Alignment Elimination from Adams’ Grammars

Härmel Nestra

1 Institute of Computer Science, University of Tartu, J. Liivi 2, 50409 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 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.

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

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 \( G = (N, T, \delta, s) \) where:

- \( N \) is a finite set of non-terminals;
- \( T \) is a finite set of terminals;
- \( \delta \) 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 \( \delta : N \rightarrow E_G \) and \( s \in E_G \), where the set \( E_G \) of all parsing expressions writable in \( G \) is defined inductively as follows:

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

\[
\begin{align*}
\text{doexp} & ::= \text{do}\ (<\text{stmts}>) / <\text{stmts}> \\
\text{stmts} & ::= (|\text{<stmt>}|^+) \\
\text{stmt} & ::= (\text{<stmt>};\text{<stmt>})^*[;]$
\end{align*}
\]

Here, $<\text{stmts}>$ and $<\text{stmt}>$ 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 $|$). In the relaxed mode, however, relation $@$ is used to indicate that the indentation baseline of the contents can be anything. (Technically, $@$ 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 fulcrum 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. Section 4 refers to related work and Sect. 5 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 $\mathcal{R}(X)$ denote the set of all binary relations on set $X$, i.e., $\mathcal{R}(X) = \wp(X \times X)$. Standard examples are $> \in \mathcal{R}(\mathbb{N})$ (consisting of all pairs $(n, m)$ of natural numbers such that $n > m$) and $\triangle \in \mathcal{R}(\mathbb{N})$ (the identity
relation consisting of all pairs of equal natural numbers); the indentation extension also
makes use of \( \oplus \in \mathbb{R}(\mathbb{N}) \) (the relation containing all pairs of natural numbers). Whenever
\( \varphi \in \mathbb{R}(X) \) and \( Y \subseteq X \), denote \( \varphi(Y) = \{ x \in X : \exists y \in Y. (y, x) \in \varphi \} \) (the image of \( Y \) under
relation \( \varphi \)). The inverse relation of \( \varphi \) is defined by \( \varphi^{-1} = \{ (x, y) : (y, x) \in \varphi \} \), and the
composition of relations \( \sigma \) and \( \varphi \) by \( \sigma \circ \varphi = \{ (x, z) : \exists y. (x, y) \in \sigma \land (y, z) \in \varphi \} \). Finally,
denote \( \mathbb{R}^{+}(X) = \{ \varphi \in \mathbb{R}(X) : \forall x \in X. \varphi^{-1}(\{ x \}) \neq \emptyset \} = \{ \varphi \in \mathbb{R}(X) : \varphi(X) = X \} \).

### 2.1 Adams' grammars

Extend the definition of \( \mathcal{E}_G \) given in Sect. 1.1 with the following three additional clauses:

8. \( \rho^\varphi \in \mathcal{E}_G \) for every \( \rho \in \mathcal{E}_G \) and \( \varphi \in \mathbb{R}(\mathbb{N}) \) (indentation);
9. \( \rho_\sigma \in \mathcal{E}_G \) for every \( \rho \in \mathcal{E}_G \) and \( \sigma \in \mathbb{R}(\mathbb{N}) \) (token position);
10. \( \rho^I \in \mathcal{E}_G \) for every \( \rho \in \mathcal{E}_G \) (alignment).

