From EBNF to PEG

Roman R. Redziejowski

Giraf's Research
roman.redz@swipnet.se

Abstract. Parsing Expression Grammar (PEG) is a way to define a recursive-descent parser with limited backtracking. Its properties are useful in many applications. In spite of its apparent similarity to Extended Backus-Naur Form (EBNF), PEG often defines quite a different language, sometimes difficult to describe exactly. However, a recent result by Medeiros shows that an EBNF grammar having the LL(1) property can be transcribed verbatim into a PEG that not only defines the same language, but also parses the input in exactly the same way. We show that such transcription is possible for a wider class of EBNF grammars, which is interesting because the backtracking of PEG is often a convenient way to circumvent just the LL(1) restriction.

1 Introduction

Parsing Expression Grammar (PEG), as introduced by Ford [1, 2], is a way to define 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 usually imposed by top-down parsers. These properties are useful in many applications. However, PEG is not well understood as a language specification tool. In [3], the author tried to find the language defined by a given parsing expression. Unfortunately, it was only possible to obtain some approximations that became more and more difficult as the complexity of the expression increased.

Instead of trying to find out what a given PEG does, one can try to construct a PEG parser recognizing a given language. This approach has been explored by Medeiros in [4]. He shows the construction of PEGs for languages defined by regular expressions, as well as right-linear, LL(1) and LL(k) context-free grammars. As shown there, any context-free grammar with LL(1) property can be directly interpreted as PEG, with the unordered choice "`|`" replaced by the ordered choice "`/`". Unfortunately, this is not of much use when we employ PEG just in order to circumvent the LL(1) restriction. But it turns out that the result from [4] can be extended to a wider class of grammars. As an example, take the following EBNF grammar:

Literal → *DecimalLiteral* `|` *BinaryLiteral*
DecimalLiteral → `[0-9]`* `"."` `[0-9]`*
BinaryLiteral → `[01]`$^+$ `"B"`

This grammar is not LL(1), and not even LL(k) for any k: both *DecimalLiteral* and *BinaryLiteral* may start with any number of zeros and ones. A classical top-down parser constructed from this grammar cannot choose between these

alternatives of *Literal* by looking at any predefined number of characters ahead. But, treated as a PEG parser, this grammar recognizes exactly the language defined by its EBNF interpretation.

To see how it happens, suppose the PEG parser is presented with the input "`10B`". It performs a series of calls *Literal* → *DecimalLiteral* → `[0-9]`*, consuming "`10`", after which it fails to recognize "`.`", backtracks, and proceeds to try *BinaryLiteral*. Parsing procedure for the nonterminal *DecimalLiteral* has here the same overall effect as if it was a terminal: it explores the input ahead, and either consumes the recognized portion, or backtracks and in effect does nothing. In fact, the grammar above would be LL(1) if *DecimalLiteral* and *BinaryLiteral* were terminals, corresponding to tokens produced by a lexer.

In the following, we try to answer this question: given an EBNF grammar, when can I transcribe it directly into PEG and obtain a correct parser?

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 $y \in Y$.

For $x, y \in \Sigma^*$, x is a *prefix* of y if $y = xu$ for some $u \in \Sigma^*$. We write $x \leq y$ to mean that x is a prefix of y. For $X \subseteq \Sigma^*$, the set of all prefixes of $x \in X$ is denoted by $\mathrm{Pref}(X)$.

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 and reflexive 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 EBNF. The grammar is a set of rules of the form $A \mapsto e$, where e is a *syntax expression* and A is the name given to it. The expression e has one of these forms:

- $e_1\, e_2$ (*Sequence*),
- $e_1 | e_2$ (*Choice*),

where each of e_1, e_2 is an expression name, a letter from Σ, or ε. The set of all names appearing in the rules is denoted by N. For $A \in N$, $e(A)$ is the expression e in $A \mapsto e$. We define $\mathbb{E} = N \cup \Sigma \cup \varepsilon$.

When $\mathbb{G}$ is interpreted as PEG, the rule $A \mapsto e$ is a definition $A \leftarrow e$ of parsing expression e, and $e_1|e_2$ is the ordered choice. When $\mathbb{G}$ is interpreted as EBNF, the rule is a production $A \to e$, and $e_1|e_2$ is the unordered choice.
For obvious reasons, we assume $\mathbb{G}$ to be free from left recursion.

This grammar is reduced to bare bones in order to simplify the reasoning. Any full EBNF grammar or PEG without predicates can be reduced to such form by steps like these:

– Replacing $[a_1\text{-}a_n]$ by $a_1|a_2|\ldots|a_n$;
– Rewriting $e_1e_2e_3$ and $e_1|e_2|e_3$ as $e_1(e_2e_3)$ respectively $e_1|(e_2|e_3)$ and introducing new names for expressions in parentheses;
– Replacing e^+ by ee^*;
– Replacing e^* by $A \mapsto eA \,|\, \varepsilon$.

