Fundamenta Informaticae 133 (2014) 257–270
DOI 10.3233/FI-2014-1075
IOS Press

More About Converting BNF to PEG

Roman R. Redziejowski*
Giraf's Research
roman.redz@swipnet.se

Abstract. Parsing Expression Grammar (PEG) encodes a recursive-descent parser with limited backtracking. The parser has many useful properties. Converting PEG to an executable parser is a rather straightforward task. Unfortunately, PEG is not well understood as a language definition tool. It is thus of a practical interest to construct PEGs for languages specified in some familiar way, such as Backus-Naur Form (BNF). The problem was attacked by Medeiros in an elegant way by noticing that both PEG and BNF can be formally defined in a very similar way. Some of his results were extended in a previous paper by this author. We continue here with further extensions.

1. Introduction

Parsing Expression Grammar (PEG), as introduced by Ford in [4], encodes a recursive-descent parser with limited backtracking. Using the "memoization" or "packrat" technology described in [2, 3], the parser can work in linear time. It does not require a separate "lexer" to preprocess the input, and the limited backtracking lifts the LL(1) restriction usually imposed by top-down parsers. These properties are useful in many applications.

Converting PEG to an executable parser is a rather straightforward task, and several parser generators are available to perform it. The Wikipedia entry on comparison of parser generators contains a long list. However, PEG is not well understood as a language specification tool. It is thus of a practical interest to construct PEGs for languages specified in some familiar way, such as Backus-Naur Form (BNF). This was successfully attempted by Medeiros in his Ph.D. thesis [6]. The thesis is in Portuguese; an extended English version is now available on Computing Research Repository [5]. The PEG and BNF are very similar in appearance, but their formal definitions are worlds apart. The strategy invented by Medeiros was to replace these definitions by "natural semantics". The natural semantics of PEG and BNF are very close to each other, which can be efficiently used to study relationships between the two kinds of grammar.

*Address for correspondence: Ceremonimästarvägen 10, SE-181 40 Lidingö, Sweden

One of the results obtained using this method was that a BNF grammar with LL(1) property is converted by only minor typographical changes into a PEG defining the same language. One can say that such grammar is its own PEG parser. This turns out to be true for a much larger class of grammars. In a previous paper [7], we formulated a sufficient condition for a grammar to be its own PEG parser. There is no general method to check this condition, as it involves deciding emptiness of an intersection of context-free languages. There are, however, special cases that are easy to check for. One of them is just the LL(1) property.

In [7], we suggested a method of checking the condition by approximations. It applies to a class of grammars that we called LL(1P). It means that a PEG parser can choose its way by looking at the input within the reach of one parsing procedure, instead of just one letter as in the case of LL(1). We present here the method in a slightly different form, and indicate how it could be extended to LL(kP) grammars that allow the parser to look ahead within the reach of k procedures.

The feature of a PEG parser that gives it the ability to look far ahead is backtracking. Because the backtracking is limited, some simple grammars cannot act as their own PEG parsers. In [5,6], one shows how to construct PEG parsers for such grammars using the PEG's "and-predicate" to look as far ahead as needed. The constructions presented there apply to strong LL(k) and LL-regular grammars. We note that the method can be applied to a wider class of grammars and present conditions to be satisfied by the predicate, but give no general method to construct it.

All proofs are collected in the Appendix.

2. Some Notation

We consider a finite alphabet Σ of *letters*. A finite string of letters is a *word*. The string of 0 letters is called the *empty word* and is denoted by ε. The set of all words is Σ^*. A subset of Σ^* is a *language*.

As usual, we write XY to mean the concatenation of languages X and Y, that is, the set of all words xy where $x \in X$ and $y \in Y$.

A relation R on $\mathbb{E}$ is a subset of $\mathbb{E} \times \mathbb{E}$. As usual, we write $R(e)$ to mean the set of all e' such that $(e, e') \in R$, and $R(E)$ to mean the union of $R(e)$ for all $e \in E \subseteq \mathbb{E}$. The transitive closure of R is denoted by R^+, and the product of relations R and S by $R \times S$.

3. The Grammar

We consider a grammar $\mathbb{G}$ over the alphabet Σ that will be interpreted as either PEG or BNF. The grammar consists of a finite nonempty set N of symbols distinct from the letters of Σ, and a set of equations, one for each symbol of N. The equation for $A \in N$ has one of these two forms:

- $A = e_1 e_2$ ("sequence"),
- $A = e_1 | \, e_2$ ("choice"),

where each of e_1, e_2 is an *expression*: an element of N, a letter from Σ, or the symbol $\boldsymbol{\varepsilon}$[1]. The set $N \cup \Sigma \cup \{\boldsymbol{\varepsilon}\}$ of all expressions is in the following denoted by $\mathbb{E}$.

[1]The boldface $\boldsymbol{\varepsilon}$ here is a special symbol, to be distinguished from the empty word ε.

When the grammar $\mathbb{G}$ is interpreted as BNF, the equation $A = e$ represents a definition sometimes written as $A ::= e$, and $e_1 | \, e_2$ is the unordered choice. When it is interpreted as PEG, the equation represents a definition of parsing expression, written as $A \leftarrow e$ in [4], and $e_1 | \, e_2$ is the ordered choice. In each case, the elements of Σ are the "terminals" of the grammar, and the elements of N are the "nonterminals".

The grammar has been reduced to bare bones in order to simplify the analysis. Any BNF grammar and PEG without predicates can be reduced to this form by removing syntactic sugar and introducing new nonterminals.

3.1. The BNF Interpretation

The grammar $\mathbb{G}$ interpreted as BNF is a mechanism for generating words:

- ε generates ε.
- $a \in \Sigma$ generates itself.
- $A = e_1 e_2$ generates any word generated by e_1 followed by any word generated by e_2.
- $A = e_1 | \, e_2$ generates any word generated by e_1 or e_2.

The language $\mathcal{L}(e)$ of $e \in \mathbb{E}$ is the set of all words that can be generated by e.