Parsing of an expression \( \rho^\varphi \) means parsing of \( \rho \) while assuming that the part of the input
string corresponding to \( \rho \) forms a new indentation block whose baseline is in relation \( \varphi \) to the
baseline of the surrounding block. (Baselines are identified with column numbers.) The
position construct \( \rho_\sigma \), missing in [3], determines how tokens of the input can be situated w.r.t.
the current indentation baseline. Finally, parsing an expression \( \rho^I \) means parsing of \( \rho \) 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 \cite{1}, we call the relations that determine token
positioning w.r.t. the indentation baseline \textit{token modes}. In the token mode \( \tau \) for example,
tokens may appear only to the right of the indentation baseline. Applying the position operator
with relation \( \tau \) to parts of Haskell grammar to be parsed in the indentation mode
avoids indenting every single terminal in the example in Sect. 1.1. Also, indenting terminals
with \( \tau \) 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\( \tau \). Recall from Sect. 1.1 that \( \mathcal{N} \)
and \( \mathcal{T} \) denote the set of non-terminal and terminal symbols of the grammar, respectively, and
\( \delta : \mathcal{N} \to \mathcal{E}_G \) is the production function. Concerning the semantics of PEG\( \tau \), each expression
parses an input string of terminals \( (w \in \mathcal{T}^\ast) \) in the context of a current set of indentation
baseline candidates \( (I \in \mathcal{P}(\mathbb{N})) \) and a current alignment flag indicating whether the next
terminal should be aligned or not \( (b \in \mathbb{B}) \), assuming a certain token mode \( (\tau \in \mathbb{R}(\mathbb{N})) \).
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 \( (w, I, b) \in \mathcal{T}^\ast \times \mathcal{P}(\mathbb{N}) \times \mathbb{B} \) are behaving as \textit{operation states} of parsing, as
each parsing step may use these data and update them. We will write \( \text{State} = \mathcal{T}^\ast \times \mathcal{P}(\mathbb{N}) \times \mathbb{B} \)
as we never deal with different terminal sets, dependence on \( T \) is not explicitly marked,
and denote by \( \text{State} + 1 \) the set of possible results of parsing, i.e., \( \{ \top(s) : s \in \text{State} \} \cup \{ \bot \} \).

The assertion that parsing expression \( e \) in grammar \( G \) with input string \( w \) in the context
of \( I \) and \( b \) assuming token mode \( \tau \) results in \( o \in \text{State} + 1 \) 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
\( G, \tau \vdash e \to o \) 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 \( a \in \mathcal{T}, a, \tau \vdash_G (w, I, b) \to o \) holds in two cases:
Alignment Elimination from Adams’ Grammars

By clause 2, we have in such cases (not even 0, >, similarity operates on the alignment flag. I indentation baseline must be in relation 0 with the current indentation baseline known to be 0 if no alignment flag is set, and coincide with the indentation baseline otherwise. For the same reason, consuming a token in column i restricts the set of allowed indentations to [i] or {i} depending on the alignment flag. In both cases, the alignment flag is set to 0. In clause 8 for 0, the set 0 of allowed indentation is replaced by 0 (I) as the local indentation baseline must be in relation 0 with the current indentation baseline known to be in I. After successful parsing of 0 with the resulting set of allowed local indentations being I, the set of allowed indentations of the surrounding block is restricted to (I). Clause 10 similarly operates on the alignment flag.

For a toy example, consider parsing of \([ab]^>\) with the operation state \((a^2b^3, [N], 0f)\) assuming the token mode 0. For that, we must parse \([ab]\) with \((a^2b^3, N \setminus \{0\}, 0f)\) by clause 8 since \(0^{-1} (N) = N \setminus \{0\}\). For that in turn, we must parse \(ab\) with \((a^2b^3, N \setminus \{0\}, tt)\) by clause 10. By clause 2, we have \(a, \geq \rightarrow (a^2b^3, N \setminus \{0\}, tt) \rightarrow (b^3, \{2\}, 0f)\) (as \(2 \in N \setminus \{0\}\)) and \(b, \geq \rightarrow (b^3, \{2\}, 0f)\) (as \(2.3 \in 0^{-1}\)). Therefore, by clause 4, \(ab, \geq \rightarrow (a^2b^3, N \setminus \{0\}, tt) \rightarrow (e, \{2\}, 0f)\). Finally, \([ab]^>\geq \rightarrow (a^2b^3, N \setminus \{0\}, 0f) \rightarrow (e, \{2\}, 0f)\) and \([ab]^>\geq \rightarrow (a^2b^3, N, 0f) \rightarrow (e, \{0, 1\}, 0f)\) 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 N).