3.1 The PEG Interpretation

When $\mathbb{G}$ is interpreted as PEG, the elements of $\mathbb{E}$ are parsing procedures that can call each other recursively. In general, parsing procedure is applied to a string from Σ^* at a position indicated by some "cursor". It tries to recognize a portion of the string ahead of the cursor. If it succeeds, it "consumes" the recognized portion by advancing the cursor and returns "success"; otherwise, it returns "failure" and does not consume anything (does not advance the cursor). 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.
– $A \mapsto e_1\, e_2$: Call e_1. If it succeeded, call e_2 and return success if e_2 succeeded. If e_1 or e_2 failed, reset cursor as it was before the invocation of e_1 and return failure.
– $A \mapsto e_1|e_2$: 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.

Backtracking occurs in the Sequence expression. If e_1 succeeds and consumes some input, and then e_2 fails, the cursor is moved back to where it was before trying e_1. If this Sequence was called as the first alternative in a Choice, Choice has an opportunity to try another alternative on the same input. However, the backtracking is limited: once e_1 in the Choice $e_1|e_2$ succeeded, e_2 will never be tried on the same input, even if the parse fails later on.

Following [4], we define formally the action of a parsing procedure with the help of relation $\overset{\text{PEG}}{\rightsquigarrow} \subseteq \mathbb{E} \times \Sigma^* \times \{\Sigma^* \cup \text{fail}\}$. For $e \in \mathbb{E}$ and $x, y \in \Sigma^*$:

– $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ means: "e applied to input xy consumes x and returns success",
– $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}$ means: "e applied to input x returns failure".

The relation itself is defined using the method of "natural semantics": $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} X$ holds if and only if it can be proved using the inference rules in Figure 1. The rules come from [4], but were modified by integrating the step $e(A) \Rightarrow A$ into the last four rules.

The proof tree of $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} X$, when followed from bottom up, mimics procedure calls in the process of parsing the string x. Rule **seq.1p** with $Z = \text{fail}$ corresponds to backtracking in Sequence, and rule **choice.2p** to calling e_2 after e_1 failed.

According to Lemma 2.3.1 in [4], a proof of $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} X$ exists if and only if there exists the corresponding derivation $\Rightarrow_G$ in the formal definition of PEG given by Ford in [2]. According to [2], such derivation exists if the grammar is "well-formed". Grammar $\mathbb{G}$ is well-formed because it contains neither left recursion nor iteration. This gives the following fact:

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

$$\textbf{(empty.1p)}\quad \frac{}{[\varepsilon]\ x \overset{\text{PEG}}{\rightsquigarrow} x} \qquad \textbf{(letter.1p)}\quad \frac{}{[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} x}$$

$$\textbf{(letter.2p)}\quad \frac{b \neq a}{[b]\ ax \overset{\text{PEG}}{\rightsquigarrow} \text{fail}} \qquad \textbf{(letter.3p)}\quad \frac{}{[a]\ \varepsilon \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}$$

$$\textbf{(seq.1p)}\quad \frac{[e_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz \quad [e_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} Z \quad e(A) = e_1 e_2}{[A]\ xyz \overset{\text{PEG}}{\rightsquigarrow} Z}$$

$$\textbf{(seq.2p)}\quad \frac{[e_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail} \quad e(A) = e_1 e_2}{[A]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail}}$$

$$\textbf{(choice.1p)}\quad \frac{[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y \quad e(A) = e_1 | e_2}{[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} y}$$

$$\textbf{(choice.2p)}\quad \frac{[e_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y \quad e(A) = e_1 | e_2}{[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} Y}$$

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

Fig. 1. PEG semantics

3.2 The CFG Interpretation

The grammar $\mathbb{G}$ with "$\mapsto$" interpreted as "$\rightarrow$" corresponds quite exactly to the traditional Context-Free Grammar (CFG), so we shall speak of its interpretation as CFG. Traditionally, CFG is a mechanism for generating words:

- ε generates empty word.
- $a \in \Sigma$ generates itself.
- $A \mapsto e_1 e_2$ generates any word generated by e_1 followed by any word generated by e_2.
- $A \mapsto e_1 | e_2$ 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.

$$\textbf{(empty.1c)}\quad \frac{}{[\varepsilon]\ x \overset{\text{CFG}}{\rightsquigarrow} x} \qquad \textbf{(letter.1c)}\quad \frac{}{[a]\ ax \overset{\text{CFG}}{\rightsquigarrow} x}$$

$$\textbf{(seq.1c)}\quad \frac{[e_1]\ xyz \overset{\text{CFG}}{\rightsquigarrow} yz \quad [e_2]\ yz \overset{\text{CFG}}{\rightsquigarrow} z \quad e(A) = e_1 e_2}{[A]\ xyz \overset{\text{CFG}}{\rightsquigarrow} z}$$

$$\textbf{(choice.1c)}\quad \frac{[e_1]\ xy \overset{\text{CFG}}{\rightsquigarrow} y \quad e(A) = e_1 | e_2}{[A]\ xy \overset{\text{CFG}}{\rightsquigarrow} y}$$

$$\textbf{(choice.2c)}\quad \frac{[e_2]\ xy \overset{\text{CFG}}{\rightsquigarrow} y \quad e(A) = e_1 | e_2}{[A]\ xy \overset{\text{CFG}}{\rightsquigarrow} y}$$

Fig. 2. CFG semantics

Following [4], we define formally the meaning of a CFG using relation $\overset{\text{CFG}}{\rightsquigarrow}$ similar to $\overset{\text{PEG}}{\rightsquigarrow}$. The relation is defined in the way similar to $\overset{\text{PEG}}{\rightsquigarrow}$: $[e]\ x \overset{\text{CFG}}{\rightsquigarrow} y$ holds

if and only if it can be proved using the inference rules in Figure 2. Again, these rules come from [4] and were modified by integrating the step $e(A) \Rightarrow A$ into the last three.

