From EBNF to PEG
Extended Abstract

Roman R. Redziejowski

roman.redz@swipnet.se

1 Introduction

This is a continuation of paper [5] presented at CS&P'2012 and of its improved version [6]. The subject is conversion of grammars from Extended Backus-Naur Form to Parsing Expression Grammars. Parsing Expression Grammar (PEG), as introduced by Ford [1,2], is essentially a recursive-descent parser with limited backtracking. The parser does not require a separate "lexer" to preprocess the input, and the limited backtracking lifts the LL(1) restriction imposed on top-down parsers.

In spite of its apparent similarity to Extended Backus-Naur Form (EBNF), PEG often defines quite a different language. The question is: when an EBNF grammar can be used as its own PEG parser? As found by Medeiros [3,4], this is, in particular, true for LL(1) languages. Which is of not much use if we use PEG just to circumvent the LL(1) restriction. But, as noticed in [5], this result is valid for a much wider class of grammars. Take as an example this grammar:

$$\begin{array}{lll} S = A|B & & \\ A = CxZ & B = DyZ & \\ C = (a|c)^+ & D = (a|d)^+ & Z = z^+ \end{array} \tag{1}$$

This grammar is not LL(1), and not even LL(k) for any k: each of A and B may start with any number of a's. But, a PEG parser invoked with input "aaayzz" will first try A, call C accepting "aaa", then finding "y" instead of "x" it will backtrack and successfully accept B.

2 Previous Result

In [5,6], we considered a very simple grammar that has only two forms of rules: "choice" $A = e_1|e_2$ and "sequence" $A = e_1e_2$, where A is the name of the rule, and each of e_1, e_2 is a letter of the input alphabet Σ, the name of a rule, or the empty-word marker ε. Any EBNF grammar or PEG can be reduced to this form by introducing additional rules. The names of rules are the "nonterminals" of the grammar. A nonterminal, a letter, the empty-word marker, or a formula $e_1|e_2$ or e_1e_2 is referred to as an "expression". The set of all expressions of the grammar is denoted by $\mathbb{E}$. The grammar is assumed not to be left-recursive.

Following [3,4], we used the method of "natural semantics" to formally define two interpretations of the grammar: as EBNF and as PEG. The method consists in defining two relations, $\overset{\text{BNF}}{\leadsto}$ and $\overset{\text{PEG}}{\leadsto}$.

Relation $\overset{\text{BNF}}{\leadsto}$ is a subset of $\mathbb{E} \times \Sigma^* \times \Sigma^*$. We write $[e]\ x \overset{\text{BNF}}{\leadsto} y$ to mean that the relation holds for $e \in \mathbb{E}$ and $x, y \in \Sigma^*$. The relation is formally defined by a set of inference rules shown in Figure 1: it holds if and only if it can be proved using these rules. The rules are so constructed that $[e]\ xy \overset{\text{BNF}}{\leadsto} y$ if and only if the prefix x of xy belongs to the language $\mathcal{L}(e)$ of e according to the EBNF interpretation.

$$\frac{}{[\varepsilon]\ x \overset{\text{BNF}}{\leadsto} x}\ \textbf{(empty.b)} \qquad \frac{}{[a]\ ax \overset{\text{BNF}}{\leadsto} x}\ \textbf{(letter.b)}$$

$$\frac{[e_1]\ xyz \overset{\text{BNF}}{\leadsto} yz \quad [e_2]\ yz \overset{\text{BNF}}{\leadsto} z}{[e_1e_2]\ xyz \overset{\text{BNF}}{\leadsto} z}\ \textbf{(seq.b)}$$

$$\frac{[e_1]\ xy \overset{\text{BNF}}{\leadsto} y}{[e_1|e_2]\ xy \overset{\text{BNF}}{\leadsto} y}\ \textbf{(choice.b1)} \qquad \frac{[e_2]\ xy \overset{\text{BNF}}{\leadsto} y}{[e_1|e_2]\ xy \overset{\text{BNF}}{\leadsto} y}\ \textbf{(choice.b2)}$$

Fig. 1. EBNF semantics

Relation $\overset{\text{PEG}}{\leadsto}$ is a subset of $\mathbb{E} \times \Sigma^* \times \{\Sigma^* \cup \text{fail}\}$. We write $[e]\ x \overset{\text{PEG}}{\leadsto} Y$ to mean that the relation holds for $e \in \mathbb{E}$, $x \in \Sigma^*$ and $Y \in \{\Sigma^* \cup \text{fail}\}$. The relation is formally defined by a set of inference rules shown in Figure 2: it holds if and only if it can be proved using these rules. The rules are so constructed that $[e]\ xy \overset{\text{PEG}}{\leadsto} y$ if and only if parsing expression e applied to xy consumes x, and $[e]\ x \overset{\text{PEG}}{\leadsto} \text{fail}$ if and only if e fails when applied to x.

