Grammars for Indentation-Sensitive Parsing

Härmel Nestra

Institute of Computer Science, University of Tartu, Tartu, Estonia
harmel.nestra@ut.ee

Abstract

Adams' extension of parsing expression grammars enables specifying indentation sensitivity using two non-standard grammar constructs – indentation by a binary relation and alignment. This paper is a theoretical study of Adams' grammars. It proposes a step-by-step transformation of well-formed Adams' grammars for elimination of the alignment construct from the grammar. The idea that alignment could be avoided was suggested by Adams but no process for achieving this aim has been described before. This paper also establishes general conditions that binary relations used in indentation constructs must satisfy in order to enable efficient parsing.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, D.3.4 Processors, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Parsing expression grammars, indentation, grammar transformation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.45

1 Introduction

Parsing expression grammars (PEG) introduced by Ford [6] serve as a modern framework for specifying the syntax of programming languages and are an alternative to the classic context-free grammars (CFG). The core difference between CFG and PEG is that descriptions in CFG can be ambiguous while PEGs are inherently deterministic. A syntax specification written in PEG can in principle be interpreted as a top-down parser for that syntax; in the case of left recursion, this treatment is not straightforward but doable (see, e.g., [8]).

Formally, a PEG is a quadruple $G = (N, T, \delta, \mathsf{s})$ where:

- N is a finite set of *non-terminals*;
- T is a finite set of *terminals*;
- δ is a function mapping each non-terminal to its replacement (corresponding to the set of productions of CFG);
- s is the *start expression* (corresponding to the start symbol of CFG).

So $\delta : N \to \mathcal{E}_G$ and $\mathsf{s} \in \mathcal{E}_G$, where the set $\mathcal{E}_G$ of all *parsing expressions* writable in G is defined inductively as follows:

1. $\varepsilon \in \mathcal{E}_G$ (the empty string);
2. $a \in \mathcal{E}_G$ for every $a \in T$ (the terminals);
3. $X \in \mathcal{E}_G$ for every $X \in N$ (the non-terminals);
4. $\mathsf{pq} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{q} \in \mathcal{E}_G$ (*concatenation*)
5. $\mathsf{p}/\mathsf{q} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{q} \in \mathcal{E}_G$ (*choice*);
6. $!\mathsf{p} \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$ (*negation*, or *lookahead*);
7. $\mathsf{p}^* \in \mathcal{E}_G$ whenever $\mathsf{p} \in \mathcal{E}_G$ (*repetition*).

All constructs of PEG except for negation are direct analogues of constructs of the EBNF form of CFG, but their semantics is always deterministic. So p^* repeats parsing of p until failure, and p/q always tries to parse p first, q is parsed only if p fails. For example, the expression ab/a consumes the input string `ab` entirely while a/ab only consumes its first character. The corresponding EBNF expressions $ab \mid a$ and $a \mid ab$ are equivalent, both can match either a or ab from the input string. Negation $!p$ tries to parse p and fails if p succeeds; if p fails then $!p$ succeeds with consuming no input. Other constructs of EBNF like non-null repetition p^{+} and optional occurrence $[p]$ can be introduced to PEG as syntactic sugar.

Languages like Python and Haskell allow the syntactic structure of programs to be shown by indentation and alignment, instead of the more conventional braces and semicolons. Handling indentation and alignment in Python has been specified in terms of extra tokens INDENT and DEDENT that mark increasing and decreasing of indentation and must be generated by the lexer. In Haskell, rules for handling indentation and alignment are more sophisticated. Both these languages enable to locally use a different layout mode where indentation does not matter, which additionally complicates the task of formal syntax specification. Adams and Ağacan [3] proposed an extension of PEG notation for specifying indentation sensitivity and argued that it considerably simplifies this task for Python, Haskell and many other indentation-sensitive languages.

In this extension, expression $p^{>}$, for example, denotes parsing of p while assuming a greater indentation than that of the surrounding block. In general, parsing expressions may be equipped with binary relations (as was $>$ in the example) that must hold between the baselines of the local and the current indentation block. In addition, $¦p¦$ denotes parsing of p while assuming the first token of the input being aligned, i.e., positioned on the current indentation baseline. For example, the do expressions in Haskell can be specified by

$$\begin{array}{lll}
\langle doexp \rangle & ::= & \texttt{do}^{>}\ (\langle istmts \rangle / \langle stmts \rangle) \\
\langle istmts \rangle & ::= & (¦\langle stmt \rangle¦^{+})^{>} \\
\langle stmts \rangle & ::= & \texttt{\{}^{>}(\langle stmt \rangle(\texttt{;}\langle stmt \rangle)^{*}[\texttt{;}]\texttt{\}})^{\circledast}
\end{array}$$

Here, $\langle istmts \rangle$ and $\langle stmts \rangle$ stand for statement lists in the indentation and relaxed mode, respectively. In the indentation mode, a statement list is indented (marked by $>$ in the second production) and all statements in it are aligned (marked by $¦\cdot¦$). In the relaxed mode, however, relation $\circledast$ is used to indicate that the indentation baseline of the contents can be anything. (Technically, $\circledast$ is the binary relation containing all pairs of natural numbers.) Terminals `do` and `{` are also equipped with $>$ to meet the Haskell requirement that subsequent tokens of aligned blocks must be indented more than the first token.

Alignment construct provides fulcra for disambiguating the often large variety of indentation baseline candidates. Besides simplicity of this grammar extension and its use, a strength of it lies in the fact that grammars can still serve as parsers.

The rest of the paper is organized as follows. Section 2 formally introduces additional constructs of PEG for specifying code layout, defines their semantics and studies their semantic properties. In Sect. 3, a semantics-preserving process of eliminating the alignment construct from grammars is described. General criteria for deciding if parsing can handle a relation efficiently are found in Sect. 4. Section 5 refers to related work and Sect. 6 concludes.

2 Indentation extension of PEG

Adams and Ağacan [3] extend PEGs with the indentation and alignment constructs. We propose a slightly different extension with three rather than two extra constructs. Our approach agrees with that implemented by Adams in his `indentation` package for Haskell [1], whence calling the grammars in our approach *Adams' grammars* is justified. All differences between the definitions in this paper and in [3] are listed and discussed in Subsect. 2.4.

Let $\mathbb{N}$ denote the set of all natural numbers, and let $\mathbb{B} = \{tt, ff\}$ (the Boolean domain). Denote by $\wp(X)$ the set of all subsets of set X, and let $\Re(X)$ denote the set of all binary relations on set X, i.e., $\Re(X) = \wp(X \times X)$. Standard examples are $> \in \Re(\mathbb{N})$ (consisting of all pairs (n, m) of natural numbers such that $n > m$) and $\triangle \in \Re(\mathbb{N})$ (the identity relation consisting of all pairs of equal natural numbers); the indentation extension also makes use of $\circledast \in \Re(\mathbb{N})$ (the relation containing all pairs of natural numbers). Whenever $\rho \in \Re(X)$ and $Y \subseteq X$, denote $\rho(Y) = \{x \in X : \exists y \in Y.(y, x) \in \rho\}$ (the image of Y under relation ρ). The inverse relation of ρ is defined by $\rho^{-1} = \{(x, y) : (y, x) \in \rho\}$, and the composition of relations σ and ρ by $\sigma \circ \rho = \{(x, z) : \exists y.(x, y) \in \sigma \wedge (y, z) \in \rho\}$. Finally, denote $\Re^{+}(X) = \{\rho \in \Re(X) : \forall x \in X.\rho^{-1}(\{x\}) \neq \varnothing\} = \{\rho \in \Re(X) : \rho(X) = X\}$.