The proof tree of $[e]\ x \overset{\text{CFG}}{\rightsquigarrow} y$, when followed from bottom up, mimics procedure calls in an imagined recursive-descent parser processing the string x. The procedures correspond to elements of $\mathbb{E}$, and the procedure for Choice always chooses the correct alternative, either by full backtracking or by an oracle.

It can be verified that $\mathcal{L}(e) = \{x \in \Sigma^* : [e]\ xy \overset{\text{CFG}}{\rightsquigarrow} y \text{ for some } y \in \Sigma^*\}$. It can also be verified that if $x \in \mathcal{L}(e)$, $[e]\ xy \overset{\text{CFG}}{\rightsquigarrow} y$ holds for every $y \in \Sigma^*$.

4 When Are the Two Interpretations Equivalent?

The following has been proved as Lemma 4.3.1 in [4]:

Proposition 1. *For any $e \in \mathbb{E}$ and $x, y \in \Sigma^*$, $[e]\ xy \overset{PEG}{\rightsquigarrow} y$ implies $[e]\ xy \overset{CFG}{\rightsquigarrow} y$.*

Proof. We spell out the proof sketched in [4]. It is by induction on the height of the proof tree.

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

(Induction step) Assume that for every proof tree for $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ of height $n \geq 1$ exists a proof of $[e]\ xy \overset{\text{CFG}}{\rightsquigarrow} y$. 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 $[e_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} yz$, $[e_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} z$, and $e(A) = e_1 e_2$ using **seq.1p**. By induction hypothesis, $[e_1]\ xyz \overset{\text{CFG}}{\rightsquigarrow} yz$ and $[e_2]\ yz \overset{\text{CFG}}{\rightsquigarrow} z$, so $[A]\ xy \overset{\text{CFG}}{\rightsquigarrow} x$ follows from **seq.1c**.
- $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} x$ derived from $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ and $e(A) = e_1 e_2$ using **choice.1p**. By induction hypothesis, $[e_1]\ xy \overset{\text{CFG}}{\rightsquigarrow} y$, so $[A]\ xy \overset{\text{CFG}}{\rightsquigarrow} x$ follows from **choice.1c**.
- $[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} x$ derived from $[e_1]\ xy \overset{\text{PEG}}{\rightsquigarrow}$ fail, $[e_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$, and $e(A) = e_1 e_2$ using **choice.2p**. By induction hypothesis, $[e_2]\ xy \overset{\text{CFG}}{\rightsquigarrow} y$, so $[A]\ xy \overset{\text{CFG}}{\rightsquigarrow} x$ follows from **choice.2c**. □

Known examples show that the reverse of Proposition 1 is, in general, not true. We intend to formulate some conditions under which the reverse does hold. The problem is complicated by another known fact, namely that $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ does not imply $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y'$ for every $y' \in \Sigma^*$: the action of e depends on the string ahead. We have to consider the action of e in relation to the entire input string.

Let \$, the end-of-text marker, be a letter that does not appear in any rules. Define some $S \in N$ as the starting rule of the grammar. As $\mathcal{L}(e)$ does not depend on the input ahead, we have $[S]\ w\$ \overset{\text{CFG}}{\rightsquigarrow} \$$ if and only if $w \in \mathcal{L}(S)$. We note that every partial result in the proof tree of $[S]\ w\$ \overset{\text{CFG}}{\rightsquigarrow} \$$ has the form $[e]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ for some $e \in \mathbb{E}$ and $x, y \in \Sigma^*$.

For $A \in N$, define Tail(A) to be any string that appears ahead of the "CFG parser" just after it completed the call to A while processing some $w \in \mathcal{L}(S)$. Formally, $y\$ \in \text{Tail}(A)$ if there exists a proof tree for $[S]\ w\$ \overset{\text{CFG}}{\rightsquigarrow} \$$ that contains $[A]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ as a partial result.

Proposition 2. *If every Choice* $A \mapsto e_1|e_2 \in \mathbb{G}$ *satisfies*

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

there exists a proof of $[S]\ w\$ \overset{PEG}{\rightsquigarrow} \$$ *for each* $w \in \mathcal{L}(S)$*. Moreover, for every partial result* $[e]\ xy\$ \overset{CFG}{\rightsquigarrow} y\$$ *in the proof tree of* $[S]\ w\$ \overset{CFG}{\rightsquigarrow} \$$ *there exists a proof of* $[e]\ xy\$ \overset{PEG}{\rightsquigarrow} y\$$.

Proof. Take any $w \in \mathcal{L}(S)$. It means there exists proof tree for $[S]\ w\$ \overset{\text{CFG}}{\rightsquigarrow} \$$. We are going to show that for each partial result $[e]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ in that tree there exists a proof of $[e]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} \$$. We show it using induction on the height of the proof tree.