Note that this definition involves circular dependencies. For instance, if \(S(X) = X\) for some \(X \in N\) then \(X, \rightarrow o\) if \(X, \rightarrow o\) by clause 3. There is no result of parsing in such cases (not even 0). We call this behaviour divergence.
2.2 Properties of the semantics

Ford [5] proves that parsing in PEG is unambiguous, whereby the consumed part of an input string always is its prefix. Theorem 2.3 below is an analogous result for PEG\textsuperscript{\textgreater}. Besides the uniqueness of the result of parsing, it states that if we only consider relations in \( R^+(N) \) 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 \( \sqsubseteq \) the implication order of truth values (i.e., \( tt \sqsubseteq ff \)). Denote by \( \geq \) the pointwise order on operation states, i.e., \( (w, I, b) \geq (w', I', b') \) iff \( w \geq w', I \supseteq I' \) and \( b \sqsubseteq b' \).

**Theorem 2.3.** Let \( G = (N, T, \delta, s) \) be a PEG\textsuperscript{\textgreater}, \( e \in E_G \), \( \tau \in R^+(N) \) and \( s \in \text{State} \). Then \( e, \tau \vdash_G s \rightarrow o \) for at most one \( o \), whereby \( o = \top(s') \) implies \( s \geq s' \). Also if \( s = (w, I, b) \) and \( s' = (w', I', b') \) then \( s \neq s' \) implies both \( w \geq w' \) and \( b' \sqsubseteq b \), and \( I \neq \emptyset \) implies \( I' \neq \emptyset \).

**Proof.** By induction on the shape of the derivation tree of the assertion \( e, \tau \vdash_G s \rightarrow o \).

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

For \( p \in E_G \), \( \kappa(p) \), \( \tau \vdash_G s \rightarrow o \) holds in two cases:
- If there exists a state \( s' \) such that \( p, \tau \vdash_G \alpha(s) \rightarrow \top(s') \) and \( o = \top(s \land \gamma(s')) \);
- If \( p, \tau \vdash_G \alpha(s) \rightarrow \bot \) and \( o = \bot \).

The meanings of indentation and alignment constructs are distinguished solely by \( \alpha \) and \( \gamma \). For many properties, proofs that rely on this abstract common definition can be carried out, assuming that \( \gamma \) is monotone, preserves the largest element and follows together with \( \alpha \) the axiom \( x \land \gamma(y) \leq \gamma(x \land y) \). The class of all meet semilattices \( L \) with top element, equipped with mappings \( \alpha, \gamma \) satisfying these three conditions, contains identities (i.e., semilattices \( L \) with \( \alpha = \gamma = \text{id}_L \)) and is closed under compositions (of different \( \alpha \) and \( \gamma \)) on the same semilattice \( L \).

We say that \( \alpha \) and \( \gamma \) are **semantically equivalent** in \( G \) and denote \( p \sim_G q \) iff \( p, \tau \vdash_G s \rightarrow o \iff q, \tau \vdash_G s \rightarrow o \) for every \( \tau \in R^+(N) \), \( s \in \text{State} \) and \( o \in \text{State} + 1 \).

For example, one can easily prove that \( p^\# \sim_G p \sim_G \varepsilon p, p(qr) \sim_G (pq)r, p/(pq)/r \sim_G (p/q)/r, p(q/r) \sim_G pq/pr, p/q \sim_G p/lpq \) for all \( p, q, r \in E_G \) [6]. We are particularly interested in equivalences involving the additional operators of PEG\textsuperscript{\textgreater}. 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\textsuperscript{\textgreater} w.r.t. other constructs:

**Theorem 2.3.** Let \( G = (N, T, \delta, s) \) be a PEG\textsuperscript{\textgreater}. Then:
Alignment Elimination from Adams’ Grammars

1. \( \varepsilon \sim_G \varepsilon, (pq)\varepsilon \sim_G p\varepsilon q\varepsilon, (p/q)\varepsilon \sim_G p\varepsilon /q\varepsilon, (!p)\varepsilon \sim_G !p\varepsilon, (p^*)\varepsilon \sim_G (p\varepsilon)^* \), \( (p\varepsilon)^* \sim_G \varepsilon \).