Following [5,6], we formally define the BNF interpretation by means of a relation $\overset{\text{BNF}}{\rightsquigarrow} \subseteq (\mathbb{E} \times \Sigma^*) \times \Sigma^*$. We write $[e] \; x \overset{\text{BNF}}{\rightsquigarrow} y$ to mean that the relation holds for $e \in \mathbb{E}$ and $x, y \in \Sigma^*$. The relation is 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 notation "$A = e$" in this and the next Figure means "the equation for A is $A = e$".)

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

$$\frac{A = e_1 e_2 \quad [e_1] \; xyz \overset{\text{BNF}}{\rightsquigarrow} yz \quad [e_2] \; yz \overset{\text{BNF}}{\rightsquigarrow} z}{[A] \; xyz \overset{\text{BNF}}{\rightsquigarrow} z} \textbf{ (seq.b)}$$

$$\frac{A = e_1 | \, e_2 \quad [e_1] \; xy \overset{\text{BNF}}{\rightsquigarrow} y}{[A] \; xy \overset{\text{BNF}}{\rightsquigarrow} y} \textbf{ (choice.b1)} \qquad \frac{A = e_1 | \, e_2 \quad [e_2] \; xy \overset{\text{BNF}}{\rightsquigarrow} y}{[A] \; xy \overset{\text{BNF}}{\rightsquigarrow} y} \textbf{ (choice.b2)}$$

Figure 1. BNF semantics

One can verify by structural induction that for any $y \in \Sigma$, we have $[e] \; xy \overset{\text{BNF}}{\rightsquigarrow} y$ if and only if $x \in \mathcal{L}(e)$. This gives $\mathcal{L}(e) = \{x \in \Sigma^* : \, [e] \; x \overset{\text{BNF}}{\rightsquigarrow} \varepsilon\}$.

As it will be seen later on, some results depend on whether $\varepsilon \in \mathcal{L}(e)$ for given $e \in \mathbb{E}$. This can be decided using a simple iterative algorithm, independent of the above formal definition.

3.2. The PEG Interpretation

When the grammar $\mathbb{G}$ is interpreted as PEG, the expressions represent parsing procedures that can call each other recursively. In general, parsing procedure is applied to a word from Σ^* and tries to recognize an initial portion of that word. If it succeeds, it "consumes" the recognized portion and returns "success";

otherwise, it returns "failure" and does not consume anything. The action of different procedures is as follows:

- ε: Indicate success without consuming any input.
- $a \in \Sigma$: If the text ahead starts with a, consume it and return success. Otherwise return failure.
- $A = e_1\, e_2$: Call e_1. If it succeeded, call e_2 and return success if e_2 succeeded.
  If e_1 or e_2 failed, backtrack: reset the input as it was before the invocation of e_1 and return failure.
- $A = e_1 |\, e_2$: Call e_1. Return success if it succeeded.
  Otherwise call expression e_2 and return success if e_2 succeeded or failure if it failed.

Note that backtracking is limited to the sequence expression. Once e_1 in $e_1 |\, e_2$ succeeded, a subsequent failure will not cause backtracking to try e_2.

We formally define the PEG interpretation by means of a relation $\overset{\text{PEG}}{\rightsquigarrow} \subseteq (\mathbb{E} \times \Sigma^*) \times \{\Sigma^* \cup \text{fail}\}$. We write $[e]\ w \overset{\text{PEG}}{\rightsquigarrow} X$ to mean that the relation holds for $e \in \mathbb{E}$, $w \in \Sigma^*$, and $X \in \{\Sigma^* \cup \text{fail}\}$. The relation is defined by a set of inference rules shown in Figure 2: it holds if and only if it can be proved using these rules.

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

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

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

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

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

Figure 2. PEG semantics

One can easily see that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ if and only if parsing expression e applied to xy consumes x, and $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$ if and only if e fails when applied to x. For example, **seq.p1** with $Z = \text{fail}$ represents backtracking in $e_1 e_2$, and **choice.p2** represents $e_1 |\, e_2$ calling e_2 after e_1 failed.

4. Previous Results

Because of the ability of PEG to look ahead, we have to compare the two interpretations of the grammar when applied to complete input strings. For this reason, we postulate that the grammar contains a unique equation $S = e\$$ where $S \in N$ is a distinguished *start symbol*, e is a member of $\mathbb{E}$, and $\$ \in \Sigma$ is a distinguished letter, the *end-of-text marker*. Both S and $\$$ appear only in this equation. We shall consider the two interpretations equivalent if for each proof of $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ exists a proof of $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ and vice-versa.

For $A \in N$ where $A \neq S$, we define $\text{Tail}(A)$ to be the set of such $y \in \Sigma^*$ that $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ appears as partial result in the proof of $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ for some $w \in \Sigma^*$. One can verify by induction on structure of

the proof tree that each such partial result must have the form $[A]\ xz\$ \overset{\text{BNF}}{\rightsquigarrow} z\$$. In other words, $\mathrm{Tail}(A)$ is the set of terminated input strings that may follow a substring from $\mathcal{L}(A)$ in a word belonging to $\mathcal{L}(S)$.

We borrow from [1, 8] the definitions of `Follow` and `First`.

For $e \in \mathbb{E}$, define $\mathtt{Last}(e)$ to be the set of all $A \in N$ such that:

- $A = e|e_1$ for some $e_1 \in \mathbb{E}$, or
- $A = e_1|e$ for some $e_1 \in \mathbb{E}$, or
- $A = e_1\, e$ for some $e_1 \in \mathbb{E}$, or
- $A = e\, e_1$ for some $e_1 \in \mathbb{E}$ where $\varepsilon \in \mathcal{L}(e_1)$.

For $e \in \mathbb{E}$, define $\mathtt{Next}(e) = \{e_1 \in \mathbb{E} :$ exists $A \in N$ such that $A = e\, e_1\}$.
This defines relations `Last` and `Next` on $\mathbb{E}$. Let $\mathtt{Follow} = \mathtt{Last}^* \times \mathtt{Next}$.

For $e \in \mathbb{E}$, define $\mathtt{first}(e)$ as follows:

- $\mathtt{first}(\boldsymbol{\varepsilon}) = \varnothing$.
- For $a \in \Sigma$, $\mathtt{first}(a) = \varnothing$.
- For $A = e_1 e_2$, $\mathtt{first}(A) = \{e_1\}$ if $\varepsilon \notin \mathcal{L}(e_1)$.
- For $A = e_1 e_2$, $\mathtt{first}(A) = \{e_1\} \cup \{e_2\}$ if $\varepsilon \in \mathcal{L}(e_1)$.
- For $A = e_1|\, e_2$, $\mathtt{first}(A) = \{e_1, e_2\}$.