(Induction base) Suppose the proof of the partial result $[e]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ consists of one step. Then it has to be the proof of $[\varepsilon]\ x\$ \overset{\text{CFG}}{\rightsquigarrow} x\$$ or $[a]\ ax\$ \overset{\text{CFG}}{\rightsquigarrow} x\$$ using **empty.1c** or **letter.1c**, respectively. But then, $[\varepsilon]\ x\$ \overset{\text{PEG}}{\rightsquigarrow} x\$$ respectively $[a]\ ax\$ \overset{\text{PEG}}{\rightsquigarrow} x\$$ by **empty.1p** or **letter.1p**.

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

- $[A]\ xyz\$ \overset{\text{CFG}}{\rightsquigarrow} z\$$ derived from $[e_1]\ xyz\$ \overset{\text{CFG}}{\rightsquigarrow} yz\$$, $[e_2]\ yz\$ \overset{\text{CFG}}{\rightsquigarrow} z\$$, and $e(A) = e_1e_2$ using **seq.1c**. 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.1p**.
- $[A]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ derived from $[e_1]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ and $e(A) = e_1|e_2$ using **choice.1c**. By induction hypothesis, $[e_1]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$, so $[A]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$ follows from **choice.1p**.
- $[A]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$, derived from $[e_2]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ and $e(A) = e_1|e_2$ using **choice.2c**. By induction hypothesis, $[e_2]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$. But, to use **choice.2p** we also need to verify that $[e_1]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow}$ fail.
  Assume that (1) holds for A and suppose that there is no proof of $[e_1]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow}$ fail. According to Lemma 1, there exists a proof of $[e_1]\ uv\$ \overset{\text{PEG}}{\rightsquigarrow} v\$$ where $uv\$ = xy\$$. According to Proposition 1, there exists a proof of $[e_1]\ uv\$ \overset{\text{CFG}}{\rightsquigarrow} v\$$, so $u \in \mathcal{L}(e_1)$. From $[e_2]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ follows $x \in \mathcal{L}(e_2)$. From $[A]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ follows $y\$ \in \mathrm{Tail}(A)$. From $uv\$ = xy\$$ follows $u \leq xy$, so $u \in \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A))$, which contradicts (1). We must thus conclude that there exists a proof of $[e_1]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow}$ fail, so there exists a proof of $[A]\ xy\$ \overset{\text{PEG}}{\rightsquigarrow} y\$$ using **choice.2p**.

The proof of $[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$ exists as a special case of the above. □

The consequence of Propositions 1 and 2 is:

Proposition 3. *The two interpretations of* $\mathbb{G}$ *are equivalent if every Choice* $A \mapsto e_1|e_2 \in \mathbb{G}$ *satisfies (1). They are equivalent not only by accepting the same language, but also by parsing the input in the same way.*

Note that the condition is sufficient, but not necessary. Unfortunately, (1) is not easy to check. The languages appearing in it are context-free languages. The problems of inclusion and emptiness of intersection are in general undecidable

for these languages, so one has to consider special cases, or use a conservative approximation. In fact, condition (1) is identical to the "general semi disjointness" considered by Schmitz in [6]; he checks it using an own grammar approximation method.

As it will be seen, the LL(1) property implies (1), which accounts for the result obtained in [4]. We are going to look for a weaker property that still implies (1). One can think of the LL(1) property as approximating words by their one-letter prefixes. We generalize this to approximating words by prefixes of arbitrary lengths.

5 Prefix Covers

We say that $Y \subseteq \Sigma^*$ is a *prefix cover* of $X \subseteq \Sigma^*$, and write $Y \sqsubseteq X$ if each nonempty word in X has a nonempty prefix in Y.

Lemma 2. *For any $X, Y, Z \subseteq \Sigma^*$:*

(a) If $X \sqsubseteq Y$ and $Y \sqsubseteq Z$ then $X \sqsubseteq Z$.
(b) If $\mathcal{P}_X \sqsubseteq X$ and $\mathcal{P}_Y \sqsubseteq Y$ then $\mathcal{P}_X \cup \mathcal{P}_Y \sqsubseteq X \cup Y$.
(c) If $\varepsilon \notin X$ and $\mathcal{P}_X \sqsubseteq X$ then $\mathcal{P}_X \sqsubseteq XY$.
(d) If $\varepsilon \in X$, $\mathcal{P}_X \sqsubseteq X$, and $\mathcal{P}_Y \sqsubseteq Y$ then $\mathcal{P}_X \cup \mathcal{P}_Y \sqsubseteq XY$.

Proof. (a) Follows from the transitivity of $\leq$.
(b) If $x \neq \varepsilon$ is in X, it has a nonempty prefix in $\mathcal{P}_X$. If it is in Y, it has a nonempty prefix in $\mathcal{P}_Y$.
(c) If $\varepsilon \notin X$, each $x \in XY$ has a nonempty prefix in X This, in turn, has a nonempty prefix in $\mathcal{P}_X$.
(d) If $\varepsilon \in X$, we have $XY = (X - \varepsilon)Y \cup Y$. From (c) we have $\mathcal{P}_X \sqsubseteq (X - \varepsilon)Y$. The stated result follows from (b). □

6 Prefix Disjointness

We say that $X \subseteq \Sigma^*$ and $Y \subseteq \Sigma^*$ are *prefix-disjoint*, and write $X \asymp Y$ to mean that for all nonempty $x \in X$ and $y \in Y$ neither $x \leq y$ nor $y \leq x$.