2. \( \varepsilon^p \sim_G \varepsilon, (p^q)^p \sim_G p^q /q^p, (p)^q \sim_G (p\varepsilon)^q \), \( (p\varepsilon)^q \sim_G (p\varepsilon)^q \).

3. \( \varepsilon_i \sim_G \varepsilon, [p/q] \sim_G [p]/[q], [p] \sim_G [q], [p\varepsilon] \sim_G [q\varepsilon] \).

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 \( \alpha, \gamma \). □

Note that indentation does not distribute with concatenation, i.e., \( (pq)^p \sim_G p^q q^p \). This is because \( (pq)^p \) assumes one indentation block with a baseline common to \( p \) and \( q \). For example, take \( p = a \in T \), \( q = b \in T \), let the token mode be \( \Delta \) and the input state be \( (a \times b, N, \varepsilon) \) (recall that \( a \times b \) means terminal \( a \) occurring in column \( i \)). We have \( a, \Delta \vdash_G (a \times b, N \setminus \{0\}, \varepsilon) \). and \( b, \Delta \vdash_G (b, \{1\}, \varepsilon) \) to \( \perp \). (since \( 2, 1 \notin \Delta \)). therefore \( a, \Delta \vdash_G (a \times b, N \setminus \{0\}, \varepsilon) \) to \( \perp \) and \( (ab)^p \). \( \Delta \vdash_G (a \times b, N \setminus \{0\}, \varepsilon) \) to \( \perp \). On the other hand, \( a, \Delta \vdash_G (a \times b, N \setminus \{0\}, \varepsilon) \) to \( \perp \). analogously, \( b^2, \Delta \vdash_G (b^2, \{0\}, \varepsilon) \) to \( (\varepsilon, \{0\}, \varepsilon) \) (since \( \{\} \) \( \leq \{\} \) \( \leq \{0\} \)) \( \perp \). Consequently, \( a \times b^2, \Delta \vdash_G (a \times b^2, N, \varepsilon) \) to \( \perp \).

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 \( p \in E_G \), \( p^\Delta \sim_G p \).
2. Composition law of indentations: For all \( p \in E_G \) and \( \varepsilon, \sigma \in R^+ \), \( (p\varepsilon)^p \sim_G p^o p^w \).
3. Distributivity of indentation and alignment: For all \( p \in E_G \) and \( \varepsilon, \sigma \in R^+ \), \( [p\varepsilon]^p \sim_G [p]^p \).
4. Idempotence of alignment: For all \( p \in E_G \), \( [p]^p \sim_G [p]^p \).
5. Cancellation of outer token modes: For all \( p \in E_G \) and \( \varepsilon, \tau \in R(N) \), \( (p\varepsilon)^\tau \sim_G p \).
6. Terminal alignment property: For all \( a \in T \), \( |a| \sim_G a\Delta \).

Proof. For claim 1, note that an indentation with the identity relation \( \Delta \) corresponds to \( a \), \( \gamma \) being identity mappings. Hence

\[
p^\Delta, \tau \vdash_G s \rightarrow o \iff \begin{cases} \exists s', p, \tau \vdash_G s \rightarrow T(s') \land o = T(s \land s') \\
\text{or} \\
p, \tau \vdash_G s \rightarrow \perp \land o = \perp \\
\end{cases}
\]

\[
\iff \begin{cases} \exists s', p, \tau \vdash_G s \rightarrow T(s') \land o = T(s') \\
\text{or} \\
p, \tau \vdash_G s \rightarrow \perp \land o = \perp \\
\end{cases}
\]

\[
\iff p, \tau \vdash_G s \rightarrow o,
\]

where \( s \land s' \) can be replaced with \( s' \) because \( s \geq s' \) by Theorem 2.1.