2.1 Adams' grammars

Extend the definition of $\mathcal{E}_G$ given in Sect. 1 with the following three additional clauses:

8. $p^{\rho} \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ and $\rho \in \Re(\mathbb{N})$ (*indentation*);
9. $p_{\sigma} \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ and $\sigma \in \Re(\mathbb{N})$ (*token position*);
10. $|p| \in \mathcal{E}_G$ for every $p \in \mathcal{E}_G$ (*alignment*).

Parsing of an expression p^{ρ} means parsing of p while assuming that the part of the input string corresponding to p forms a new indentation block whose baseline is in relation ρ to the baseline of the surrounding block. (Baselines are identified with column numbers.) The position construct p_{σ}, missing in [3], determines how tokens of the input can be situated w.r.t. the current indentation baseline. Finally, parsing an expression $|p|$ means parsing of p while assuming the first token of the input being positioned on the current indentation baseline (unlike the position operator, this construct does not affect processing the subsequent tokens).

Inspired by the `indentation` package [1], we call the relations that determine token positioning w.r.t. the indentation baseline *token modes*. In the token mode $>$ for example, tokens may appear only to the right of the indentation baseline. Applying the position operator with relation $>$ to parts of Haskell grammar to be parsed in the indentation mode avoids indenting every single terminal in the example in Sect. 1. Also, indenting terminals with $>$ is inadequate for do expressions occurring inside a block of relaxed mode but the position construct can be easily used to change the token mode for such blocks (e.g., to $\geq$).

We call a PEG extended with these three constructs a PEG$^{>}$. Recall from Sect. 1 that N and T denote the set of non-terminal and terminal symbols of the grammar, respectively, and $\delta : N \to \mathcal{E}_G$ is the production function. Concerning the semantics of PEG$^{>}$, each expression parses an input string of terminals ($w \in T^{*}$) in the context of a current set of indentation baseline candidates ($I \in \wp(\mathbb{N})$) and a current alignment flag indicating whether the next terminal should be aligned or not ($b \in \mathbb{B}$), assuming a certain token mode ($\tau \in \Re(\mathbb{N})$). Parsing may succeed, fail, or diverge. If parsing succeeds, it returns as a result a new triple containing the rest of the input w', a new set I' of baseline candidates updated according to the information gathered during parsing, and a new alignment flag b'. This result is denoted by $\top(w', I', b')$. If parsing fails, there is no result in a triple form; failure is denoted by $\bot$.

Triples of the form $(w, I, b) \in T^{*} \times \wp(\mathbb{N}) \times \mathbb{B}$ are behaving as *operation states* of parsing, as each parsing step may use these data and update them. We will write $\mathit{State} = T^{*} \times \wp(\mathbb{N}) \times \mathbb{B}$ (as we never deal with different terminal sets, dependence on T is not explicitly marked), and denote by $\mathit{State} + 1$ the set of possible results of parsing, i.e., $\{\top(s) : s \in \mathit{State}\} \cup \{\bot\}$.

The assertion that parsing expression e in grammar G with input string w in the context of I and b assuming token mode τ results in $o \in \mathit{State} + 1$ is denoted by $e, \tau \vdash_G (w, I, b) \to o$. The formal definition below must be interpreted inductively, i.e., an assertion of the form $G, \tau \vdash_e s \to o$ is valid iff it has a finite derivation by the following ten rules:

1. $\varepsilon, \tau \vdash_G s \to \top(s)$.
2. For every $a \in T$, $a, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
   - If $o = \top(w', I', \mathit{ff})$ for w', I', i such that $w = a^i w'$ (a^i denotes a occurring at column i) and either $b = \mathit{ff}$ and $i \in \tau^{-1}(I)$, $I' = I \cap \tau(\{i\})$, or $b = \mathit{tt}$ and $i \in I$, $I' = \{i\}$;
   - If $o = \bot$, and there are no w' and i such that $w = a^i w'$ with either $b = \mathit{ff}$ and $i \in \tau^{-1}(I)$ or $b = \mathit{tt}$ and $i \in I$.
3. For every $X \in N$, $X, \tau \vdash_G s \to o$ holds if $\delta(X), \tau \vdash_G s \to o$ holds.
4. For every $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$, $\mathsf{pq}, \tau \vdash_G s \to o$ holds in two cases:
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $\mathsf{q}, \tau \vdash_G s' \to o$;
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \bot$.
5. For every $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$, $\mathsf{p}/\mathsf{q}, \tau \vdash_G s \to o$ holds in two cases:
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $o = \top(s')$;
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $\mathsf{q}, \tau \vdash_G s \to o$.
6. For every $\mathsf{p} \in \mathcal{E}_G$, $!\mathsf{p}, \tau \vdash_G s \to o$ holds in two cases:
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \top(s)$;
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $o = \bot$.
7. For every $\mathsf{p} \in \mathcal{E}_G$, $\mathsf{p}^*, \tau \vdash_G s \to o$ holds in two cases:
   - If $\mathsf{p}, \tau \vdash_G s \to \bot$ and $o = \top(s)$;
   - If there exists a triple s' such that $\mathsf{p}, \tau \vdash_G s \to \top(s')$ and $\mathsf{p}^*, \tau \vdash_G s' \to o$.
8. For every $\mathsf{p} \in \mathcal{E}_G$ and $\rho \in \Re(\mathbb{N})$, $\mathsf{p}^{\rho}, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
   - If there exists a triple (w', I', b') such that $\mathsf{p}, \tau \vdash_G (w, \rho^{-1}(I), b) \to \top(w', I', b')$ and $o = \top(w', I \cap \rho(I'), b')$;
   - If $\mathsf{p}, \tau \vdash_G (w, \rho^{-1}(I), b) \to \bot$ and $o = \bot$.
9. For every $\mathsf{p} \in \mathcal{E}_G$ and $\sigma \in \Re(\mathbb{N})$, $\mathsf{p}_{\sigma}, \tau \vdash_G s \to o$ holds if $\mathsf{p}, \sigma \vdash_G s \to o$ holds.
10. For every $\mathsf{p} \in \mathcal{E}_G$, $¦\mathsf{p}¦, \tau \vdash_G (w, I, b) \to o$ holds in two cases:
    - If there exists a triple (w', I', b') such that $\mathsf{p}, \tau \vdash_G (w, I, \mathit{tt}) \to \top(w', I', b')$ and $o = \top(w', I', b \wedge b')$;
    - If $\mathsf{p}, \tau \vdash_G (w, I, \mathit{tt}) \to \bot$ and $o = \bot$.