One can easily see that prefix-disjoint languages are disjoint, but the reverse is in general not true.

Lemma 3. *Let $X \subseteq \Sigma^+$ and $Y \subseteq \Sigma^*$. For any $\mathcal{P}_X \sqsubseteq X$ and $\mathcal{P}_Y \sqsubseteq Y$, $\mathcal{P}_X \asymp \mathcal{P}_Y$ implies $X \cap \mathrm{Pref}(Y) = \varnothing$.*

Proof. Let $X, Y, \mathcal{P}_X, \mathcal{P}_Y$ be as stated. Suppose that $X \cap \mathrm{Pref}(Y) \neq \varnothing$, so there exists x such that $x \in X$, $x \in \mathrm{Pref}(Y)$. Since $x \neq \varepsilon$, we have $x = uv$ for some nonempty $u \in \mathcal{P}_X$ and $v \in \Sigma^*$. As $x \in \mathrm{Pref}(Y)$, we have $xt \in Y$ for some $t \in \Sigma^*$. As $xt \neq \varepsilon$, we have $xt = pq$ for some nonempty $p \in \mathcal{P}_Y, q \in \Sigma^*$. This gives $uvt = pq$; this means either $u \leq p$ or $p \leq u$, which contradicts $\mathcal{P}_X \asymp \mathcal{P}_Y$. □

We can now apply this result to approximate (1).

Proposition 4. *The two interpretations of* $\mathbb{G}$ *are equivalent if for every Choice* $A \mapsto e_1|e_2 \in \mathbb{G}$*:*

- $\varepsilon \notin \mathcal{L}(e_1)$
- *There exist* $\mathcal{P}_1 \sqsubseteq \mathcal{L}(e_1)$ *and* $\mathcal{P}_2 \sqsubseteq \mathcal{L}(e_2)\,\mathrm{Tail}(A)$ *such that* $\mathcal{P}_1 \asymp \mathcal{P}_2$.

Proof. Consider any Choice $A \mapsto e_1|e_2 \in \mathbb{G}$. Assume $\varepsilon \notin \mathcal{L}(e_1)$ and $\mathcal{P}_1$, $\mathcal{P}_2$ as stated. By Lemma 3, we have $\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing$, and the stated result follows from Proposition 3. □

Proposition 5. *The two interpretations of* $\mathbb{G}$ *are equivalent if for every Choice* $A \mapsto e_1|e_2 \in \mathbb{G}$*:*

- $\varepsilon \notin \mathcal{L}(e_1)$.
- *If* $\varepsilon \notin \mathcal{L}(e_2)$*, there exist* $\mathcal{P}_1 \sqsubseteq \mathcal{L}(e_1)$ *and* $\mathcal{P}_2 \sqsubseteq \mathcal{L}(e_2)$ *such that* $\mathcal{P}_1 \asymp \mathcal{P}_2$.
- *If* $\varepsilon \in \mathcal{L}(e_2)$*, there exist* $\mathcal{P}_1 \sqsubseteq \mathcal{L}(e_1)$*,* $\mathcal{P}_2 \sqsubseteq \mathcal{L}(e_2)$*, and* $\mathcal{P}_T \sqsubseteq \mathrm{Tail}(A)$ *such that* $\mathcal{P}_1 \asymp (\mathcal{P}_2 \cup \mathcal{P}_T)$.

Proof. Consider any Choice $A \mapsto e_1 \,|\, e_2 \in \mathbb{G}$. Assume $\varepsilon \notin \mathcal{L}(e_1)$, $\mathcal{P}_1 \sqsubseteq \mathcal{L}(e_1)$ and $\mathcal{P}_2 \sqsubseteq \mathcal{L}(e_2)$. Suppose $\varepsilon \notin \mathcal{L}(e_2)$. By Lemma 2(c), $\mathcal{P}_2 \sqsubseteq \mathcal{L}(e_2)\,\mathrm{Tail}(A)$, so $\mathcal{P}_1 \asymp \mathcal{P}_2$ satisfies Proposition 4. Suppose now that $\varepsilon \in \mathcal{L}(e_2)$ and $\mathcal{P}_T \sqsubseteq \mathrm{Tail}(A)$. By Lemma 2(d), $\mathcal{P}_2 \cup \mathcal{P}_T \sqsubseteq \mathcal{L}(e_2)\,\mathrm{Tail}(A)$, so $\mathcal{P}_1 \asymp \mathcal{P}_2 \cup \mathcal{P}_T$ satisfies Proposition 4. □

7 Extending the Classical FIRST

One can easily see that the classical $\textsc{First}(e)$ and $\textsc{Follow}(A)$ are prefix covers of e and $\mathrm{Tail}(A)$ that consist of one-letter words. For such sets, $X \asymp Y$ is the same as $X \cap Y = \varnothing$, so $\textsc{First}(e_1) \cap \textsc{First}(e_2) = \varnothing$ and $\textsc{First}(e_1) \cap (\textsc{First}(e_2) \cup \textsc{Follow}(A)) = \varnothing$ are special cases of the last two conditions in Proposition 5. They are exactly the LL(1) conditions.