In terms of PEG interpretation, $\mathtt{first}(e)$ is the set of parsing procedures that may be called by e at the start of its input string. Treating `first` as a relation on $\mathbb{E}$, define $\mathtt{First} = \mathtt{first}^+$. Then $\mathtt{First}(e)$ is the set of procedures that may be called directly or indirectly by e to process at the start of its input. The grammar $\mathbb{G}$ is *left-recursive* if $e \in \mathtt{First}(e)$ for some $e \in \mathbb{E}$. If the grammar is left-recursive, its PEG parser may be involved in an infinite descent, meaning that a proof of $\overset{\text{PEG}}{\rightsquigarrow}$ may not exist. On the other hand:

Proposition 4.1. If the grammar $\mathbb{G}$ is not left-recursive then for every $e \in \mathbb{E}$ and $w \in \Sigma^*$ there exists a proof of either $[e]\ w \overset{\text{PEG}}{\rightsquigarrow}$ fail or $[e]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$.

This is proved in [5, 6] by first verifying that the natural semantics given there is equivalent to the formal definition from [4], and then using a result from the same paper that grammar without left-recursion is "complete". In the Appendix, we give an independent proof that makes our presentation self-contained.

The following appears as Lemma 4.3.1 in [6] and Lemma 2.3 in [5]:

Proposition 4.2. For any $e \in \mathbb{E}$ and $x, y \in \Sigma^*$, $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ implies $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$.

As a special case, $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ implies $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$.

The following appears, in a slightly different form, as Proposition 4.3 in [7]:

Proposition 4.3. If the grammar $\mathbb{G}$ is not left-recursive and its every equation $A = e_1|\, e_2$ satisfies

$$\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\,\mathrm{Tail}(A) = \varnothing, \tag{1}$$

then $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ implies $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$.

5. Checking the Condition

Propositions 4.2 and 4.3 establish (1) as a sufficient condition for the grammar $\mathbb{G}$ to be its own PEG parser. The problem with (1) is that we have there an intersection of context-free languages whose emptiness is, in general, undecidable. The approach proposed in [7] is to approximate the involved languages by languages of the form $X\Sigma^*$. We present this approach in a slightly different form.

Let $\mathcal{E} = 2^{\mathbb{E}}$. For a set $\mathbb{X} \subseteq \mathcal{E}$ of expressions, we define the language $\mathcal{L}(\mathbb{X})$ as the union $\bigcup_{e\in\mathbb{X}} e$.
For $\mathbb{X} \subseteq \mathcal{E}$ and $L \subseteq \Sigma^*$ we write $L \sqsubseteq \mathbb{X}$ to mean that $L - \varepsilon \subseteq (\mathcal{L}(\mathbb{X}) - \varepsilon)\Sigma^*$.
For $\mathbb{X}, \mathbb{Y} \subseteq \mathcal{E}$, we write $\mathbb{X} \asymp \mathbb{Y}$ to mean that $(\mathcal{L}(\mathbb{X}) - \varepsilon)\Sigma^* \cap (\mathcal{L}(\mathbb{Y}) - \varepsilon)\Sigma^* = \varnothing$.
We read $L \sqsubseteq \mathbb{X}$ as "L is approximated by $\mathbb{X}$" and $\mathbb{X} \asymp \mathbb{Y}$ as "$\mathbb{X}$ and $\mathbb{Y}$ are prefix-disjoint".

Proposition 5.1. The condition (1) for $A = e_1 | e_2$ is satisfied if $\varepsilon \notin \mathcal{L}(e_1)$ and there exist $\mathbb{X}, \mathbb{Y} \subseteq \mathcal{E}$ such that

$$\mathcal{L}(e_1) \sqsubseteq \mathbb{X}, \tag{2}$$
$$\mathcal{L}(e_2)\,\mathrm{Tail}(A) \sqsubseteq \mathbb{Y}, \tag{3}$$
$$\mathbb{X} \asymp \mathbb{Y}. \tag{4}$$

Checking (1) is thus reduced to finding approximations $\mathbb{X}, \mathbb{Y}$ that satisfy (2)–(4). We shall need some auxiliary results to help in the search. The following properties where $L_1, L_2 \in \Sigma^*$ and $L_1 \sqsubseteq \mathbb{X}, L_2 \sqsubseteq \mathbb{Y}$ are easy to verify:

$$L_1 \cup L_2 \sqsubseteq \mathbb{X} \cup \mathbb{Y}, \tag{5}$$
$$\text{if } \varepsilon \notin L_1 \text{ then } L_1 L_2 \sqsubseteq \mathbb{X}, \tag{6}$$
$$\text{if } \varepsilon \in L_1 \text{ then } L_1 L_2 \sqsubseteq \mathbb{X} \cup \mathbb{Y}. \tag{7}$$

We also need these:

Lemma 5.2. $L \sqsubseteq A \cup \mathbb{X} \Rightarrow L \sqsubseteq \mathtt{first}(A) \cup \mathbb{X}$ for $L \subseteq \Sigma^*$, $A \in N$, and $\mathbb{X} \subseteq \mathcal{E}$.

Lemma 5.3. $\mathrm{Tail}(A) \sqsubseteq \mathtt{Follow}(A)$ for each $A \in N$.