The idea behind the conditions $i \in \tau^{-1}(I)$ and $i \in I$ occurring in clause 2 is that any column i where a token may appear is in relation τ with the current indentation baseline (known to be in I) if the alignment flag is false, and coincides with the indentation baseline otherwise. For the same reason, consuming a token in column i restricts the set of allowed indentations to $\tau(\{i\})$ or $\{i\}$ depending on the alignment flag. In both cases, the alignment flag is set to ff. Similar principles lie behind the changes of the operation states in clauses 8 and 10.

For a toy example, consider parsing of $¦ab¦^{>}$ with the operation state $(a^2b^3, \mathbb{N}, \mathit{ff})$ assuming the token mode $\geq$. For that, we must parse $¦ab¦$ with $(a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{ff})$ by clause 8 since $>^{-1}(\mathbb{N}) = \mathbb{N} \setminus \{0\}$. For that in turn, we must parse ab with $(a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt})$ by clause 10. By clause 2, we have $a, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt}) \to \top(b^3, \{2\}, \mathit{ff})$ (as $2 \in \mathbb{N} \setminus \{0\}$) and $b, \geq \vdash_G (b^3, \{2\}, \mathit{ff}) \to \top(\varepsilon, \{2\}, \mathit{ff})$ (as $(2, 3) \in \geq^{-1}$). Therefore, by clause 4, $ab, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{tt}) \to \top(\varepsilon, \{2\}, \mathit{ff})$. Finally, $¦ab¦, \geq \vdash_G (a^2b^3, \mathbb{N} \setminus \{0\}, \mathit{ff}) \to \top(\varepsilon, \{2\}, \mathit{ff})$ and $¦ab¦^{>}, \geq \vdash_G (a^2b^3, \mathbb{N}, \mathit{ff}) \to \top(\varepsilon, \{0, 1\}, \mathit{ff})$ by clauses 10 and 8. The set $\{0, 1\}$ in the final state shows that only 0 and 1 are still candidates for the indentation baseline outside the parsed part of the input (before parsing, the candidate set was the whole $\mathbb{N}$).

Note that this definition involves circular dependencies. For instance, if $\delta(X) = X$ for some $X \in N$ then $X, \tau \vdash_G s \to o$ if $X, \tau \vdash_G s \to o$ by clause 3. There is no result of parsing in such cases (not even $\bot$). We call this behaviour *divergence*.

2.2 Properties of the semantics

Ford [6] proves that parsing in PEG is unambiguous, whereby the consumed part of an input string always is its prefix. Theorem 2.1 below is an analogous result for PEG$^{>}$. Besides the uniqueness of the result of parsing, it states that if we only consider relations in $\Re^{+}(\mathbb{N})$ then the whole operation state in our setting is in a certain sense decreasing during parsing.

Denote by $\geq$ the *suffix order* of strings (i.e., $w \geq w'$ iff $w = uw'$ for some $u \in T^{*}$) and by $\sqsupseteq$ the *implication order* of truth values (i.e., $tt \sqsupset f\!f$). Denote by $\geqslant$ the pointwise order on operation states, i.e., $(w, I, b) \geqslant (w', I', b')$ iff $w \geq w'$, $I \supseteq I'$ and $b \sqsupseteq b'$.

▶ **Theorem 2.1.** *Let $G = (N, T, \delta, s)$ be a PEG$^{>}$, $\mathsf{e} \in \mathcal{E}_G$, $\tau \in \Re^{+}(\mathbb{N})$ and $s \in State$. Then $\mathsf{e}, \tau \vdash_G s \to o$ for at most one o, whereby $o = \top(s')$ implies $s \geqslant s'$. Also if $s = (w, I, b)$ and $s' = (w', I', b')$ then $s \neq s'$ implies both $w > w'$ and $b' = f\!f$, and $I \neq \varnothing$ implies $I' \neq \varnothing$.*

Proof. By induction on the shape of the derivation tree of the assertion $\mathsf{e}, \tau \vdash_G s \to o$. ◀

Theorem 2.1 enables to observe a common pattern in the semantics of indentation and alignment. Denoting by $\kappa(\mathsf{p})$ either $\mathsf{p}^{\wp}$ or $\lvert\mathsf{p}\rvert$, both clauses 8 and 10 have the following form, parametrized on two mappings $\alpha, \gamma : State \to State$:

For $\mathsf{p} \in \mathcal{E}_G$, $\kappa(\mathsf{p}), \tau \vdash_G s \to o$ holds in two cases:

- If there exists a state s' such that $\mathsf{p}, \tau \vdash_G \alpha(s) \to \top(s')$ and $o = \top(s \wedge \gamma(s'))$;
- If $\mathsf{p}, \tau \vdash_G \alpha(s) \to \bot$ and $o = \bot$.

The meanings of indentation and alignment constructs are distinguished solely by α and γ. For many properties, proofs that rely on this abstract common definition can be carried out, assuming that γ is monotone, preserves the largest element and follows together with α the axiom $x \wedge \gamma(y) \leq \gamma(\alpha(x) \wedge y)$. The class of all meet semilattices L with top element, equipped with mappings α, γ satisfying these three conditions, contains identities (i.e., semilattices L with $\alpha = \gamma = \mathrm{id}_L$) and is closed under compositions (of different α, γ defined on the same semilattice L) and under direct products. If $\rho \in \Re^{+}(\mathbb{N})$ then the conditions hold for $\alpha_1, \gamma_1 : \wp(\mathbb{N}) \to \wp(\mathbb{N})$ with $\alpha_1(I) = \rho^{-1}(I)$, $\gamma_1(I) = \rho(I)$, similarly in the case if $\alpha_2, \gamma_2 : \mathbb{B} \to \mathbb{B}$ with $\alpha_2(b) = tt$, $\gamma_2(b) = b$. Now the direct product of the identities of T^{*} and $\mathbb{B}$ with (α_1, γ_1) on $\wp(\mathbb{N})$ gives the indentation case, and the direct product of the identities of T^{*} and $\wp(\mathbb{N})$ and the Boolean lattice $\mathbb{B}$ with (α_2, γ_2) gives the alignment case.

If α, γ satisfy the conditions then $\gamma(\alpha(x)) \geq x$ since $x = x \wedge \top = x \wedge \gamma(\top) \leq \gamma(\alpha(x) \wedge \top) = \gamma(\alpha(x))$. Adding dual conditions (α monotone, $\alpha(\bot) = \bot$ and $\alpha(x) \vee y \geq \alpha(x \vee \gamma(y))$) would make (α, γ) a Galois' connection. In our cases, the dual axioms do not hold.

2.3 Semantic equivalence

▶ **Definition 2.2.** Let $G = (N, T, \delta, s)$ be a PEG$^{>}$ and $\mathsf{p}, \mathsf{q} \in \mathcal{E}_G$. We say that p and q are *semantically equivalent* in G and denote $\mathsf{p} \sim_G \mathsf{q}$ iff $\mathsf{p}, \tau \vdash_G s \to o \iff \mathsf{q}, \tau \vdash_G s \to o$ for every $\tau \in \Re^{+}(\mathbb{N})$, $s \in State$ and $o \in State + 1$.

