Parsing Expression GLL

Moss, Aaron
mossa@up.edu

Harrington, Brynn
harringt23@up.edu

Hoppe, Emily
hoppe23@up.edu

November 26, 2021

Abstract

This paper presents an extension of the GLL parsing algorithm for context-free grammars which also supports parsing expression grammars with ordered choice and lookahead. The new PEGLL algorithm retains support for unordered choice, and thus parses a common superset of context-free grammars and parsing expression grammars. As part of this work, the authors have modified an existing GLL parser-generator to support parsing expression grammars, adding operators for common parsing expressions and modifying the lexer algorithm to better support ordered choice. Performance results of the generated parsers are compared to competing parser-generators.

1 Introduction

The inherently unambiguous nature of parsing expression grammars (PEGs) makes them an attractive choice for modelling structured text such as programming languages and computing data formats, but in practice the superior performance of parsers based on context-free grammars (CFGs) has led to CFGs being more-widely used, despite the difficulty of disambiguating them. This work is an initial effort toward a unified framework that provides the advantages of both grammar formalisms: it is an algorithm adopted from an efficient, general-purpose CFG parser that supports PEG semantics without discarding support for the unordered choice operator of CFGs in cases where that ambiguity may be desirable.

More specifically, this paper presents a modification of the GLL parser-generator of Scott & Johnstone[1, 2]. The key contribution is the *FailCRF* data structure, which adds a failure path to Scott & Johnstone's call-return forest; the addition of a failure path allows the lookahead and ordered choice operations of the PEG formalism to be supported. The authors have extended Ackerman's GoGLL[3] parser-generator to implement this new algorithm, adding syntactic sugar for common PEG operators and modifying the lexer algorithm to allow the PEG parser to override the usual maximal-munch rule. This paper also presents benchmarking results from comparing our new *PEGLL* parser-generator against existing algorithms.

$$u(s) = \begin{cases} v & s = uv \\ \mathsf{fail} & \text{otherwise} \end{cases} \qquad A(s) = (\mathcal{R}(A))(s)$$

$$\varepsilon(s) = s \qquad \alpha\beta(s) = \begin{cases} \beta(\alpha(s)) & \alpha(s) \neq \mathsf{fail} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

$$\varnothing(s) = \mathsf{fail} \qquad \alpha/\beta(s) = \begin{cases} \alpha(s) & \alpha(s) \neq \mathsf{fail} \\ \beta(s) & \text{otherwise} \end{cases}$$

$$!\alpha(s) = \begin{cases} s & \alpha(s) = \mathsf{fail} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

$$\&\alpha(s) = \begin{cases} s & \alpha(s) \neq \mathsf{fail} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Figure 1: Formal definitions of parsing expressions

2 Parsing Expression Grammars

The primary difference between parsing expression grammars and the more familiar context-free grammars is *ordered choice*: PEGs, as a formalism of recursive-descent parsing, do not try subsequent alternatives of an alternation if an earlier alternative matches. The other significant difference between the PEG and CFG formalisms are the PEG *lookahead* expressions, $!\alpha$ and $\&\alpha$, which match only if the subexpression α does not (resp. does) match, but consume no input regardless. These lookahead operators provide the infinite lookahead of the PEG formalism. The other fundamental PEG operators act much like their CFG equivalents, and are described in Fig. 1 as functions over an input string s drawn from some alphabet Σ producing either a (matching) suffix of s or the special value $\mathsf{fail} \notin \Sigma^*$. In summary, the *string literal* u matches and consumes the string u, the *empty expression* ε always matches without consuming anything, while the *failure expression* $\varnothing$ never matches. The *sequence* expression $\alpha\beta$ matches α followed by β, while the *ordered choice* expression α/β only tries β if α does not match. To differentiate CFG *unordered choice*, it is represented in this paper as $\alpha|\beta$.

References

[1] E. Scott and A. Johnstone, "Gll parsing," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 7, pp. 177–189, 2010.

[2] E. Scott and A. Johnstone, "Structuring the gll parsing algorithm for performance," *Science of Computer Programming*, vol. 125, pp. 1–22, 2016.

[3] M. Ackerman, "GoGLL." `https://github.com/goccmack/gogll`, 2019. accessed 24-Nov-2021.