The Journal of Open Source Software

Lerche: Generating data file processors in Julia from EBNF grammars

James R. Hester[*1] **and Erez Shinan**[2]

1 Australian Nuclear Science and Technology Organisation, Sydney, Australia **2** Independent researcher

DOI: 10.21105/joss.03497

Software

- Review
- Repository
- Archive

Editor: Sebastian Benthall

Reviewers:

- @ziotom78
- @eschnett

Submitted: 03 May 2021
Published: 24 August 2021

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

In a scientific context, structured text is commonly encountered in data files and domain-specific languages (DSLs) for handling data. Extended Backhaus-Naur Format (EBNF) (ISO Central Secretary, 1996) is a well-established standard for describing the syntax of such structured text. A useful subset of such grammars is known as `LALR(1)`, meaning that the grammars describe text that can be unambiguously parsed based only on the tokens already seen and the next token of input (DeRemer, 1969). LALR(1) stands out for being able to parse most programming languages, while guaranteeing O(n) run-time complexity and very light memory use. The `Lerche` Julia package automatically generates a parser that processes any data file or domain-specific language that can be described using a LALR(1) EBNF. The parse tree can be immediately transformed into application-specific data structures using user-supplied rules. This parser generator fills a gap in standards-based scientific work for the Julia ecosystem.

Statement of Need

Standards for scientific data formats are essential for correct interpretation of data produced and consumed by parties that are not in direct contact. Standardisation is also becoming increasingly important as requirements for accessible and reusable data increase among funding bodies and publishers. The use of formal description languages, such as EBNF, removes ambiguity that may arise when using natural language descriptions for these standards. Furthermore, a parser that is machine-generated from an EBNF description is guaranteed to generate a correct parse tree from conformant data, unlike hand-written parsers. In addition, development of a formal scientific standard written using EBNF is aided by the availability of EBNF parsers that identify ambiguities and errors.

Scientific data readers typically form a small component of larger projects, which will dictate the programming environment and policies. This environment restricts the choice of EBNF parser generators; an EBNF parser generator that executes in the usual project environment is preferable to one that introduces new build-time dependencies and environments. `Lerche` was translated into Julia from the Python Lark project (Shinan, 2021) in order to remove the need for any extra language dependencies or build steps when developing standards-conformant data format readers for Julia projects. The generated parsers run within the Julia language environment, and are straightforward to integrate into larger Julia projects.

Other native parser tools available for Julia projects include `ParserCombinator` (Cooke, 2021), `Pegparser` (Schneider, 2020), and the built-in Julia macro system. None of these

[*] Corresponding author

take EBNF as input. Reliance on EBNF parser generators external to the Julia environment, for example, by calling Lark or pre-generating parsers, complicates distribution and creates a barrier to user contribution to packages.

dREL (Spadaccini et al., 2012) is a relational data processing language for specifying crystallographic algorithms. The dREL LALR(1) EBNF (Hester, 2021a) was developed using `Lerche` to verify its correctness and conformance to LALR(1) requirements. The `DrelTools` (Hester, 2021c) package is built around the parser automatically generated by `Lerche` from this EBNF. `Lerche` is also used by `CrystalInfoFramework` (Hester, 2021b) to generate the parser for data files written in the Crystallographic Information Framework (CIF) format (Bernstein et al., 2016).

Performance

One of Julia's attractions is the potential for performance close to that of a compiled language, while retaining features normally found in high-level interpreted languages. Fully realising this potential usually requires some investment in code design and profiling, which has so far been minimal for `Lerche.jl`. Nevertheless, parsing a 500K mmCIF text file from the worldwide Protein Data Bank (Berman et al., 2003) with `Lerche.jl` is around 3 times faster than using Lark with CPython, if initial compilation time is ignored. This improvement essentially disappears if the PyPy just-in-time Python interpreter is used instead of CPython. Therefore, although further improvements to `Lerche.jl` runtime performance are likely possible and may be realised in future versions, at this point switching to Julia and `Lerche.jl` to improve performance should be weighed carefully against the option of simply switching to a faster Python interpreter.

Features

`Lerche` optionally augments the range of applicable EBNF grammars by using contextual lexing. In this mode, only those tokens that are possible in the current parsing state are matched by the lexer, which can be useful for grammars in which certain character sequences match more than one type of token; for example, a keyword may also be a possible value for a plain sequence of characters in certain contexts, in which case a non-contextual lexer might wrongly fail to recognise the keyword.

The Lark grammars recognised by `Lerche` extend the EBNF standard in several ways. To aid in composability, they support templating, and importing rules and terminals from other grammars. To aid in refactoring, they support expressing rule semantics, which are translatable to common tree operations. For example, rules starting with _ are considered to be auxiliary, and don't produce their own node. Terminals starting with _ aren't included in the tree, which is often desired for punctation such as commas and parentheses. In order to resolve possible ambiguities or conflicts, there is support for specifying priority in terminals and rules.

References

Berman, H. M., Henrick, K., & Nakamura, H. (2003). Announcing the worldwide protein data bank. *Nature Structural Biology*, *10*(12).

Bernstein, H. J., Bollinger, J. C., Brown, I. D., Gražulis, S., Hester, J. R., McMahon, B., Spadaccini, N., Westbrook, J. D., & Westrip, S. P. (2016). Specification of the Crystallographic Information File format, version 2.0. *Journal of Applied Crystallography*, *49*(1), 277–284. https://doi.org/10.1107/S1600576715021871

Cooke, A. (2021). ParserCombinator.jl. In *Github repository*. Github. https://github.com/andrewcooke/ParserCombinator.jl

DeRemer, F. L. (1969). *Practical translators for LR(k) languages* [PhD thesis]. MIT.

Hester, J. R. (2021a). A draft annotated grammar for dREL. In *Github repository*. Github. https://github.com/COMCIFS/dREL/blob/master/annotated-grammar.rst

Hester, J. R. (2021b). CrystalInfoFramework.jl. In *Github repository*. Github. https://github.com/jamesrhester/CrystalInfoFramework.jl

Hester, J. R. (2021c). DrelTools.jl. In *Github repository*. Github. https://github.com/jamesrhester/DrelTools.jl

ISO Central Secretary. (1996). *Information technology — Syntactic metalanguage — Extended BNF* (Standard ISO/IEC 14977:1996). International Organization for Standardization. https://www.iso.org/standard/26153.html

Schneider, A. (2020). PEGParser.jl. In *Github repository*. Github. https://github.com/abeschneider/PEGParser.jl

Shinan, E. (2021). Lark. In *Github repository*. Github. https://github.com/lark-parser/Lark

Spadaccini, N., Castleden, I. R., Boulay, D. du, & Hall, S. R. (2012). dREL: A relational expression language for dictionary methods. *Journal of Chemical Information and Modeling*, *52*(8), 1917–1925. https://doi.org/10.1021/ci300076w