we wanted to automatically establish or maintain that some BNFs describe the same language, then we face the problem of grammar equivalence, which, in general, is formally undecidable. Such a formal limit is certainly a part of the problem that there is no best practice for grammar consistency checking.

Obviously, the problem becomes even more challenging once we consider the practical situation of grammars of many different forms: BNFs, parser descriptions, XML schemata, software models, etc. Such variation implies impedance mismatches. As a result, it may be hard to mentally or automatically map one grammar to the other.

3 Grammar differences

Grammars for the same language may be different for various, practically viable reasons. For instance, grammars may be tailored for a certain purpose or quality such as readability. Some grammars may have been designed independently of one another, and hence they are likely to be vastly different in the sense of structural equality of the grammar specifications. Other grammars may have been affected heavily by compromises required by implementation technologies (e.g., parsing techniques), or data models (e.g., XML Schema as opposed to BNF). To summarize, in practice, there are many complex accidental, superficial, and idiosyncratic differences between co-existing grammars of a language.

4 Language evolution

The challenges of grammar understanding and grammar consistency checking are intensified by software evolution. Both software languages as such (e.g., in the form of language documentation) and grammar-based software artifacts (e.g., compilers, source code analysis tools, IDEs) are subject to possibly independent evolution. The grammars of different versions are not even intended to describe the same language, but one would still want to understand their relative correspondence in terms of a "language delta". As a result, there are even more grammars to be checked for consistency. Also, we are no longer restricted to plain grammar equivalence, but language extensions, restrictions, or revisions would need to be captured and checked. Hence, we need a generalized form of grammar consistency checking that can also account for deltas.

5 Grammar convergence

In [4], we have begun to address the fundamental problem of grammar diversity by initiating a method for *grammar convergence*. This method combines *grammar extraction* (to obtain raw grammars from artifacts and represent them uniformly), *grammar comparison* (to determine nominal and structural differences between given grammars), and *grammar transformation* (to represent the relationships between given grammars by transformations that make the grammars structurally equal). Grammar convergence is another method of grammar engineering — as such, it is a companion of grammar recovery, adaptation, and inference.

The specific property of convergence is that an indirect path is taken when several grammars are transformed and only then related to one another by reverse engineering the transformations—as opposed to any process that starts from a single grammar, or any process that can reverse engineer the research target directly, or any process that can match software artifacts with one another directly.

6 The JLS study

In [5] we have described a completed, major study for grammar convergence, and refined the method to provide better scalability and reproducibility. The study concerned the 3 different versions of the Java Language Specification (JLS) [1, 2, 3]. Each of the 3 JLS versions contains 2 grammars: one grammar is said to be optimized for readability, and the other one is intended as a basis for implementation. The earlier assumption of all grammars being BNFs holds here.

One would expect that the different grammars per version are essentially equivalent in terms of the generated language. As a concession to practicality, implementability in particular, one grammar may be more permissive than the other. One would also expect that the grammars for the different versions generate languages that engage in an inclusion ordering because of the backwards-compatible evolution of the Java language. *Those expected relationships of (liberal) equivalence and inclusion ordering are significantly violated by the JLS grammars, as our study shows.*

The JLS is critical to the Java platform — it is a foundation for compilers, code generators, pretty-printers, IDEs, source code analysis and manipulation tools and other grammarware for the Java language. The JLS is the authoritative specification of Java. Hence, there is a strong incentive for an unambiguous, consistent and understandable set of JLS documents. Still, our study discovers substantial inconsistencies with the help of grammar convergence.

7 Contributions

The motivation of our work and its significance is not limited to the mere discovery of bugs in the Java standard or in any other set of grammars for that matter. (In fact, some JLS bugs have been discovered, time and again, by means of informal grammar inspection or other brute force methods.) The significance of our work is amplified by two arguments. First, we provide a simple and mechanized process for accidental or intended differences between grammars. Second, we are able to represent the differences in a precise, operational and accessible manner — by means of grammar transformations. In different terms, we are practically able to prove (or disprove) the equivalence of two given grammars. Our method allows for grammar diversity while supporting grammar consistency checking.

Nontrivial relationships between grammars of industrial size were recovered. (That is, we show that the grammars are equivalent modulo well-defined transformations.)

A mechanized, measurable and reproducible process for grammar convergence has been designed. The process consists of well-defined phases and its progress can be effectively tracked in terms of the numbers of nominal and structural differences between the grammars at hand.

A comprehensive operator suite for grammar transformation has been worked out, driven by the scale of the study. The suite substantially improves on previous attempts.

The complete JLS effort: the involved sources, transformations, results, and tools is publicly available through SourceForge (`http://slps.sf.net`).

The approach, as it stands, faces a *productivity problem.* The transformation part of grammar convergence requires substantial effort by the grammar engineer to actually map any given grammar difference into a (short) sequence of applications of operators for grammar transformation. For instance, the JLS transformations required several weeks of work. Such costs may be prohibitive for widespread adoption of grammar convergence.

Notable productivity gains can be expected from advanced tool support. We currently rely on basic batch execution of the transformations. Instead, the transformations could be done interactively and incrementally with good integration for grammar comparison, transformation and error diagnosis. Ultimately, we need to provide inference of relationships (in fact, transformations). Such inference is a challenging problem because the convergence process involves elements of choice that we need to better understand before we expect reasonable results.

References

[1] J. Gosling, B. Joy, and G. L. Steele. *The Java Language Specification.* Addison-Wesley, 1996. Available at `java.sun.com/docs/books/jls`.

[2] J. Gosling, B. Joy, G. L. Steele, and G. Bracha. *The Java Language Specification.* Addison-Wesley, second edition, 2000. Available at `java.sun.com/docs/books/jls`.

[3] J. Gosling, B. Joy, G. L. Steele, and G. Bracha. *The Java Language Specification.* Addison-Wesley, third edition, 2005. Available at `java.sun.com/docs/books/jls`.

[4] R. Lämmel and V. Zaytsev. An Introduction to Grammar Convergence. In *Integrated Formal Methods, 7th International Conference, IFM 2009, Proceedings*, volume 5423 of *LNCS*, pages 246–260. Springer, 2009.

[5] R. Lämmel and V. Zaytsev. Recovering Grammar Relationships for the Java Language Specification. In *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 178–186. IEEE, Sept. 2009. Full version submitted for journal publication.