We are now going to look at other sets that may be used instead of $\textsc{First}(e)$ and $\textsc{Follow}(A)$. For $A \in N$ define:

- $\mathtt{First}(A) = \{e_1, e_2\}$ if $e(A) = e_1 \,|\, e_2$,
- $\mathtt{First}(A) = \{e_1\}$ if $e(A) = e_1\, e_2$ and $\varepsilon \notin \mathcal{L}(e_1)$,
- $\mathtt{First}(A) = \{e_1, e_2\}$ if $e(A) = e_1\, e_2$ and $\varepsilon \in \mathcal{L}(e_1)$.

For $E \subseteq \mathbb{E}$, let $\mathcal{FIRST}(E)$ be the family of subsets of $\mathbb{E}$ defined as follows:

- $E \in \mathcal{FIRST}(E)$.
- If F belongs to $\mathcal{FIRST}(E)$, the result of replacing its member $A \in N$ by all elements of $\mathtt{First}(A)$ also belongs to $\mathcal{FIRST}(E)$.
- nothing else belongs to $\mathcal{FIRST}(E)$ unless its being so follows from the above.

These definitions are illustrated in Figure 3. An arrow from e_1 to e_2 means $e_2 \in \mathtt{First}(e_1)$. One can easily see that $\mathcal{FIRST}(E)$ contains the classical $\textsc{First}(E)$.

$A \mapsto C \,|\, D$

$B \mapsto E\, A$

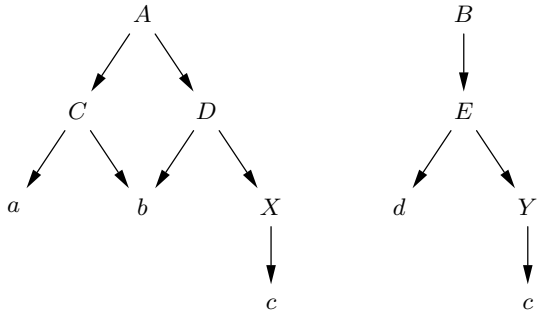
$C \mapsto a \,|\, b$

$D \mapsto b \,|\, X$

$E \mapsto d \,|\, Y$

$X \mapsto c\, D$

$Y \mapsto c\, E$



$\mathcal{FIRST}(A) = \{\{A\}, \{C, D\}, \{a, b, D\}, \{C, b, X\}, \{a, b, X\}, \{a, b, c\}\}$

$\mathcal{FIRST}(B) = \{\{B\}, \{E\}, \{d, Y\}, \{d, c\}\}$

Fig. 3. Relation `First` and family $\mathcal{FIRST}$

For $E \subseteq \mathbb{E}$, let $\mathcal{L}(E)$ denote the union $\bigcup_{e \in E} \mathcal{L}(e)$.

Lemma 4. *For each $A \in N$ holds $\mathcal{L}(\mathtt{First}(A)) \sqsubseteq \mathcal{L}(A)$.*

Proof. In case $e(A) = e_1 | \, e_2$ we have $\mathcal{L}(A) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2) = \mathcal{L}(e_1 | \, e_2)$.
Each language is its own prefix cover, so $\mathcal{L}\{e_1, e_2\} \sqsubseteq \mathcal{L}(e_1 | \, e_2)$.
In case $e(A) = e_1\, e_2$ we have $\mathcal{L}(A) = \mathcal{L}(e_1)\mathcal{L}(e_2)$. Again, $\mathcal{L}(e_1) \sqsubseteq \mathcal{L}(e_1)$ and $\mathcal{L}(e_2) \sqsubseteq \mathcal{L}(e_2)$.
If $\varepsilon \notin e_1$, we have, by Lemma 2(c), $\mathcal{L}\{e_1\} \sqsubseteq \mathcal{L}(e_1)\mathcal{L}(e_2)$.
If $\varepsilon \in e_1$, we have, by Lemma 2(d), $\mathcal{L}\{e_1, e_2\} = \mathcal{L}(e_1) \cup \mathcal{L}(e_2) \sqsubseteq \mathcal{L}(e_1)\mathcal{L}(e_2)$. □

Lemma 5. *For each $F \in \mathcal{FIRST}(E)$ holds $\mathcal{L}(F) \sqsubseteq \mathcal{L}(E)$.*

Proof. We prove the Proposition by induction on the number of replacements.

(Induction base) As each language is its own prefix cover, $\mathcal{L}(E) \sqsubseteq \mathcal{L}(E)$.