$$\frac{}{[\varepsilon]\ x \overset{\text{PEG}}{\leadsto} x}\ \textbf{(empty.p)}$$

$$\frac{}{[a]\ ax \overset{\text{PEG}}{\leadsto} x}\ \textbf{(letter.p1)} \qquad \frac{b \neq a}{[b]\ ax \overset{\text{PEG}}{\leadsto} \text{fail}}\ \textbf{(letter.p2)} \qquad \frac{}{[a]\ \varepsilon \overset{\text{PEG}}{\leadsto} \text{fail}}\ \textbf{(letter.p3)}$$

$$\frac{[e_1]\ xyz \overset{\text{PEG}}{\leadsto} yz \quad [e_2]\ yz \overset{\text{PEG}}{\leadsto} Z}{[e_1e_2]\ xyz \overset{\text{PEG}}{\leadsto} Z}\ \textbf{(seq.p1)} \qquad \frac{[e_1]\ x \overset{\text{PEG}}{\leadsto} \text{fail}}{[e_1e_2]\ x \overset{\text{PEG}}{\leadsto} \text{fail}}\ \textbf{(seq.p2)}$$

$$\frac{[e_1]\ xy \overset{\text{PEG}}{\leadsto} y}{[e_1|e_2]\ xy \overset{\text{PEG}}{\leadsto} y}\ \textbf{(choice.p1)} \qquad \frac{[e_1]\ x \overset{\text{PEG}}{\leadsto} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\leadsto} Y}{[e_1|e_2]\ xy \overset{\text{PEG}}{\leadsto} Y}\ \textbf{(choice.p2)}$$

where Y denotes y or fail and Z denotes z or fail.

Fig. 2. PEG semantics

Using these definitions, we obtained a sufficient condition for the two interpretations to be equivalent, namely, that each choice $A = e_1|e_2$ satisfies:

$$\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\,\mathrm{Tail}(A) = \varnothing, \tag{2}$$

where $\mathcal{L}(e)$ is the language of e according to the EBNF interpretation, and $\mathrm{Tail}(A)$ is any string that can follow A, up to the end of input. If S denotes the grammar's starting rule, and $\$$ is the end-of-text symbol, $\mathrm{Tail}(A)$ is formally defined as the set of strings $y\$$ such that the proof of $[S]\ w\$ \overset{\mathrm{BNF}}{\rightsquigarrow} \$$ for some $w \in \mathcal{L}(S)$ contains a partial proof of $[A]\ xy\$ \overset{\mathrm{BNF}}{\rightsquigarrow} y\$$ for some x.

The meaning of (2) is quite obvious: e_1 must not compete with e_2. The problem is in verifying it, as we have there an intersection of context-free languages whose emptiness is, in general, undecidable. The approach proposed in [5, 6] is to approximate the involved languages by languages of the form $X\Sigma^*$ where $X \subseteq \Sigma^+$. It results in the following condition, stronger than (2):

There exist $X, Y \subseteq \Sigma^+$ such that

$$\begin{aligned} &X\Sigma^* \supseteq \mathcal{L}(e_1),\\ &Y\Sigma^* \supseteq \mathcal{L}(e_2)\,\mathrm{Tail}(A),\\ &X \asymp Y, \end{aligned} \tag{3}$$

where $X \asymp Y$ means $X\Sigma^* \cap Y = Y\Sigma^* \cap X = \varnothing$.

The sets X and Y can, in particular, be the sets of possible first letters of words in $\mathcal{L}(e_1)$ respectively $\mathcal{L}(e_2)\,\mathrm{Tail}(A)$. For such sets, $X \asymp Y$ is equivalent to $X \cap Y = \varnothing$, and the condition is identical to LL(1).

Even if the language is not LL(1), it may satisfy (2) if instead of single letters we take some longer prefixes. A natural way to approximate $\mathcal{L}(e)$ by $X\Sigma^*$ is to take as X the set of strings accepted by the first parsing procedures possibly called by e. Or the first procedures called by them. If such approximations X, Y satisfying (3) above exist, we have a parser that chooses its way by examining the input ahead within the reach of one parsing procedure. It was suggested use the name LL(1p) for languages that can be so parsed.

To find the possible approximations by first procedures, we used the relation `first` where `first`(e) is the set of procedures called as first directly from e.