For example, one can easily prove that $\mathsf{p}\varepsilon \sim_G \mathsf{p} \sim_G \varepsilon\mathsf{p}$, $\mathsf{p}(\mathsf{qr}) \sim_G (\mathsf{pq})\mathsf{r}$, $\mathsf{p}/(\mathsf{q}/\mathsf{r}) \sim_G (\mathsf{p}/\mathsf{q})/\mathsf{r}$, $\mathsf{p}(\mathsf{q}/\mathsf{r}) \sim_G \mathsf{pq}/\mathsf{pr}$, $\mathsf{p}/\mathsf{q} \sim_G \mathsf{p}/!\mathsf{pq}$ for all $\mathsf{p}, \mathsf{q}, \mathsf{r} \in \mathcal{E}_G$ [6]. We are particularly interested in equivalences involving the additional operators of PEG$^{>}$. In Sect. 3, they will be useful in eliminating alignment and position operators. The following Theorem 2.3 states distributivity laws of the three new operators of PEG$^{>}$ w.r.t. other constructs:

▶ **Theorem 2.3.** *Let $G = (N, T, \delta, s)$ be a $PEG^{>}$. Then:*
1. $\varepsilon_\sigma \sim_G \varepsilon$, $(pq)_\sigma \sim_G p_\sigma q_\sigma$, $(p/q)_\sigma \sim_G p_\sigma/q_\sigma$, $(!p)_\sigma \sim_G !p_\sigma$, $(p^*)_\sigma \sim_G (p_\sigma)^*$, $(p^\rho)_\sigma \sim_G (p_\sigma)^\rho$, $¦p¦_\sigma \sim_G ¦p_\sigma¦$ *for all* $\sigma \in \Re^+(\mathbb{N})$;
2. $\varepsilon^\rho \sim_G \varepsilon$, $(p/q)^\rho \sim_G p^\rho/q^\rho$, $(!p)^\rho \sim_G !p^\rho$, $(p_\sigma)^\rho \sim_G (p^\rho)_\sigma$ *for all* $\rho \in \Re^+(\mathbb{N})$;
3. $¦\varepsilon¦ \sim_G \varepsilon$, $¦p/q¦ \sim_G ¦p¦/¦q¦$, $¦!p¦ \sim_G !¦p¦$, $¦p_\sigma¦ \sim_G ¦p¦_\sigma$.

Proof. The equivalences in claim 1 hold as the token mode steadily distributes to each case of the semantics definition. Those in claims 2 and 3 have straightforward proofs using the joint form of the semantics of indentation and alignment and the axioms of α, γ. ◀

Note that indentation does not distribute with concatenation, i.e., $(pq)^\rho \not\sim_G p^\rho q^\rho$. This is because $(pq)^\rho$ assumes one indentation block with a baseline common to p and q while $p^\rho q^\rho$ tolerates different baselines for p and q. For example, take $p = a \in T$, $q = b \in T$, let the token mode be $\triangle$ and the input state be $(a^1b^2, \mathbb{N}, \mathit{ff})$ (recall that a^i means terminal a occurring in column i). We have $a, \triangle \vdash_G (a^1b^2, \mathbb{N} \setminus \{0\}, \mathit{ff}) \to \top(b^2, \{1\}, \mathit{ff})$ and $b, \triangle \vdash_G (b^2, \{1\}, \mathit{ff}) \to \bot$ (since $(2, 1) \notin \triangle$), therefore $ab, \triangle \vdash_G (a^1b^2, \mathbb{N} \setminus \{0\}, \mathit{ff}) \to \bot$ and $(ab)^>, \triangle \vdash_G (a^1b^2, \mathbb{N}, \mathit{ff}) \to \bot$. On the other hand, $a, \triangle \vdash_G (a^1b^2, \mathbb{N} \setminus \{0\}, \mathit{ff}) \to \top(b^2, \{1\}, \mathit{ff})$ implies $a^>, \triangle \vdash_G (a^1b^2, \mathbb{N}, \mathit{ff}) \to \top(b^2, \{0\}, \mathit{ff})$ (since $\mathbb{N} \cap (> (\{1\})) = \{0\}$) and, analogously, $b^>, \triangle \vdash_G (b^2, \{0\}, \mathit{ff}) \to \top(\varepsilon, \{0\}, \mathit{ff})$ (since $>^{-1}(\{0\}) = \mathbb{N} \setminus \{0\} \ni 2$ and $\{0\} \cap (> (\{2\})) = \{0\}$). Consequently, $a^>b^>, \triangle \vdash_G (a^1b^2, \mathbb{N}, \mathit{ff}) \to \top(\varepsilon, \{0\}, \mathit{ff})$.

We can however prove the following facts:

▶ **Theorem 2.4.** *Let $G = (N, T, \delta, s)$ be a $PEG^{>}$.*
1. *Identity indentation law: For all $p \in \mathcal{E}_G$, $p^\triangle \sim_G p$.*
2. *Composition law of indentations: For all $p \in \mathcal{E}_G$ and $\rho, \sigma \in \Re^+(\mathbb{N})$, $(p^\rho)^\sigma \sim_G p^{\sigma \circ \rho}$.*
3. *Distributivity of indentation and alignment: For all $p \in \mathcal{E}_G$ and $\rho \in \Re^+(\mathbb{N})$, $¦p¦^\rho \sim_G ¦p^\rho¦$.*
4. *Idempotence of alignment: For all $p \in \mathcal{E}_G$, $¦¦p¦¦ \sim_G ¦p¦$.*
5. *Cancellation of outer token modes: For all $p \in \mathcal{E}_G$ and $\sigma, \tau \in \Re(\mathbb{N})$, $(p_\sigma)_\tau \sim_G p_\sigma$.*
6. *Terminal alignment property: For all $a \in T$, $¦a¦ \sim_G a_\triangle$.*

Proof. Claim 1 follows easily from the semantics of indentation. By the conditions imposed on α and γ, it follows that the composition of the effects of indentations or alignments with respective mapping pairs (α_1, γ_1) and (α_2, γ_2) coincides with the effect of a prospective construct of similar kind with mapping pair $(\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2)$. Claims 2–4 follow directly from this observation, as the composition of structures (α, γ) used for indentation and alignment is commutative and the structure (α, γ) used for alignment is idempotent. Claim 5 is trivial. Claim 6 follows from a straightforward case study. ◀

Theorems 2.3 and 2.4 enact bringing alignments through all syntactic constructs except concatenation. Alignment does not distribute with concatenation, because in parsing of an expression of the form $¦pq¦$, the terminal to be aligned can be in the part of the input consumed by p or (if parsing of p succeeds with consuming no input) by q. Alignment can nevertheless be moved through concatenation if any successful parsing of the first expression in the concatenation either never consumes any input or always consumes some input:

▶ **Theorem 2.5.** *Let $G = (N, T, \delta, s)$ be a $PEG^{>}$ and $p, q \in \mathcal{E}_G$.*
1. *If $p, \tau \vdash_G s \to \top(s')$ implies $s' = s$ for all $\tau \in \Re^+(\mathbb{N})$, $s, s' \in State$, then $¦pq¦ \sim_G ¦p¦¦q¦$.*
2. *If $p, \tau \vdash_G s \to \top(s')$ implies $s' \neq s$ for all $\tau \in \Re^+(\mathbb{N})$, $s, s' \in State$, then $¦pq¦ \sim_G ¦p¦q$.*