Concerning claims 2–4, let \( k_1, k_2 \) be two constructs whose semantics follow the common pattern of indentation and alignment with mapping pairs \( (\alpha_1, \gamma_1) \) and \( (\alpha_2, \gamma_2) \), respectively. Then

\[
k_2(k_1(p)), \tau \vdash_G s \rightarrow o
\]

\[
\iff \begin{cases} \exists s', k_1(p), \tau \vdash_G \alpha_2(s) \rightarrow T(s') \land o = T(s \land \gamma_2(s')) \text{ or} \\
\text{or} \\
k_1(p), \tau \vdash_G \alpha_2(s) \rightarrow \perp \land o = \perp \\
\end{cases}
\]

\[
\iff \begin{cases} \exists s'', p, \tau \vdash_G \alpha_1(\alpha_2(s)) \rightarrow T(s'') \land o = T(s \land \gamma_2(\alpha_2(s) \land \gamma_1(s''))) \text{ or} \\
\text{or} \\
p, \tau \vdash_G \alpha_1(\alpha_2(s)) \rightarrow \perp \land o = \perp \\
\end{cases}
\]
By monotonicity of \( \gamma_2 \) and the fact that \( s \leq \gamma_2(\alpha(s)) \), we have \( s \land \gamma_2(\alpha(s) \land \gamma_1(s')) \leq s \land \gamma_2(\alpha_2(s)) \land \gamma_2(\gamma_1(s')) = s \land \gamma_2(\gamma_1(s')) \). By the third axiom of \( \alpha_2 \) and \( \gamma_2 \), we also have \( \gamma_2(\alpha_2(s) \land \gamma_1(s')) \geq s \land \gamma_2(\gamma_1(s')) \) whence \( s \land \gamma_2(\alpha_2(s) \land \gamma_1(s')) \geq s \land \gamma_2(\gamma_1(s')) \). Consequently, \( s \land \gamma_2(\alpha_2(s) \land \gamma_1(s')) \) can be replaced with \( s \land \gamma_2(\gamma_1(s')) \). Hence the semantics of the composition of \( \mathcal{F} \) and \( \mathcal{F}_2 \) follows the pattern of semantics of indentation and alignment for mappings \( \alpha_1 \circ \alpha_2 \) and \( \gamma_2 \circ \gamma_1 \). To prove claim 2, it now suffices to observe that the mappings \( \alpha, \gamma \) in the semantics of (\( \cdot \))\( ^{\alpha \gamma} \) equal the compositions of the corresponding mappings for the semantics of (\( \cdot \))\( ^{\alpha} \) and (\( \cdot \))\( ^{\gamma} \). For claim 3, it suffices to observe that the mappings \( \alpha, \gamma \) given for an indentation and for alignment modify different parts of the operation state whence their order of application is irrelevant. Claim 4 holds because the mappings \( \alpha, \gamma \) in the alignment semantics are both idempotent.

Finally, claim 5 is trivial and 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 \( \langle p q \rangle \), 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\( ^{>\ast} \) and \( p, q \in \mathcal{E}_G \).

1. If \( p, \tau \vdash_G s \rightarrow \top(s') \) implies \( s' = s \) for all \( \tau \in \mathbb{R}^+(\mathbb{N}), s, s' \in \text{State} \), then \( \| pq \| \sim_G \| p \| \| q \| \).
2. If \( p, \tau \vdash_G s \rightarrow \top(s') \) implies \( s' \neq s \) for all \( \tau \in \mathbb{R}^+(\mathbb{N}), s, s' \in \text{State} \), 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 \( \alpha, \gamma \) 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\( ^{>\ast} \). Let \( a \in T, \sigma \in \mathbb{R}^+(\mathbb{N}) \) and \( w \in T^\ast, I \in \varphi(\mathbb{N}), o \in \text{State} + 1 \). Then \( a_\sigma, \Delta \vdash_G (w, I, \varnothing) \rightarrow o \iff a_\sigma, \Delta \vdash_G (w, I, \varnothing) \rightarrow o \).