The method suggested in [7] consists of starting with $\mathbb{X} = e_1$ and $\mathbb{Y} = e_2$ or $\mathbb{Y} = e_2 \cup \mathtt{Follow}(A)$, depending on whether or not $\varepsilon \in \mathcal{L}(e_2)$. They obviously satisfy (2) and (3). If they do not satisfy (4), we keep refining them using Lemma 5.2 until we (hopefully) find sets that do. This can be illustrated on the following grammar:

$$\begin{array}{ll} S = X\$ & X = A|B \\ A = CY & C = a^+c^+ \\ B = DZ & D = a^+d^+ \end{array} \tag{8}$$

where $\mathcal{L}(Y)$ and $\mathcal{L}(Z)$ are difficult to compare. (In order to save space, we do not show C and D in our simplified form.) To check (1) for $X = A|B$, we start with $\mathbb{X} = \{A\}$ and $\mathbb{Y} = \{B\}$. As $\{A\} \asymp \{B\}$ is not easy (or possible) to check, we use Lemma 5.2 to replace A by $\mathtt{first}(A) = \{C\}$ and B by $\mathtt{first}(B) = \{D\}$. We have $(a^+c^+)\Sigma^* \cap (a^+d^+)\Sigma^* = \varnothing$, so $\{C\} \asymp \{D\}$, showing that X satisfies (1).

In general, the refining consists of repeatedly replacing any $A \in N$ appearing in $\mathbb{X}$ and / or $\mathbb{Y}$ by $\texttt{first}(A)$. When there is nothing more to replace, we are left with the sets of "first letters" used in checking the LL(1) condition. For such sets, (4) means $\mathbb{X} \cap \mathbb{Y} = \varnothing$, which confirms the result about LL(1) grammars from [5, 6]. Note that (8) above is not LL(1), and not even LL(k) for any k.

The set $\mathbb{X}$ in (2) obtained after the refinement is a subset of $\texttt{First}(e_1)$. When the grammar $\mathbb{G}$ is interpreted as PEG parser, $\mathbb{X}$ is thus a set of parsing procedures that the procedure e_1 in $A = e_1 | \, e_2$ may call, directly or indirectly, at the start of its input. When one of these procedures succeeds, $\mathbb{X} \asymp \mathbb{Y}$ ensures that e_2 cannot succeed on the same input. When all fail, (2) ensures that e_1 cannot succeed. The parser backtracks to try e_2. The parser can thus choose between e_1 and e_2 by examining the input within the reach of one parsing procedure from $\texttt{First}(e_1)$. Such grammars were in [7] referred to as LL(1P).

As an example, (8) interpreted as PEG parser can choose between A and B by looking at the input within the reach of one parsing procedure, C.

While all possible refinements $\mathbb{X}$ and $\mathbb{Y}$ satisfying (2) and (3) can be generated by a mechanical procedure, there is, in general, no such procedure to check them for (4), as we have there, in general, an intersection of context-free languages. Still, many simple cases can be automatically detected and handled.

6. Beyond LL(1P)

Suppose that grammar (8) is modified as follows:

$$\begin{array}{llll} S = X\$ & X = A|B & & \\ A = EF & F = CY & C = c^+ & E = a^+ \\ B = EG & G = DZ & D = d^+ & \end{array} \tag{9}$$

Here we have $\texttt{first}(A) = \texttt{first}(B)$, so no amount of refinement using Lemma 5.2 can produce $\mathbb{X} \asymp \mathbb{Y}$. However, we note that $\mathcal{L}(A) \subseteq \mathcal{L}(EC)\Sigma^*$, $\mathcal{L}(B) \subseteq \mathcal{L}(EG)\Sigma^*$, and $\mathcal{L}(EC)\Sigma^* \cap \mathcal{L}(ED)\Sigma^* = \varnothing$. With $\mathrm{Tail}(B) = \$$, the condition (1) is approximated as follows:

$$\mathcal{L}(A)\Sigma^* \cap \mathcal{L}(B)\$ \;\subseteq\; \mathcal{L}(EC)\Sigma^*\Sigma^* \cap \mathcal{L}(ED)\Sigma^*\$ \;\subseteq\; \mathcal{L}(EC)\Sigma^* \cap \mathcal{L}(ED)\Sigma^* = \varnothing,$$

showing that the grammar (9) is its own PEG parser. This parser chooses between A and B by looking at the input within the reach of two parsing procedures, E and C. We can call such grammar LL(2P).

Let $\mathcal{E}^2 = \{e_1 e_2 : e_1 \in \mathbb{E}, e_2 \in \mathbb{E}\}$. The definitions of $\mathcal{L}(\mathbb{X})$ and $L \sqsubseteq \mathbb{X}$ apply unchanged to $\mathbb{X} \subseteq \mathcal{E} \cup \mathcal{E}^2$, and conditions (2)–(4) still imply (1). Using this notation, we can state our observations above as $\mathcal{L}(A) \sqsubseteq EC$, $\mathcal{L}(B) \sqsubseteq ED$, and $\{EC\} \asymp \{ED\}$.

The question is how do we systematically find the sets $\mathbb{X}, \mathbb{Y} \subseteq \mathcal{E} \cup \mathcal{E}^2$ that satisfy (2)–(4). The calculation rules (5)–(7) remain valid, but Lemmas 5.3 and 5.2 apply to subsets of $\mathcal{E}$. We need an algorithm to calculate $\texttt{Follow}_2 \subseteq \mathcal{E}^2$ such that $\mathrm{Tail}(A) \sqsubseteq \texttt{Follow}_2(A)$ and one for refining the elements of $\mathcal{E}^2$. This is a subject of further research. An extension to LL(kP) seems natural.

7. Looking Farther Ahead

The property of PEG parser that enables it to look far ahead is backtracking. But the limited backtracking of PEG cannot be used to look ahead beyond e_1 in $e_1 | \, e_2$. This is illustrated by the following grammar,

borrowed from [5, 6]:

$$\begin{array}{ll} S = X\$ & \\ X = A|B & A = ab|C \\ B = a|Cd & C = c \end{array} \tag{10}$$

Interpreted as BNF, this grammar defines $\mathcal{L}(S) = \{ab\$, c\$, a\$, cd\$\}$. But interpreted as a PEG parser, it does not accept $cd\$$: A succeeds on c via C, X returns this success to S, which fails when it finds d instead of \$. Once A succeeded, there is no backtracking to try B; X cannot look beyond A when faced with c as the first letter. 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 $c\$$ for A and $a\$$ or cd for B.

The full PEG, as defined in [4], has an operation to look ahead as far as needed. It is the "and-predicate" $\&e$ that means: "invoke the expression e on the text ahead and backtrack; return success if e succeeded or failure if it failed". It can be inserted into (10) to look beyond A:

$$\begin{array}{ll} S = X\$ & \\ X = A\&\$|B & A = ab|C \\ B = a|Cd & C = c \end{array} \tag{11}$$

As the result, after A succeeding on the first c of $cd\$$, $\&\$$ fails on the following d, so $A\&\$$ fails and B is successfully tried.