Proof. Straightforward case study. ◀

Theorem 2.5 (1) holds also for indentation (instead of alignment), the same proof in terms of α, γ is valid. Finally, the following theorem states that position and indentation of terminals are equivalent if the alignment flag is false and the token mode is the identity:

▶ **Theorem 2.6.** *Let $G = (N, T, \delta, s)$ be a PEG$^{>}$. Let $a \in T$, $\sigma \in \Re^{+}(\mathbb{N})$ and $w \in T^{*}$, $I \in \wp(\mathbb{N})$, $o \in State + 1$. Then $a_{\sigma}, \triangle \vdash_{G} (w, I, ff) \to o \iff a^{\sigma}, \triangle \vdash_{G} (w, I, ff) \to o$.*

Proof. Straightforward case study. ◀

2.4 Differences of our approach from previous work

Our specification of PEG$^{>}$differs from the definition used by Adams and Ağacan [3] by three essential aspects listed below. The last two discrepancies can be understood as bugs in the original description that have been corrected in the Haskell `indentation` package by Adams [1]. This package also provides means for locally changing the token mode. All in all, our modifications fully agree with the `indentation` package.

1. The position operator p_{σ} is missing in [3]. The treatment there assumes just one default token mode applying to the whole grammar, whence token positions deviating from the default must be specified using the indentation operator. The benefits of the position operator were shortly discussed in Subsect. 2.1.
2. According to the grammar semantics provided in [3], the alignment flag is never changed at the end of parsing of an expression of the form $\lvert p \rvert$. This is not appropriate if p succeeds without consuming any token, as the alignment flag would unexpectedly remain true during parsing of the next token that is out of scope of the alignment operator. The value the alignment flag had before starting parsing $\lvert p \rvert$ should be restored in this case. This is the purpose of conjunction in the alignment semantics described in this paper.
3. In [3], an alignment is interpreted w.r.t. the indentation baseline of the block that corresponds to the parsing expression to which the alignment operator is applied. Indentation operators occurring inside this expression and processed while the alignment flag is true are neglected. In the semantics described in our paper, raising the alignment flag does not suppress new indentations. Alignments are interpreted w.r.t. the indentation baseline in force at the aligned token site. This seems more appropriate than the former approach where the indentations cancelled because of an alignment do not apply even to the subsequent non-aligned tokens. Distributivity of indentation and alignment fails in the semantics of [3]. Note that alignment of a block nevertheless suppresses the influence of position operators whose scope extend over the first token of the block.

Our grammar semantics has also two purely formal deviations from the semantics used by Adams and Ağacan [3] and Ford [6].

1. We keep track of the rest of the input in the operation state while both [3, 6] expose the consumed part of the input instead. This difference was introduced for simplicity and to achieve uniform decreasing of operation states in Theorem 2.1.
2. We do not have explicit step counts. They were used in [6] to compose proofs by induction. We provide analogous proofs by induction on the shape of derivation trees.

3 Elimination of alignment and position operators

Adams [2] describes alignment elimination in the context of CFGs. In [3], Adams and Ağacan claim that alignment elimination process for PEGs is more difficult due to the lookahead construct. To our knowledge, no concrete process of semantics-preserving alignment elimination is described for PEGs before. We provide one below for well-formed grammars. We rely on the existence of position operators in the grammar; this is not an issue since we also show that position operators can be eliminated from alignment-free grammars.

We describe our process informally on an example; a general description together with correctness theorems can be found in our online paper [9].

As the repetition operator can always be eliminated (by adding a new non-terminal A_{p} with $\delta(A_{\mathsf{p}}) = \mathsf{p}A_{\mathsf{p}}/\varepsilon$ for each subexpression p that occurs under the star operator), we may assume that the input grammar G is repetition-free. The process also assumes that G is well-formed, all negations are applied to atomic expressions, and all choices are disjoint. A choice expression p/q is called *disjoint* if parsing of p and q cannot succeed in the same input state and token mode. *Well-formedness* is a decidable conservative approximation of the predicate that is true iff parsing in G never diverges (it definitely excludes grammars with left recursion but can exclude also some safe grammars). Well-formedness of PEGs was introduced by Ford [6]. Extending the notion to expressions containing the extra operators of PEG$^{>}$ is straightforward, details are provided in [9]. Achieving the other two preconditions can be considered as a preparatory and previously studied (e.g. in [6] as stage 1 of negation elimination) step of the process.

We will work on the example grammar $G = (N, T, \delta, \mathsf{s})$ where $N = \{A, B\}$, $T = \{a, b, c\}$, $\delta = \left\{A \mapsto !!c/a/B, B \mapsto b\lvert AA\rvert^{>}\right\}$ and $\mathsf{s} = A_{\geq}$. This grammar is well-formed and choices in the rule for A are disjoint ($!!c$, a and B can succeed only if the input string starts with c, a or b, respectively). Not all negations are in front of atoms; this can be fixed by introducing a new non-terminal C with rule $C \mapsto !c$ and replacing the rule for A with $A \mapsto !C/a/B$. Elimination of alignment and position operators from the grammar is done in 3 stages.

1. *Transform G to an equivalent grammar G_1 where for each expression of the form pq occurring in δ or s, parsing of p either never succeeds without consuming some input or can succeed only if consuming no input.*
   This "splitting" step enables to later bring alignments through concatenations (by Theorem 2.5). It only modifies rules and the start expression. The new set of rules and start expression could be

   $$\delta_1 = \left\{A \mapsto a/B \quad B \mapsto b\lvert AA/A!!c/!!cA/!!c!!c\rvert^{>}\right\}, \qquad \mathsf{s}_1 = (A/!!c)_{\geq}$$

   (in the version of the grammar with non-terminal C, there would be an additional rule for C that never succeeds). The formal process described in [9] would give a somewhat more complicated result but this simplified variant works fine and perfectly explains the ideas. The alternative with negation is removed from the rule of A to allow parsing of A succeed only if consuming some input (the same transformation would be performed on other non-terminals if it was necessary). The removed alternative (which happens to succeed only if consuming no input) is inserted into each concatenation of A, as well as into the start expression. Basically the same transformation was used by Ford [6] on stage 2 of his negation elimination process.
2. *Using the semantic equivalences of Subsect. 2.3, move all alignments down to atoms. Rewrite alignment of terminals in terms of the position operator and the identity relation. For each existing non-terminal X, introduce a new non-terminal with a rule whose right-hand side is obtained from $\lvert\delta_1(X)\rvert$ by moving all alignments down to non-terminals, and replace all aligned non-terminals with the corresponding new non-terminals.*
   In our example, we have to introduce two new non-terminals A' and B' with right-hand sides obtained from $\lvert a/B\rvert$ and $\lvert b\lvert AA/A!!c/!!cA/!!c!!c\rvert^{>}\rvert$, respectively. Using that