**Proof.** Straightforward case study.

### 2.4 Differences of our approach from previous work

Our specification of PEG\( ^{>\ast} \) 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 \( p_\sigma \) 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 \( \langle p \rangle \). 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 \( \langle p \rangle \) should be restored in this case.

This is the purpose of conjunction in the alignment semantics described in this paper.
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.

3.1 Approximation semantics and well-formed expressions

For defining well-formedness, we first need to introduce approximation semantics that consists of assertions of the form $e \Rightarrow_G n$ where $e \in \mathcal{E}_G$ and $n \in \{-1, 0, 1\}$. This semantics is a decidable extension of the predicate that tells whether parsing of $e$ may succeed with consuming no input (result 0), succeed with consuming some input (result 1) or fail (result −1). No particular input strings, indentation sets etc. are involved, whence the semantics is not deterministic. The following set of clauses define the approximation semantics inductively.

1. $\varepsilon \Rightarrow_G 0$.
2. For every $a \in T$, $a \Rightarrow_G 1$ and $a \Rightarrow_G −1$.
3. For every $X \in N$, $X \Rightarrow_G n$ if $\delta(X) \Rightarrow_G n$.
4. For every $p, q \in \mathcal{E}_G$, $pq \Rightarrow_G n$ holds in four cases:
   - $p \Rightarrow_G 0$, $q \Rightarrow_G 0$ and $n = 0$;
   - There exist $n', n'' \in \{0, 1\}$ such that $p \Rightarrow_G n'$, $q \Rightarrow_G n''$, $1 \in \{n', n''\}$ and $n = 1$;
   - There exists $n' \in \{0, 1\}$ such that $p \Rightarrow_G n'$, $q \Rightarrow_G −1$ and $n = −1$;
   - $p \Rightarrow_G −1$ and $n = −1$.
5. For every $p, q \in \mathcal{E}_G$, $p/q \Rightarrow_G n$ holds in two cases:
   - $p \Rightarrow_G n$ and $n \in \{0, 1\}$;
   - $p \Rightarrow_G −1$ and $q \Rightarrow_G n$.
6. For every $p \in \mathcal{E}_G$, $!p \Rightarrow_G n$ holds in two cases:
For every

7. For every \( p \in E_G, p^* \rightarrow_G n \) holds in two cases:
   - \( p \rightarrow_G -1 \) and \( n = 0 \);
   - There exists \( n' \in \{0, 1\} \) such that \( p \rightarrow_G n' \) and \( n = -1 \).
8. For every \( p \in E_G \) and \( q \in \mathbb{R}(N), p^* \rightarrow_G n \) if \( p \rightarrow_G n \).
9. For every \( p \in E_G \) and \( \sigma \in \mathbb{R}(N), p_\sigma \rightarrow_G n \) if \( p \rightarrow_G n \).
10. For every \( p \in E_G, |p| \rightarrow_G n \) if \( p \rightarrow_G n \).

On the PEG constructs (1–7), our definition basically copies that given by Ford [6], except for the case \( p^* \rightarrow_G 1 \) where our definition requires \( p \rightarrow_G -1 \) besides \( p \rightarrow_G 1 \). This is sound since if parsing of \( p \) never fails then parsing of \( p^* \) cannot terminate. The difference does not matter in the grammar transformations below as they assume repetition-free grammars.

\[ \textbf{Theorem 3.1.} \text{Let } G = (N, T, \delta, s) \text{ be a PEG}\^+ . \text{ Assume that } e, \tau \vdash_G s \rightarrow o \text{ for some } \tau \in \mathbb{R}(N) \text{ and } s \in \text{State}, o \in \text{State} + 1. \text{ Then:} \]

1. If \( o = \top(s) \) then \( e \rightarrow_G 0 \);
2. If \( o = \top(s') \) for some \( s' \neq s \) then \( e \rightarrow_G 1 \);
3. If \( o = \bot \) then \( e \rightarrow_G -1 \).

\[ \textbf{Proof.} \text{ By induction on the shape of the derivation tree of the assertion } e, \tau \vdash_G s \rightarrow o. \]