This scheme is applied in [5, 6] to construct PEG parsers for strong LL(k) and LL-regular grammars. To study if it can be used for other classes of grammars, we replace each expression $A = e_1|\, e_2$ in the grammar $\mathbb{G}$ that cannot be verified to satisfy (1) with $A = e_1\&e_0|\, e_2$, where $e_0 \in \mathbb{E}$. We call the resulting grammar $\mathbb{G}'$, and define semantics of the new expressions by the additional inference rules shown in Figure 3. From now on, the relation $\overset{\text{PEG}}{\rightsquigarrow}$ applies to $\mathbb{G}'$, while $\overset{\text{BNF}}{\rightsquigarrow}$ still applies to $\mathbb{G}$.

$$\frac{A = e_1\&e_0|\, e_2 \quad [e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y \quad [e_0]\ y \overset{\text{PEG}}{\rightsquigarrow} z}{[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y} \quad \textbf{(choice.p3)}$$

$$\frac{A = e_1\&e_0|\, e_2 \quad [e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y}{[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y} \quad \textbf{(choice.p4)}$$

$$\frac{A = e_1\&e_0|\, e_2 \quad [e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} z \quad [e_0]\ z \overset{\text{PEG}}{\rightsquigarrow} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y}{[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y} \quad \textbf{(choice.p5)}$$

where Y denotes y or fail.

Figure 3. Extended PEG semantics

The problem is to choose the expression e_0. The obvious requirement is that the grammar $\mathbb{G}'$ should not be left-recursive. To specify this formally, we add one more rule to the definition of `first`:

– For $A \in N$ where $A = e_1\&e_0|\, e_2$, $\quad \texttt{first}(A) = \{e_0, e_1, e_2\}$.

The grammar $\mathbb{G}'$ is not left-recursive if $e \notin \texttt{first}^+(e)$ for all $e \in \mathbb{E}$ holds with `first` so adjusted. We need to make sure that Propositions 4.1 and 4.2 hold for the modified grammar.

Proposition 7.1. If the grammar $\mathbb{G}'$ is not left-recursive then for every $e \in \mathbb{E}$ and $w \in \Sigma^*$ there exists a proof of either $[e]\ w \overset{\text{PEG}}{\rightsquigarrow}$ fail or $[e]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$.

Proposition 7.2. For any $e \in \mathbb{E}$ and $x, y \in \Sigma^*$, $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ implies $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$, where $\overset{\text{BNF}}{\rightsquigarrow}$ applies to $\mathbb{G}$ and $\overset{\text{PEG}}{\rightsquigarrow}$ to $\mathbb{G}'$.

We can now state the requirements for e_0:

Proposition 7.3. If the grammar $\mathbb{G}'$ is not left-recursive and its every equation $A = e_1 \& e_0 |\, e_2$ satisfies:

$$\text{For each } w \in \text{Tail}(A), [e_0]\ w \overset{\text{PEG}}{\rightsquigarrow} x \text{ for some } x \in \Sigma^*, \tag{12}$$

$$\mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^* \cap \mathcal{L}(e_2)\,\text{Tail}(A) = \varnothing\ . \tag{13}$$

then $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ implies $[S]\ w \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$.

A systematic way of choosing a suitable e_0 is still to be found. A specific construction for strong LL(k) grammars is shown in [5, 6]. It results in $\mathcal{L}(e_0)$ consisting of k-letter words.
To close the subject, we note this limitation:

Proposition 7.4. An expression e_0 satisfying (12) and (13) may exist only if

$$\mathcal{L}(e_1)\,\text{Tail}(A) \cap \mathcal{L}(e_2)\,\text{Tail}(A) = \varnothing. \tag{14}$$

Appendix

Proof of Proposition 4.1

To be short, we shall say that "e handles w" if for every $w \in \Sigma^*$ there exists a proof of either $[e]\ w \overset{\text{PEG}}{\rightsquigarrow}$ fail or $[e]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$. To show that each $e \in \mathbb{E}$ handles every $w \in \Sigma^*$, we are use induction on the length of w. The induction base is proved below as Lemma 4.1.B and induction step as Lemma 4.1.S. Both are proved by induction on the size of $\texttt{First}(e)$. We use an observation that if the grammar $\mathbb{G}$ is not left-recursive, the set $\texttt{First}(e')$ where $e' \in \texttt{first}(e)$ has fewer elements than $\texttt{First}(e)$.

Lemma 4.1.S. Suppose $\mathbb{G}$ is not left-recursive, and each $e \in \mathbb{E}$ handles all words of length less than given $n > 0$. Then each $e \in \mathbb{E}$ also handles all words of length n.

Proof:
Is by induction on the size of $\texttt{First}(e)$. Consider a word w of length n.

(Induction base) Each e with $\texttt{First}(e)$ of size 0 handles w.
Such e is either $\boldsymbol{\varepsilon}$ or $a \in \Sigma$. From **empty.p**, **letter.p1**, and **letter.p2** follows, respectively, $[\boldsymbol{\varepsilon}]\ w \overset{\text{PEG}}{\rightsquigarrow} w$, $[a]\ w \overset{\text{PEG}}{\rightsquigarrow} x$ if $w = ax$, and $[a]\ w \overset{\text{PEG}}{\rightsquigarrow}$ fail otherwise.

(Induction step) Suppose each e with $\texttt{First}(e)$ of size less than $m > 0$ handles w. Consider e' with $\texttt{First}(e')$ of size m. By induction hypothesis, each $e \in \texttt{first}(e')$ handles w. Since $m > 0$, must be $e' = A \in N$. Two cases are possible:

(Case 1) $A = e_1 e_2$. We have $e_1 \in \texttt{first}(A)$, so e_1 handles w.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$, we have $[A]\ w \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$ by **seq.p2**.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} w$, we have $\varepsilon \in \mathcal{L}(e_1)$ by Proposition 4.2, so $e_2 \in \texttt{first}(A)$, and e_2 handles w. We have $[e_2]\ w \overset{\text{PEG}}{\rightsquigarrow} Z$ where Z is either fail or z where $w = yz$. This gives $[A]\ \varepsilon \overset{\text{PEG}}{\rightsquigarrow} Z$ by **seq.p1**.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$ and $x \neq \varepsilon$, y has length less than n and e_2 handles y according to the Lemma's assumption. We have $[e_2]\ y \overset{\text{PEG}}{\rightsquigarrow} Z$ where Z is either fail or z where $w = yz$. This gives $[A]\ \varepsilon \overset{\text{PEG}}{\rightsquigarrow} Z$ by **seq.p1**.