3 Beyond LL(1p)

In the example grammar (1), the first procedures of A and B are, respectively, C and D, and $\mathcal{L}(C) \not\asymp \mathcal{L}(D)$, so this grammar is not LL(1p). However, $X = \mathcal{L}(Cx)$ and $Y = \mathcal{L}(Dz)$ satisfy (3), guaranteeing that the grammar defines the same language under both interpretation. Here the parser chooses its way by looking at the text ahead within the reach of two parsing procedures. We can refer to such grammar as being LL(2p). As we remarked before, it is not LL(2).

Checking if the grammar is LL(kp) requires finding possible sets of first k procedures. This can be done using relation $\texttt{first}_k$, similar to that used for checking LL(k). Although the sets become large for larger k, it is a mechanical procedure. However, checking of the relation $\asymp$ between these sets may not be simple as it involves intersection of context-free languages. If we are lucky, the languages may be regular, as in the above example. But in general, using approximation by first procedures to check (2) is not always feasible, even for $k = 1$.

4 Looking Farther Ahead

The mechanism used above to look far ahead in the input is the backtracking of PEG. But, this backtracking is limited. When faced with $e_1|e_2$, the parser cannot look ahead beyond e_1 and then backtrack if it does not like what it sees there. There exist grammars where the parser has to look beyond e_1 to make the correct decision. An example is the following grammar, modeled after [3,4]:

$$\begin{aligned} &S = Xz && \\ &X = A|B && A = ab|C \\ &B = a|Cd && C = c \end{aligned} \tag{4}$$

Interpreted as PEG, this grammar does not accept the string cdz that belongs to $\mathcal{L}(S)$: A succeeds on c via C, leaving no chance to B. The grammar is not LL(1p), nor even LL(kp) in the sense defined above. However, the grammar is LL(2): a top-down parser can choose between A and B by looking at two letters ahead: they are ab or cz for A and az or cd for B. The reason for the failure of PEG is that X cannot look beyond A when faced with c as the first letter.

PEG has a special operation to examine the input ahead: the "and-predicate" $\&e$. It means: "invoke the expression e on the text ahead and backtrack; return success if e succeeded or failure if it failed". This can be formally defined by two inference rules:

$$\frac{[e]\ xy \overset{\text{PEG}}{\leadsto} y}{[\&e]\ xy \overset{\text{PEG}}{\leadsto} xy}\ \textbf{(and.p1)} \qquad \frac{[e]\ x \overset{\text{PEG}}{\leadsto} \text{fail}}{[\&e]\ x \overset{\text{PEG}}{\leadsto} \text{fail}}\ \textbf{(and.p2)}$$

In order to look beyond e_1 in $A = e_1|e_2$, we can modify the grammar by adding $\&e_0$ after e_1, obtaining $A = e_1\&e_0|e_2$.

Consider as an example the grammar (4). The only rule that does not satisfy (2) is $X = A|B$. One can easily see that by replacing it with $X = A\&z|B$, we obtain a PEG defining the same language as the EBNF grammar (4). This idea has been used in [3,4] to construct PEGs for LL(k) languages. We consider it here for a wider class of languages.

We are going to consider the grammar where each choice has the form $A = e_1\&e_0|e_2$. EBNF does not have the and-predicate; in order to speak of two interpretations of the grammar, we define $\&e$ to be a dummy EBNF operation with $\mathcal{L}(\&e) = \varepsilon$:

$$\frac{}{[\&e]\ x \overset{\text{BNF}}{\leadsto} x}\quad \textbf{(and.b)}$$

The problem is to choose the expression e_0. We are going to show that the two interpretations are equivalent if each choice $A = e_1\&e_0|e_2$ satisfies these conditions:

$$\text{Parsing expression } e_0 \text{ succeeds on every } w \in \text{Tail}(A), \tag{5}$$

$$\mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^* \cap \mathcal{L}(e_2)\,\text{Tail}(A) = \varnothing\ . \tag{6}$$

(Note that by taking $e_0 = \varepsilon$, we obtain an &-free choice and (5,6) become identical to (2).)

The demonstration consists of three Propositions:

Proposition 1. *For every $e \in \mathbb{E}$ and $w \in \Sigma^*$, there exists a proof of either $[e]\ w \overset{PEG}{\leadsto} \text{fail}$ or $[e]\ w \overset{PEG}{\leadsto} y$ where $w = xy$ for some x.*

Proof. This is proved in [3] using a result from [2]. A self-contained proof given in [6] is easily extended to include the and-predicate.

Proposition 2. *For each $e \in \mathbb{E}$ and $x, y \in \Sigma^*$, $[e]\ xy \overset{PEG}{\leadsto} y$ implies $[e]\ xy \overset{BNF}{\leadsto} y$.*

Proof. This is proved as Lemma 4.3.1 in [3]. The proof is easily extended to include the and-predicate in EBNF.