$\lvert AA\rvert \sim \lvert A\rvert A$, $\lvert A!!c\rvert \sim \lvert A\rvert !!c$, $\lvert !!cA\rvert \sim \lvert !!c\rvert\lvert A\rvert$ and $\lvert !!c!!c\rvert \sim \lvert !!c\rvert\lvert !!c\rvert$, we end up with

$$\delta_2 = \left\{ \begin{array}{llll} A & \mapsto a/B & B & \mapsto b(A'A/A'!!c/!!c_\triangle A'/!!c_\triangle !!c_\triangle)^> \\ A' & \mapsto a_\triangle/B' & B' & \mapsto b_\triangle(A'A/A'!!c/!!c_\triangle A'/!!c_\triangle !!c_\triangle)^> \end{array} \right\}, \quad \mathsf{s}_2 = (A/!!c)_\geq .$$

3. *Using the semantic equivalences of Subsect. 2.3, move all position operators down to atoms. For each non-terminal X and relation τ used by position operators, introduce a new non-terminal with a rule whose right-hand side is obtained from $(\delta_2(X))_\tau$ by moving position operators down to atoms, and replace all non-terminals under position operators with the corresponding new non-terminals. Replace position operators applied to terminals with indentation, omit identity indentations.*

In our example, $\geq$ is the only relation used by position operators. Hence we must introduce one new non-terminal for each existing non-terminal. Denote them $\hat{A}$, $\hat{B}$, $\hat{A}'$, $\hat{B}'$. As the old non-terminals will never be used when parsing the new start expression, we can omit their rules. The rules of the new non-terminals and the start expression are

$$\delta_3 = \left\{ \begin{array}{llll} \hat{A} & \mapsto a^\geq/\hat{B} & \hat{B} & \mapsto b^\geq(\hat{A}'\hat{A}/\hat{A}'!!c^\geq/!!c\hat{A}'/!!c!!c)^> \\ \hat{A}' & \mapsto a/\hat{B}' & \hat{B}' & \mapsto b(\hat{A}'\hat{A}/\hat{A}'!!c^\geq/!!c\hat{A}'/!!c!!c)^> \end{array} \right\}, \quad \mathsf{s}_3 = \hat{A}/!!c^\geq .$$

Note how terminals that do not have to be aligned have indentation $\geq$ while terminals to be aligned have no indentation. Parsings in the resulting grammar must run with the alignment flag unset and assume the identity token mode.

At step 1, the sizes of the right-hand sides of the rules can grow exponentially though the number of rules stays unchanged. Preprocessing the grammar via introducing new non-terminals in such a way that all concatenations were applied to atoms (similarly to Ford [6]) would hinder the growth, but the size in the worst case remains exponential. Steps 2 and 3 cause at most a linear growth of right-hand sides.

4 Which relations are good?

Speed of grammar-driven parsing of expressions that involve relations depends on the nature of the relations. The representation of the baseline candidate sets in operation states plays a particular role. Adams and Ağacan [3] prove that, during parsing of expressions that involve only relations $\triangle$, $>$, $\geq$ and $\circledast$, all intermediate sets I occurring in operation states have the form of a connected interval of natural numbers (possibly extending to infinity). This enables to represent any set I by its minimum $\min I$ and supremum $\sup I$ (supremum means maximum for finite sets and ∞ for infinite ones).

In practice, languages may require other indentation relations. Adams [2] mentions $\{(i+2,i) : i \in \mathbb{N}\}$ needed for occam, the `indentation` package [1] implements constant relations $\{(c,i) : i \in \mathbb{N}\}$. Here we generalize the result of [3] by finding a criterion for deciding which indentation relations preserve the interval form of the set of baseline candidates. The result also applies to the relations used by position operators since they matter only during parsing of terminals and the way they are used there is the same as in the case of indentation.

In this section, we denote $l_\rho(i) = \min(\rho(\{i\}))$ and $h_\rho(i) = \sup(\rho(\{i\}))$ for any $\rho \in \Re(\mathbb{N})$ and $i \in \mathbb{N}$. Functions l and h are undefined on i if $\rho(\{i\}) = \varnothing$. *Intervals* are sets of the form $\{j \in \mathbb{N} : n \leq j \leq o\}$ for any $n \in \mathbb{N}$, $o \in \mathbb{N} \cup \{\infty\}$. For uniform treatment, $\varnothing$ is also considered an interval (the case $n > o$ in the definition). This has no bad consequences as the set of baseline candidates is guaranteed to stay non-empty by Theorem 2.1.

4.1 Relations that keep indentation sets as intervals

When parsing of an expression of the form e^{ρ} starts, it must create a new set $\rho^{-1}(I)$ where I is the current set of indentation baseline candidates. There are two obvious conditions that must hold for $\rho^{-1}(I)$ being an interval whenever I is:

1. For each $i \in \mathbb{N}$, $\rho^{-1}(\{i\})$ must be an interval, as I can be a one-element set.
2. For any $i \in \mathbb{N}$, $\rho^{-1}(\{i\}) \cup \rho^{-1}(\{i+1\})$ must be an interval, as I can be $\{i, i+1\}$.

One can easily prove by induction on the size of I that if a relation ρ satisfies conditions 1 and 2 then $\rho^{-1}(I)$ is an interval for any interval I. Note that condition 2 holds iff for any two consecutive natural numbers i and j in any order, $l_{\rho^{-1}}(i) \leq h_{\rho^{-1}}(j) + 1$.

At the end of parsing of an expression of the form e^{ρ}, a new set $I \cap \rho(I')$ must be created to combine the information provided by the set I of baseline candidates for the surrounding indentation and the set I' of baseline candidates for the local indentation. Hence $I \cap \rho(I')$ must be an interval whenever I and I' are. Taking $I = \mathbb{N}$ and $I' = \{i\}$ or $I' = \{i, i+1\}$, we see as before that $\rho(\{i\})$ and $\rho(\{i, i+1\})$ must be intervals for every $i \in \mathbb{N}$. Conversely, an easy induction on the number of elements in I' shows that if all sets $\rho(\{i\})$ and $\rho(\{i, i+1\})$ are intervals then $\rho(I')$ is an interval for any interval I'. As the intersection of two intervals is an interval, this condition is also sufficient for $I \cap \rho(I')$ being an interval.

To conclude, if for each used relation ρ, all sets of the form $\rho^{-1}(\{i\})$, $\rho^{-1}(\{i, i+1\})$, $\rho(\{i\})$ and $\rho(\{i, i+1\})$ are intervals whereby $\rho \in \Re^{+}(\mathbb{N})$, then all sets of baseline candidates occurring in the operation state are intervals during any parsing that starts with an interval as the baseline set. By Theorem 4.2 below, the set $\rho(\{i, i+1\})$ can be omitted from this list, so three out of four conditions remain. For every relation $\rho \in \Re^{+}(\mathbb{N})$ that fails to meet these three conditions, one can find a parsing expression e and an initial state such that a non-interval set appears during parsing. Indeed, the set $\rho^{-1}(I)$ for an arbitrarily chosen finite interval $I = \{i, i+1, \ldots, i+k\}$ is evaluated during parsing of $\mathsf{e} = {\mid}a(bc^{\rho})^{\geq}{\mid}_{\geq}$ on an input string of the shape $a^{i}b^{i+k}w$, and for any $i \in \mathbb{N}$, if $\rho(\{i\})$ is not an interval and hence contains some $n \in \mathbb{N}$ then $\rho(\{i\})$ is evaluated during parsing of $\mathsf{e} = (ab^{\rho})_{\triangle}$ on the input $a^{n}b^{i}$.