(Induction step) Assume $\mathcal{L}\{e_1, e_2, \ldots, e_n\} \sqsubseteq \mathcal{L}(E)$. As sets are not ordered, we can always consider replacing e_1. Let $\mathtt{First}(e_1) = \{f_1, f_2\}$. (The proof will be similar if $\mathtt{First}(e_1)$ has only one element.) According to Lemma 4, $\mathcal{L}\{f_1, f_2\} \sqsubseteq \mathcal{L}(e_1)$. We also have $\mathcal{L}\{e_2, \ldots, e_n\} \sqsubseteq \mathcal{L}\{e_2, \ldots, e_n\}$. Using Lemma 2(b), we obtain:

$$\begin{aligned}\mathcal{L}\{f_1, f_2, e_2, \ldots, e_n\} &= \mathcal{L}\{f_1, f_2\} \cup \mathcal{L}\{e_2, \ldots, e_n\} \\ &\sqsubseteq \mathcal{L}(e_1) \cup \mathcal{L}\{e_2, \ldots, e_n\} = \mathcal{L}\{e_1, e_2, \ldots, e_n\}.\end{aligned}$$

From Lemma 2(a) follows $\mathcal{L}\{f_1, f_2, e_2, \ldots, e_n\} \sqsubseteq \mathcal{L}(E)$. □

We can apply this result directly to ε-free grammars:

Proposition 6. *If none of the rules contains ε, the two interpretations of $\mathbb{G}$ are equivalent if for every Choice $A \mapsto e_1 | e_2 \in \mathbb{G}$ there exist* $\textsc{First}_1 \in \mathcal{FIRST}(e_1)$ *and* $\textsc{First}_2 \in \mathcal{FIRST}(e_2)$ *such that* $\mathcal{L}(\textsc{First}_1) \asymp \mathcal{L}(\textsc{First}_2)$.

Proof. Suppose the required $\textsc{First}_1$ and $\textsc{First}_2$ exist. If $\mathbb{G}$ is ε-free, we have $\varepsilon \notin \mathcal{L}(e_1)$ and $\varepsilon \notin \mathcal{L}(e_2)$. From Lemma 5 we have $\mathcal{L}(\textsc{First}_1) \sqsubseteq \mathcal{L}(e_1)$ and $\mathcal{L}(\textsc{First}_2) \sqsubseteq \mathcal{L}(e_2)$. The stated result follows from Proposition 5. □

8 Extending the Classical FOLLOW

To handle the case when $\mathbb{G}$ is not ε-free, we need to find suitable prefix covers of $\text{Tail}(A)$. For this purpose, we borrow from [7] the definition of $\texttt{Follow} \subseteq \mathbb{E} \times \mathbb{E}$. For $e \in \mathbb{E}$, define $\texttt{Last}(e)$ to be the set of all $A \in N$ such that:

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

For $e \in \mathbb{E}$, define $\texttt{Next}(e) = \{e_1 \in \mathbb{E} : \text{exists } A \mapsto ee_1 \in \mathbb{G}\}$.
Finally, define $\texttt{Follow} = \texttt{Last}^* \times \texttt{Next}$.

Lemma 6. *For each $A \in N$, $\mathcal{L}(\texttt{Follow}(A)) \sqsubseteq \text{Tail}(A)$.*

Proof. Consider some $A \in N$ and $y\$ \in \text{Tail}(A)$. By definition, there is a proof of $[S]\ w\$ \overset{\text{CFG}}{\rightsquigarrow} \$$ that contains $[A]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ 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) $[A_1]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ where $e(A_1) = A \,|\, e$ from $[e]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ using **choice.1c**.
(b) $[A_1]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ where $e(A_1) = e \,|\, A$ from $[e]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ using **choice.2c**.
(c) $[A_1]\ zxy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ where $e(A_1) = e\, A$ from $[e]\ zxy\$ \overset{\text{CFG}}{\rightsquigarrow} xy\$$ using **seq.1c**.
(d) $[A_1]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ where $e(A_1) = A\, e$ from $[e]\ \varepsilon y\$ \overset{\text{CFG}}{\rightsquigarrow} y\$$ using **seq.1c**.
(e) $[A_1]\ xy'y''\$ \overset{\text{CFG}}{\rightsquigarrow} y''\$$ where $e(A_1) = A\,B$, $y'y'' = y$ and $y' \neq \varepsilon$ from $[B]\ y'y''\$ \overset{\text{CFG}}{\rightsquigarrow} y''\$$ using **seq.1c**.

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:

$$\frac{[A_0]\ xy\$ \overset{\text{CFG}}{\rightsquigarrow} y\$ \quad \ldots \quad \text{as in (a)-(d)}}{\dfrac{[A_1]\ x_1y\$ \overset{\text{CFG}}{\rightsquigarrow} y\$ \quad \ldots \quad \text{as in (a)-(d)}}{\dfrac{[A_2]\ x_2y\$ \overset{\text{CFG}}{\rightsquigarrow} y\$ \quad \ldots \quad \text{as in (a)-(d)}}{\ldots \qquad \dfrac{}{\dfrac{[A_{n-1}]\ x_{n-1}y\$ \overset{\text{CFG}}{\rightsquigarrow} y\$ \quad \ldots \quad \text{as in (a)-(d)}}{\dfrac{[A_n]\ x_ny\$ \overset{\text{CFG}}{\rightsquigarrow} y\$ \quad [B]\ y'y''\$ \overset{\text{CFG}}{\rightsquigarrow} y''\$ \quad e(A_{n+1}) = A_nB}{[A_{n+1}]\ x_ny'y''\$ \overset{\text{CFG}}{\rightsquigarrow} y''\$}}}}}$$