Proposition 3. *If every choice $A = e_1\&e_0|e_2$ satisfies (5,6) then for every $w \in \mathcal{L}(S)$ there exists a proof of $[S]\ w\$ \overset{PEG}{\leadsto} \$$.*

Proof. We show that for every partial result $[e]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ in the proof of $[S]\ w\$ \overset{\text{BNF}}{\leadsto} \$$ there exists a proof of $[e]\ xy\$ \overset{\text{PEG}}{\leadsto} y\$$. We use induction on the height n of the proof tree for $[e]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$.

The case of $n = 1$ is easy. Take any $n \geq 1$ and assume the Proposition holds for every tree of height less or equal n. Consider a proof of $[e]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ having height $n+1$. The only non-trivial situation is a proof tree of height $n+1$ where the last step results in $[A]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ for $A = e_1\&e_0|e_2$. Two cases are possible:

Case 1: The result is derived from $[e_1]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ using **and.b**, **seq.b** and **choice.b1**. By induction hypothesis there exists proof of $[e_1]\ xy\$ \overset{\text{PEG}}{\leadsto} y\$$. By definition, $y\$ \in \text{Tail}(A)$. As e_0 succeeds on each string in Tail(A), we have $[\&e_0]\ y\$ \overset{\text{PEG}}{\leadsto} y\$$ from **and.p1**. We can construct a proof of $[e_1\&e_0|e_2]\ xy\$ \overset{\text{PEG}}{\leadsto} y\$$ using **seq.p1** and **choice.p1**.

Case 2: The result is derived from $[e_2]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ using **and.b**, **seq.b** and **choice.b2**. By induction hypothesis there exists proof of $[e_2]\ xy\$ \overset{\text{PEG}}{\leadsto} y\$$. In order to use **choice.p2**, we have to show that $[e_1\&e_0]\ xy\$ \overset{\text{PEG}}{\leadsto} \text{fail}$.
Suppose this is not true. Then, by Proposition 1 exist proofs of $[e_1]\ uv\$ \overset{\text{PEG}}{\leadsto} v\$$ and $[\&e_0]\ v\$ \overset{\text{PEG}}{\leadsto} v\$$ for some u, v such that $uv = xy$. By Proposition 2 exists proof of $[e_1]\ uv\$ \overset{\text{BNF}}{\leadsto} v\$$, which means $u \in \mathcal{L}(e_1)$. The proof of $[\&e_0]\ v\$ \overset{\text{PEG}}{\leadsto} v\$$ requires $[e]\ ts\$ \overset{\text{PEG}}{\leadsto} s\$$ for some t, s such that $ts = v$. By Proposition 2 exists proof of $[e_0]\ ts\$ \overset{\text{BNF}}{\leadsto} s\$$, which means $t \in \mathcal{L}(e_0)$. Thus $xy = uts \in \mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^*$. But $[e_2]\ xy\$ \overset{\text{BNF}}{\leadsto} y\$$ means $xy \in \mathcal{L}(e_2)\,\text{Tail}(A)$, which contradicts (6). ☐

One can easily see that the necessary condition for the required e_0 to exist is:

$$\mathcal{L}(e_1)\,\mathrm{Tail}(A) \cap \mathcal{L}(e_2)\,\mathrm{Tail}(A) = \varnothing\ .$$

A systematic way of choosing a suitable e_0 is still to be found.
It seems that for the LL(k) languages e_0 should be the expression consuming exactly FOLLOW$_k(A)$.

References

1. Ford, B.: Packrat parsing: a practical linear-time algorithm with backtracking. Master's thesis, Massachusetts Institute of Technology (2002) `http://pdos.csail.mit.edu/papers/packrat-parsing:ford-ms.pdf`
2. Ford, B.: Parsing expression grammars: A recognition-based syntactic foundation. In Jones, N.D., Leroy, X., eds.: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, ACM (2004) 111–122
3. Medeiros, S.: Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro (2010)
4. Mascarenhas, F., Medeiros, S., Ierusalimschy, R.: On the relation between context-free grammars and Parsing Expression Grammars. UFRJ Rio de Janeiro, UFS Aracaju, PUC-Rio, Brazil (2013) `http://arxiv.org/pdf/1304.3177v1`
5. Redziejowski, R.R.: From EBNF to PEG. In Popova-Zeugmann, L., ed.: Proceedings of the 21th International Workshop on Concurrency, Specification and Programming Berlin, Germany, September 26-28, 2012, Humboldt University of Berlin (2012) 324–335
`http://ceur-ws.org/Vol-928/`
6. Redziejowski, R.R.: From EBNF to PEG. Fundamenta Informaticae (2013) to appear