(Case 2) $A = e_1 | e_2$. We have $e_1 \in \texttt{first}(A)$ and $e_2 \in \texttt{first}(A)$, so each of e_1, e_2 handles w.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$, we have $[A]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ by **choice.p1**.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$, we have $[e_2]\ w \overset{\text{PEG}}{\rightsquigarrow} Y$ where Y is either fail or y where $w = yz$. This gives $[A]\ w \overset{\text{PEG}}{\rightsquigarrow} Y$ by **choice.p2**.

In each case, $e' = A$ handles w. □

Lemma 4.1.B. If $\mathbb{G}$ is not left-recursive, each $e \in \mathbb{E}$ handles the word of length 0.

Proof:
The proof is almost identical to that of Lemma 4.1.S if we take $w = \varepsilon$ and use **letter.p3** instead of **letter.p1**, and **letter.p2** in the induction base. The third item in Case 1 is not applicable. □

Proof of Proposition 4.2

We spell out the proof sketched in [6]. It is by induction on the height of the proof tree.

(Induction base) Suppose the proof of $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ has height 1. Then it has to be the proof of $[\boldsymbol{\varepsilon}]\ x \overset{\text{PEG}}{\rightsquigarrow} x$ or $[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} x$ using **empty.p** or **letter.p1**, respectively. But then, $[\boldsymbol{\varepsilon}]\ x \overset{\text{BNF}}{\rightsquigarrow} x$ respectively $[a]\ ax \overset{\text{BNF}}{\rightsquigarrow} x$ by **empty.b** or **letter.b**.

(Induction step) Assume that for every proof tree for $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ of height $n \geq 1$ there exists a proof of $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$. Consider a proof tree for $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ of height $n+1$. Its last step must be one of these:
- $[A]\ xyz \overset{\text{PEG}}{\rightsquigarrow} x$ derived from $A = e_1 e_2$, $[e_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz$, and $[e_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} z$ using **seq.p1**.
  By induction hypothesis, $[e_1]\ xyz \overset{\text{BNF}}{\rightsquigarrow} yz$ and $[e_2]\ yz \overset{\text{BNF}}{\rightsquigarrow} z$, so $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} x$ follows from **seq.b**.
- $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} x$ derived from $A = e_1 e_2$ and $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using **choice.p1**.
  By induction hypothesis, $[e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$, so $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} x$ follows from **choice.b1**.
- $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} x$ derived from $A = e_1 e_2$ and $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$, $[e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$, using **choice.p2**.
  By induction hypothesis, $[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$, so $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} x$ follows from **choice.b2**.

Proof of Proposition 4.3

Assume $\mathbb{G}$ satisfies the stated conditions. Take any w such that $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$. We are going to show that for each result $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ in the proof tree of $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ there exists a proof of $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$. We show it using induction on the height of the proof tree. The "result" here means both partial result and the final $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$.

(Induction base) Suppose the proof of $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ has height 1. Then it has to be the proof of $[\boldsymbol{\varepsilon}]\ x \overset{\text{BNF}}{\rightsquigarrow} x$ or $[a]\ ax \overset{\text{BNF}}{\rightsquigarrow} x$ using **empty.b** or **letter.b**, respectively. But then, $[\boldsymbol{\varepsilon}]\ x \overset{\text{PEG}}{\rightsquigarrow} x$ respectively $[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} x$ by **empty.p** or **letter.p1**.

(Induction step) Assume that for every result $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ that has proof tree of height $n \geq 1$ there exists a proof of $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$. Consider a result $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ with proof tree of height $n+1$. Its last step must be one of these:

- $[A]\ xyz \overset{\text{BNF}}{\rightsquigarrow} z$ derived from $A = e_1 e_2$, $[e_1]\ xyz \overset{\text{BNF}}{\rightsquigarrow} yz$, and $[e_2]\ yz \overset{\text{BNF}}{\rightsquigarrow} z$ using **seq.b**. By induction hypothesis, $[e_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz$ and $[e_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} z$, so $[A]\ xyz \overset{\text{PEG}}{\rightsquigarrow} z$ follows from **seq.p1**.
- $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ derived from $A = e_1 / e_2$ and $[e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b1**. By induction hypothesis, $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$, so $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ follows from **choice.p1**.
- $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$, derived from $A = e_1 / e_2$ and $[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b2**. By induction hypothesis, $[e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$. But, to use **choice.p2** we also need to verify that $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$.
  Suppose that there is no proof of $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$. Then, according to Proposition 4.1, there exists a proof of $[e_1]\ uv \overset{\text{PEG}}{\rightsquigarrow} v$ where $uv = xy$. According to Proposition 4.2, there exists a proof of $[e_1]\ uv \overset{\text{BNF}}{\rightsquigarrow} v$, so $u \in \mathcal{L}(e_1)$. From $[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ follows $x \in \mathcal{L}(e_2)$. As $A \neq S$, we have $y \in \text{Tail}(A)$. From $v \in \Sigma^*$ and $uv = xy$ follows $\mathcal{L}(e_1) \cap \mathcal{L}(e_2)\,\text{Tail}(A) \neq \varnothing$, which contradicts (1). We must thus conclude that there exists a proof of $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$, so there exists a proof of $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using **choice.p2**.