4.2 Implementation issues

By condition 1 at the beginning of Subsect. 4.1, any feasible relation ρ is uniquely determined by the pair of functions $(l_{\rho^{-1}}, h_{\rho^{-1}})$. Similarly, ρ is determined by (l_{ρ}, h_{ρ}) because of the analogous condition for $\rho(\{i\})$. Functions $l_{\rho^{-1}}$ and $h_{\rho^{-1}}$ are total as we assume $\rho \in \Re^{+}(\mathbb{N})$, while l_{ρ} and h_{ρ} can be partial. For the four relations considered in [3] for instance,

$$
\begin{array}{lcllcl}
(l_{\triangle^{-1}}(i), h_{\triangle^{-1}}(i)) & = & (i, i); & (l_{\triangle}(i), h_{\triangle}(i)) & = & (i, i); \\
(l_{>^{-1}}(i), h_{>^{-1}}(i)) & = & (i+1, \infty); & (l_{>}(i), h_{>}(i)) & = & (0, i-1) \text{ (provided } i > 0); \\
(l_{\geq^{-1}}(i), h_{\geq^{-1}}(i)) & = & (i, \infty); & (l_{\geq}(i), h_{\geq}(i)) & = & (0, i); \\
(l_{\circledast^{-1}}(i), h_{\circledast^{-1}}(i)) & = & (0, \infty); & (l_{\circledast}(i), h_{\circledast}(i)) & = & (0, \infty).
\end{array}
$$

We recall two well-known notions.

▶ **Definition 4.1.** **1.** Call a function $f : \mathbb{N} \to \mathbb{Z} \cup \{\infty\}$ *non-decreasing* iff, for every $i, j \in \mathbb{N}$, $i \leq j$ implies $f(i) \leq f(j)$.

2. Call a function $f : \mathbb{N} \to \mathbb{Z} \cup \{\infty\}$ *weakly unimodal* iff there exists some $m \in \mathbb{N}$ such that, for every $i, j \in \mathbb{N}$, $i \leq j \leq m$ implies $f(i) \leq f(j)$ and $m \leq i \leq j$ implies $f(i) \geq f(j)$.

Unimodality of f means that the values of f are increasing until some argument m called *mode* and decreasing after that. Weakness specifies that increasing and decreasing can be non-strict (letting values at consecutive arguments equal). We will use also the corresponding

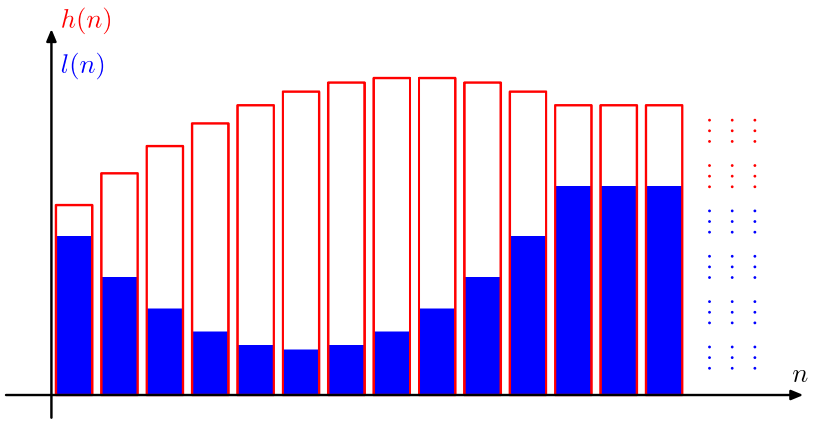

Figure 1 Functions h with the weak unimodality and l with the reverse unimodality property

reverse properties that hold for $f : \mathbb{N} \to \mathbb{Z}$ iff $-f$ has the original property. Thereby f is called *non-increasing* iff $-f$ is non-decreasing. Figure 1 depicts two functions l, h defined on $\mathbb{N}$ such that $l < h$ and both h and $-l$ are weakly unimodal (blue filled and red empty bars depict l and h, respectively).

▶ **Theorem 4.2.** *Let $\rho \in \Re^{+}(\mathbb{N})$ satisfy conditions 1 and 2 at the beginning of Subsect. 4.1. Then the following conditions are equivalent:*

(*) *For every $i \in \mathbb{N}$, $\rho(\{i\})$ is an interval;*
(**) *Each of $h_{\rho^{-1}}$ and $-l_{\rho^{-1}}$ is either non-decreasing or weakly unimodal;*
(***) *For every $i \in \mathbb{N}$, both $\rho(\{i\})$ and $\rho(\{i, i+1\})$ are intervals.*

Proof. Assume (*) and suppose that (**) does not hold. If $h_{\rho^{-1}}$ is neither non-decreasing nor weakly unimodal then there exist $i, j, j' \in \mathbb{N}$, $j + 1 < j'$ such that $i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j'))$ and, for every j'', if $j < j'' < j'$ then $h_{\rho^{-1}}(j'') < i$. By condition 2 assumed by the theorem, $l_{\rho^{-1}}(j) \le h_{\rho^{-1}}(j+1) + 1 \le i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j')) \le h_{\rho^{-1}}(j)$ and similarly $l_{\rho^{-1}}(j') \le h_{\rho^{-1}}(j'-1) + 1 \le i \le \min(h_{\rho^{-1}}(j), h_{\rho^{-1}}(j')) \le h_{\rho^{-1}}(j')$. Hence $\rho(\{i\})$ contains both j and j' but none of the numbers between j and j', which contradicts the fact that $\rho(\{i\})$ is an interval. The case with $-l_{\rho^{-1}}$ being neither non-decreasing nor weakly unimodal is handled analogously. Thus (*) implies (**).

Now assume (**). For every natural number i, denote $H_i = \{j \in \mathbb{N} : h_{\rho^{-1}}(j) \ge i\}$ and $L_i = \{j \in \mathbb{N} : l_{\rho^{-1}}(j) \le i\}$; then $\rho(\{i\}) = H_i \cap L_i$. By (**), H_i and L_i are intervals, hence $\rho(\{i\})$ is an interval for each i. Note that $i < i'$ implies $H_i \supseteq H_{i'}$ and $L_i \subseteq L_{i'}$. Moreover, $H_{i+1} \cup L_i = \mathbb{N}$: Indeed, $j \notin L_i$ implies $l_{\rho^{-1}}(j) > i$, meaning that $h_{\rho^{-1}}(j) \ge l_{\rho^{-1}}(j) \ge i + 1$ whence $j \in H_{i+1}$. Clearly $\rho(\{i, i+1\}) = \rho(\{i\}) \cup \rho(\{i+1\}) = (H_i \cap L_i) \cup (H_{i+1} \cap L_{i+1})$. To prove that $\rho(\{i, i+1\})$ is an interval, suppose that $j \in H_i \cap L_i$, $j' \in H_{i+1} \cap L_{i+1}$ and $j < j'' < j'$ (the case $j' < j'' < j$ is similar). As $H_{i+1} \subseteq H_i$, both j and j' belong to H_i. Similarly as $L_i \subseteq L_{i+1}$, both j and j' belong to L_{i+1}. Consequently, also $j'' \in H_i \cap L_{i+1}$ since both H_i and L_{i+1} are intervals. Now if $j'' \in H_{i+1}$ then $j'' \in H_{i+1} \cap L_{i+1} \subseteq \rho(\{i, i+1\})$, and if $j'' \in L_i$ then $j'' \in H_i \cap L_i \subseteq \rho(\{i, i+1\})$. Hence (**) implies (***).