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 $[B]\ y'y''\$ \overset{\text{CFG}}{\rightsquigarrow} y''\$$ we have $y' \in \mathcal{L}(B)$; since $y = y'y''$ and $y' \neq \varepsilon$, $y\$$ has a nonempty prefix in $\mathcal{L}(B) \subseteq \mathcal{L}(\texttt{Follow}(A))$. □

Proposition 7. *The two interpretations of* $\mathbb{G}$ *are equivalent if for every Choice* $A \mapsto e_1 | e_2 \in \mathbb{G}$*:*

- $\varepsilon \notin \mathcal{L}(e_1)$.
- *If* $\varepsilon \notin \mathcal{L}(e_2)$*, there exist* $\textsc{First}_1 \in \mathcal{FIRST}(e_1)$ *and* $\textsc{First}_2 \in \mathcal{FIRST}(e_2)$ *such that* $\mathcal{L}(\textsc{First}_1) \asymp \mathcal{L}(\textsc{First}_2)$.
- *If* $\varepsilon \in \mathcal{L}(e_2)$*, there exist* $\textsc{First}_1 \in \mathcal{FIRST}(e_1)$, $\textsc{First}_2 \in \mathcal{FIRST}(e_2)$, *and* $\textsc{Follow} \in \mathcal{FIRST}(\texttt{Follow}(A))$ *such that* $\mathcal{L}(\textsc{First}_1) \asymp \mathcal{L}(\textsc{First}_2 \cup \textsc{Follow})$.

Proof. Suppose the required $\textsc{First}_1$, $\textsc{First}_2$, and $\textsc{Follow}$ exist. From Lemma 5 we have $\mathcal{L}(\textsc{First}_1) \sqsubseteq \mathcal{L}(e_1)$ and $\mathcal{L}(\textsc{First}_2) \sqsubseteq \mathcal{L}(e_2)$. From Lemmas 5 and 6 we have $\mathcal{L}(\textsc{Follow}) \sqsubseteq \text{Tail}(A)$. The stated result follows from Proposition 5. □

9 Final Remarks

We have shown that a direct transcription of an EBNF grammar to equivalent PEG is possible for grammars outside the LL(1) class. These are the grammars where the choice of the way to proceed is made by examining the input within the reach of one parsing procedure instead of examining a single letter. One can call them "LL(1P)" grammars, the "1P" meaning "one procedure". However, checking the conditions stated by Proposition 7 is not as simple as verifying the LL(1) property. The families $\mathcal{FIRST}$ can be constructed in a mechanical way, and are presumably not very large. But, checking the disjointness of their members may be difficult. It becomes more difficult as one moves up the graph in Figure 3; at the top of that graph we come close to the condition (1).

If the sets $\textsc{First}_1$, $\textsc{First}_2$, and $\textsc{Follow}$ satisfying Proposition 7 exist, the information can be used for the improvement suggested by Mizushima et al. in [5]. The set $\textsc{First}_1$ lists the alternative parsing procedures that will be eventually invoked to start the processing of e_1. If a procedure $P \in \textsc{First}_1$ succeeds, but the further processing of e_1 fails, the conditions of Proposition 7 mean that e_2 is doomed to fail, unless it succeeds on empty string. But in this case, the subsequent processing of Tail(A) will fail. One can thus insert a "cut" operator after P to save memoization and an unnecessary attempt at e_2.

The class of EBNF grammars that can be directly transcribed to PEG includes cases where the choice of the way to proceed is made by looking at the input within the reach of more than one parsing procedure. The following is an example of such grammar:

InputLine → *Assignment* `|` *Expression*
Assignment → *Name* `"="` *Expression*
Expression → *Primary* `([+-]` *Primary*`)`*
Primary → *Name* `|` *Number*
Name → `[a-z]`$^+$
Number → `[0-9]`$^+$

Here both alternatives of *InputLine* may begin with *Name*. It is the next procedure after *Name*, namely one for `"="`, that decides whether to proceed or backtrack. This class of grammars, which may be called "LL(2P)", requires further analysis. Propositions 3, 4, and 5 still apply there.

Last but not least: it appears that natural semantics, as used by Medeiros to define the meaning of a grammar, is a very convenient tool to investigate the grammar's properties.

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. Redziejowski, R.R.: Some aspects of Parsing Expression Grammar. Fundamenta Informaticae **85**(1–4) (2008) 441–454
4. Medeiros, S.: Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro (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`.
5. Mizushima, K., Maeda, A., Yamaguchi, Y.: Packrat parsers can handle practical grammars in mostly constant space. In Lerner, S., Rountev, A., eds.: Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE'10, Toronto, Ontario, Canada, June 5-6, 2010, ACM (2010) 29–36
6. Schmitz, S.: Modular syntax demands verification. Technical Report ISRN I3S/RR 2006-32-FR, Laboratoire I3S, Sophia Antipolis (2006)
7. Tremblay, J.P., Sorenson, P.G.: The Theory and Practice of Compiler Writing. McGraw-Hill (1985)