Proof of Proposition 5.1

Assume $\varepsilon \notin \mathcal{L}(e_1)$ and $\mathbb{X}, \mathbb{Y}$ as stated by (2)–(4). We have $\mathcal{L}(e_1) = \mathcal{L}(e_1) - \varepsilon \subseteq (\mathcal{L}(\mathbb{X}) - \varepsilon)\Sigma^*$.
Each word in $\text{Tail}(A)$ ends with \$, so $\mathcal{L}(e_2)\,\text{Tail}(A) = \mathcal{L}(e_2)\,\text{Tail}(A)) - \varepsilon \subseteq (\mathcal{L}(\mathbb{Y}) - \varepsilon)\Sigma^*$.
This gives $\mathcal{L}(e_1)\Sigma^* \cap \mathcal{L}(e_2)\,\text{Tail}(A) \subseteq (\mathcal{L}(\mathbb{X}) - \varepsilon)\Sigma^* \cap (\mathcal{L}(\mathbb{Y}) - \varepsilon)\Sigma^* = \varnothing$.

Proof of Lemma 5.2

We note that $\mathcal{L}(A) \sqsubseteq \texttt{first}(A)$:

- If $A = e_1 / e_2$, we have $\mathcal{L}(A) = \mathcal{L}(\texttt{first}(A) \sqsubseteq \texttt{first}(A)$ from $\mathcal{L}(\mathbb{X}) \sqsubseteq \mathbb{X}$.
- If $A = e_1\, e_2$ and $\varepsilon \notin e_1$, we have $\mathcal{L}(A) = \mathcal{L}(e_1)\mathcal{L}(e_2) \sqsubseteq \{e_1\} = \texttt{first}(A)$ from (6).
- If $A = e_1\, e_2$ and $\varepsilon \in e_1$, we have $\mathcal{L}(A) = \mathcal{L}(e_1)\mathcal{L}(e_2) \sqsubseteq \{e_1, e_2\} = \texttt{first}(A)$ from (7).

Using the definition of $\sqsubseteq$ one can verify that $L \sqsubseteq \mathbb{A}\ \&\ \mathcal{L}(\mathbb{A}) \sqsubseteq \mathbb{B}\ \Rightarrow\ L \sqsubseteq \mathbb{B}$ holds for $\mathbb{A}, \mathbb{B} \subseteq \mathcal{E}$.
With $\mathbb{A} = \{A\} \cup \mathbb{X}$ and $\mathbb{B} = \texttt{first}(A) \cup \mathbb{X}$, this gives $L \sqsubseteq A\ \Rightarrow\ L \sqsubseteq \texttt{first}(A) \cup \mathbb{X}$ with the help of (5) and $\mathcal{L}(\mathbb{X}) \sqsubseteq \mathbb{X}$.

Proof of Lemma 5.3

Consider some $A \in N$ and $y \in \text{Tail}(A)$. By definition, there is a proof of $[S]\ w \overset{\text{BNF}}{\rightsquigarrow} \varepsilon$ that contains $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ as one of the partial results. This partial result must be used in a subsequent derivation. This derivation can only result in one of the following:

(a) $[A_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ where $A_1 = A / e$ from $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b1**.
(b) $[A_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ where $A_1 = e / A$ from $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b2**.
(c) $[A_1]\ zxy \overset{\text{BNF}}{\rightsquigarrow} y$ where $A_1 = e\, A$ from $[e]\ zxy \overset{\text{BNF}}{\rightsquigarrow} xy$ using **seq.b**.
(d) $[A_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ where $A_1 = A\, e$ from $[e]\ \varepsilon y \overset{\text{BNF}}{\rightsquigarrow} y$ using **seq.b**.
(e) $[A_1]\ xy'y'' \overset{\text{BNF}}{\rightsquigarrow} y''$ where $A_1 = A\, B$, $y'y'' = y$ and $y' \neq \varepsilon$ from $[B]\ y'y'' \overset{\text{BNF}}{\rightsquigarrow} y''$ using **seq.b**.

In each of the cases (a)-(d), the result is similar to the original one, and the alternative derivations (a)-(e) apply again. We may have a chain of derivations (a)-(d), but it must end with (e) as y must eventually be reduced. We have thus in general a sequence of steps of this form:

$$\frac{[A_0]\ xy \overset{\text{BNF}}{\rightsquigarrow} y \quad \ldots \quad \text{as in (a)-(d)}}{\frac{[A_1]\ x_1 y \overset{\text{BNF}}{\rightsquigarrow} y \quad \ldots \quad \text{as in (a)-(d)}}{\frac{[A_2]\ x_2 y \overset{\text{BNF}}{\rightsquigarrow} y \quad \ldots \quad \text{as in (a)-(d)}}{\ldots \qquad \frac{}{\frac{[A_{n-1}]\ x_{n-1} y \overset{\text{BNF}}{\rightsquigarrow} y \quad \ldots \quad \text{as in (a)-(d)}}{A_{n+1} = A_n B \quad [A_n]\ x_n y \overset{\text{BNF}}{\rightsquigarrow} y \quad [B]\ y'y'' \overset{\text{BNF}}{\rightsquigarrow} y''}}}}}$$
$$[A_{n+1}]\ x_n y'y'' \overset{\text{BNF}}{\rightsquigarrow} y''$$

where $A_0 = A$ and $n \geq 0$. We have $A_1 \in \texttt{Last}(A_0)$, $A_2 \in \texttt{Last}(A_1)$, etc., $A_n \in \texttt{Last}(A_{n-1})$, and $B \in \texttt{Next}(A_n)$, which means $B \in \texttt{Follow}(A)$. From $[B]\ y'y'' \overset{\text{BNF}}{\rightsquigarrow} y''$ we have $y' \in \mathcal{L}(B)$; since $y = y'y''$ and $y' \neq \varepsilon$, we have $\mathcal{L}(B) - \varepsilon\Sigma^* \subseteq \mathcal{L}(\text{Tail}(A))$.

Proof of Proposition 7.1

The proof is the same as that of Proposition 4.1, with the difference that $\overset{\text{PEG}}{\rightsquigarrow}$ applies to $\mathbb{G}'$, `first` is extended by the additional rule, and the following case added in the proof of Lemma 4.1.S:
(Case 3) $A = e_1 \& e_0 |\, e_2$. We have $e_0, e_1, e_2 \in \texttt{first}(A)$, so each of e_0, e_1, e_2 handles w.

- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$ and $[e_0]\ y \overset{\text{PEG}}{\rightsquigarrow} z$, we have $[A]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ by **choice.p3**.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow} y$ where $w = xy$ and $[e_0]\ y \overset{\text{PEG}}{\rightsquigarrow}$ fail, we have, by **choice.p4**, $[e_2]\ w \overset{\text{PEG}}{\rightsquigarrow} Y$ where Y is either fail or y where $w = yz$.
- If $[e_1]\ w \overset{\text{PEG}}{\rightsquigarrow}$ fail, we have $[e_2]\ w \overset{\text{PEG}}{\rightsquigarrow} Y$ we have, by **choice.p5**, $[e_2]\ w \overset{\text{PEG}}{\rightsquigarrow} Y$.

Proof of Proposition 7.2

The proof is the same as that of Proposition 4.3 with the difference that $\overset{\text{PEG}}{\rightsquigarrow}$ applies to $\mathbb{G}'$, and an additional case where e in the last step in the proof of $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ is the expression $A = e_1 \& e_0 |\, e_2$ of $\mathbb{G}'$. There are two possibilities:

(1) $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ is derived from $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using **choice.p3**. (The proof for e_0 is irrelevant.) By induction hypothesis there exists a proof of $[e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$. We have $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ from **choice.b1**.

(2) $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ is derived from $[e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ using **choice.p4** or **choice.p5**. (The proofs for e_1 and e_0 are irrelevant.) By induction hypothesis there exists a proof of $[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$. We have $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ from **choice.b2**.

Proof of Proposition 7.3

Recall that $\overset{\text{PEG}}{\rightsquigarrow}$ applies to $\mathbb{G}'$, while $\overset{\text{BNF}}{\rightsquigarrow}$ applies to $\mathbb{G}$. The proof is identical to that of Proposition 4.3 except for the case where e in the last step in the proof of $[e]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ is the expression $A = e_1 | e_2$ that is replaced by $A = e_1 \& e_0 | e_2$ in $\mathbb{G}'$. (In the case where $A = e_1 | e_2$ is not changed in $\mathbb{G}'$, the proof of Proposition 4.3 still applies, as the expression as assumed to satisfy (1).) Two cases are possible:

(1) $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ is derived from $[e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b1**. By induction hypothesis there exists a proof of $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$. As $A \neq S$, we have $y \in \text{Tail}(A)$. According to (12), we have $[e_0]\ y \overset{\text{PEG}}{\rightsquigarrow} z$ for some z. We have $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ from **choice.p3**.

(2) $[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ is derived from $[e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y$ using **choice.b2**. By induction hypothesis there exists proof of $[e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$. If $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow}$ fail, we have $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ from **choice.p4**.

If e_1 does not fail on xy then by Proposition 7.1 exists a proof of $[e_1]\ uvw \overset{\text{PEG}}{\rightsquigarrow} vw$ for some $uvw = xyz$. By Proposition 7.2 exists a proof of $[e_1]\ uvw \overset{\text{BNF}}{\rightsquigarrow} vw$, which means $u \in \mathcal{L}(e_1)$. Suppose e_0 does not fail on uv. Then, by Proposition 7.1 exists a proof of $[e_0]\ st \overset{\text{PEG}}{\rightsquigarrow} t$ where $st = uv$. By Proposition 7.2 exists a proof of $[e_0]\ st \overset{\text{BNF}}{\rightsquigarrow} t$, which means $s \in \mathcal{L}(e_0)$. Thus $xyz = uvw = ust \in \mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^*$. But $[e_2]\ xyz \overset{\text{BNF}}{\rightsquigarrow} yz$ means $xyz \in \mathcal{L}(e_2)\,\text{Tail}(A)$, which contradicts (13); thus, $[e_0]\ vw \overset{\text{PEG}}{\rightsquigarrow}$ fail, and we have $[A]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz$ from **choice.p5**.

Proof of Proposition 7.4

Suppose (14) is false, so there exist $u \in \mathcal{L}(e_1), v \in \text{Tail}(A), x \in \mathcal{L}(e_2), y \in \text{Tail}(A)$ such that $uv = xy$. Suppose there exists e_0 satisfying (12) and (13). According to (12) exist $w, z \in \Sigma^*$ such that $wz = v$ and $[e_0]\ wz \overset{\text{PEG}}{\rightsquigarrow} z$. By Proposition 4.2, $w \in \mathcal{L}(e_0)$, which means $uv = uwz \in \mathcal{L}(e_1)\mathcal{L}(e_0)\Sigma^*$. As $uwz = xy \in \mathcal{L}(e_2)\,\text{Tail}(A)$, this contradicts (13).

References

[1] Aho, A. V., Sethi, R., Ullman, J. D.: *Compilers. Principles, Techniques, and Tools,* Addison-Wesley, 1987.

[2] Ford, B.: *Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking,* Master Thesis, Massachusetts Institute of Technology, September 2002, `http://pdos.csail.mit.edu/papers/packrat-parsing:ford-ms.pdf`.

[3] Ford, B.: Packrat parsing: simple, powerful, lazy, linear time, functional pearl, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002* (M. Wand, S. L. P. Jones, Eds.), ACM, 2002.

[4] Ford, B.: Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004* (N. D. Jones, X. Leroy, Eds.), ACM, Venice, Italy, 14–16 January 2004.

[5] Mascarenhas, F., Medeiros, S., Ierusalimschy, R.: *On the Relation between Context-Free Grammars and Parsing Expression Grammars,* Technical report, UFRJ Rio de Janeiro, UFS Aracaju, PUC-Rio, Brazil, 2013, `http://arxiv.org/pdf/1304.3177v1`.

[6] Medeiros, S.: *Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto*, Ph.D. Thesis, Pontifícia Universidade Católica do Rio de Janeiro, August 2010,
http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0611957_10_pretextual.pdf
http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0611957_10_cap_01.pdf
http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0611957_10_cap_02.pdf
etc.
http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0611957_10_cap_05.pdf
http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0611957_10_postextual.pdf.

[7] Redziejowski, R. R.: From EBNF to PEG, *Fundamenta Informaticae*, **128**, 2013, 177–191.

[8] Tremblay, J.-P., Sorenson, P. G.: *The Theory and Practice of Compiler Writing*, McGraw-Hill, 1985.