Finally, (***) trivially implies (*). ◀

Knowing the modes of both $h_{\rho^{-1}}$ and $-l_{\rho^{-1}}$ (in the non-decreasing case with no upper bound, ∞ can be used as the mode), $\rho^{-1}(I)$ can be computed by $O(1)$ evaluations of $l_{\rho^{-1}}$ and $h_{\rho^{-1}}$ and $O(1)$ comparisons of natural numbers for any interval I. Indeed, $\sup(\rho^{-1}(I))$

equals the value of $h_{\rho^{-1}}$ at one of the endpoints of I or at the mode (if the mode belongs to I), or is ∞ (if $h_{\rho^{-1}}$ is non-decreasing and unbounded); $\min(\rho^{-1}(I))$ can be found similarly.

The set of natural numbers where h_ρ and l_ρ are defined is an interval. An argument similar to the proof of part (*) $\Rightarrow$ (**) of Theorem 4.2 shows that each of h_ρ and $-l_\rho$ is either non-decreasing or weakly unimodal within its domain of definition. Hence by symmetry, it is possible to compute $\rho(I')$ for any interval I' by $O(1)$ evaluations of l_ρ and h_ρ and $O(1)$ comparisons if the modes of both h_ρ and $-l_\rho$ are known. Partiality of h_ρ and l_ρ is not an issue since Theorem 2.1 guarantees that the set of indentation baseline candidates becomes never empty. Obviously the intersection of known intervals I and $\rho(I')$ is computable by $O(1)$ comparisons.

Consequently, by representing relations ρ with records that consist of functions l_ρ, h_ρ, $l_{\rho^{-1}}$ and $h_{\rho^{-1}}$ together with the modes of $-l_\rho$, h_ρ, $-l_{\rho^{-1}}$ and $h_{\rho^{-1}}$, every indentation causes only an $O(1)$ time overhead (if the values of the functions are computable in $O(1)$ time). It is reasonable to expect that the parser implementer provides the right representations for all the relations in use as the number of these relations is normally quite small.

For the four relations in [3], the modes can be defined as follows (they are not unique as the functions can increase or decrease non-strictly):

$$\begin{array}{lcllcl}
(m(-l_{\triangle^{-1}}), m(h_{\triangle^{-1}})) & = & (0,\infty); & (m(-l_{\triangle}), m(h_{\triangle})) & = & (0,\infty);\\
(m(-l_{>^{-1}}), m(h_{>^{-1}})) & = & (0,0); & (m(-l_{>}), m(h_{>})) & = & (1,\infty);\\
(m(-l_{\geq^{-1}}), m(h_{\geq^{-1}})) & = & (0,0); & (m(-l_{\geq}), m(h_{\geq})) & = & (0,\infty);\\
(m(-l_{\circledast^{-1}}), m(h_{\circledast^{-1}})) & = & (0,0); & (m(-l_{\circledast}), m(h_{\circledast})) & = & (0,0).
\end{array}$$

5 Related work

PEGs were first introduced and studied by Ford [6] who also showed them to be closely related with the TS system [5] and TDPL [4], as well as to their generalized forms [5, 4].

Adams [2] and Adams and Ağacan [3] provide an excellent overview of previous approaches to describing indentation-sensitive languages and attempts of building indentation features into parser libraries. Our work is a theoretical study of the approach proposed in [3] while some details of the semantics used in our paper were "corrected" in the lines of Adams' `indentation` package for Haskell [1]. This package enables specifying indentation sensitivity within the Parsec and Trifecta parser combinator libraries. A process of alignment operator elimination is previously described for CFGs by Adams [2].

Matsumura and Kuramitsu [7] develop a very general extension of PEG that also enables to specify indentation. Their framework is powerful but complicated. The approach proposed in [3] and followed by us is in contrast with [7] by focusing on indentation and aiming to maximal simplicity and convenience of usage.

6 Conclusion

We studied the extension of PEG proposed by Adams and Ağacan [3] for indentation-sensitive parsing. This extension uses operators for marking indentation and alignment besides the classic ones. Having added one more operator (position) for convenience, we found a lot of useful semantic equivalences that are valid on expressions written in the extended grammars. We applied these equivalences subsequently for defining a process that algorithmically eliminates all alignment and position operators from well-formed grammars.

We analyzed practical limitations of the indentation extension of PEG from the aspect of efficient expressibility and computability of the relations and sets needed during parsing.

We found a wide class of relations that, provided the minimum and supremum of the set of numbers related to any given number is computable in $O(1)$ time, cause only $O(1)$ overhead at each parsing step.

Acknowledgements. I thank the anonymous reviewers whose comments helped to improve the paper considerably.

References

1 Michael D. Adams. The `indentation` package. URL: `http://hackage.haskell.org/package/indentation`.

2 Michael D. Adams. Principled parsing for indentation-sensitive languages: Revisiting Landin's offside rule. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Rome, Italy - January 23 - 25, 2013*, pages 511–522. ACM, 2013. `doi:10.1145/2429069.2429129`.

3 Michael D. Adams and Ömer S. Ağacan. Indentation-sensitive parsing for Parsec. In Wouter Swierstra, editor, *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell, Gothenburg, Sweden, September 4-5, 2014*, pages 121–132. ACM, 2014. `doi:10.1145/2633357.2633369`.

4 Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

5 Alexander Birman and Jeffrey D. Ullman. Parsing algorithms with backtrack. *Information and Control*, 23(1):1–34, 1973. `doi:10.1016/S0019-9958(73)90851-6`.

6 Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 111–122. ACM, 2004. `doi:10.1145/964001.964011`.

7 Tetsuro Matsumura and Kimio Kuramitsu. A declarative extension of parsing expression grammars for recognizing most programming languages. *JIP*, 24(2):256–264, 2016. `doi:10.2197/ipsjjip.24.256`.

8 Sérgio Medeiros, Fabio Mascarenhas, and Roberto Ierusalimschy. Left recursion in parsing expression grammars. In Francisco Heron de Carvalho Junior and Luís Soares Barbosa, editors, *Programming Languages - 16th Brazilian Symposium, SBLP 2012, Natal, Brazil, September 23-28, 2012. Proceedings*, volume 7554 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2012. `doi:10.1007/978-3-642-33182-4_4`.

9 Härmel Nestra. Alignment elimination from Adams' grammars, 2017. `arXiv:1706.06497`.