\( \textbf{Well-formedness} \text{ is a decidable conservative approximation of the predicate that is true iff parsing in } G \text{ never diverges (it definitely excludes grammars with left recursion but can exclude also some safe grammars). \textbf{Well-formedness} of PEGs was introduced by Ford [6]. The following set of clauses is an inductive definition of predicate } WF_G, \text{ well-formedness of expressions, for } PEG^+ : \]

1. \( \varepsilon \in WF_G \);
2. For every \( a \in T, a \in WF_G \);
3. For every \( X \in N, X \in WF_G \) if \( \delta(X) \in WF_G \);
4. For every \( p, q \in E_G, pq \in WF_G \) if \( p \in WF_G \) and, in addition, \( p \rightarrow_G 0 \) implies \( q \in WF_G \);
5. For every \( p, q \in E_G, p/q \in WF_G \) if \( p \in WF_G \) and, in addition, \( p \rightarrow_G -1 \) implies \( q \in WF_G \);
6. For every \( p \in E_G, \sigma \in \mathbb{R}^+(N), p_\sigma \in WF_G \) if \( p \in WF_G \);
7. For every \( p \in E_G, p^* \in WF_G \) if \( p \neq 0 \) and \( p \in WF_G \);
8. For every \( p \in E_G, \sigma \in \mathbb{R}^+(N), p_\sigma \in WF_G \) if \( p \in WF_G \);
9. For every \( p \in E_G \) and \( \sigma \in \mathbb{R}^+(N), p_\sigma \in WF_G \) if \( p \in WF_G \);
10. For every \( p \in E_G, |p| \in WF_G \) if \( p \in WF_G \).

This definition rejects non-terminals with directly or indirectly left recursive rules since for a concatenation \( pq \) to be well-formed, \( p \) must be well-formed, leading to an infinite derivation in the case of any kind of left recursion. On the other hand, requiring both \( p \in WF_G \) and \( q \in WF_G \) in the clause for \( pq \in WF_G \) would be too restrictive since this would reject non-terminals with meaningful recursive productions like \( X \Rightarrow aX/\varepsilon \).

Again, clauses for PEG constructs (1–7) mostly copy the definition given by Ford [6]. This time, the choice case is an exception. In [6], \( p/q \) is considered well-formed only if both \( p \) and \( q \) are well-formed, which needlessly rejects non-terminals with safe recursive rules like \( X \Rightarrow \varepsilon/X \). We require \( q \in WF_G \) only if \( q \) could possibly be executed, i.e. if \( p \rightarrow_G -1 \).

A grammar \( G = (N, T, \delta, s) \) is called \textbf{well-formed} if \( p \in WF_G \) for every expression \( p \) occurring as a subexpression in some \( \delta(X) \) or \( s \). Ford [6] proves by induction on the length
Alignment Elimination from Adams’ Grammars

of the input string that, in well-formed grammars, parsing of every expression whose all subexpressions are well-formed terminates on every input string. We can prove an analogous result in a similar way but we prefer to generalize the statement to a stricter semantics which enables to occasionally construct easier proofs later. The new semantics, which we call strict, is defined by replacing the choice clause in the definition of Subsect. 2.1 with the following:

5. For every \( p, q \in E_G \), \( p/q, \tau \vdash_G s \rightarrow o \) holds in two cases:
   - There exists a triple \( s' \) such that \( p, \tau \vdash_G s \rightarrow \top(s') \), \( o = \top(s') \), and, in addition, \( p \rightarrow_G -1 \) implies \( q, \tau \vdash_G s \rightarrow o' \) for some \( o' \in \text{State} + 1 \);
   - \( p, \tau \vdash_G s \rightarrow \bot \) and \( q, \tau \vdash_G s \rightarrow o \).

The new semantics is more restrictive since, to finish parsing of an expression of the form \( p/q \) after parsing \( p \) successfully, also \( q \) must be parsed if \( p \rightarrow_G -1 \) happens to be the case. In the standard semantics, parsing of \( p/q \) does not have to try \( q \) if parsing of \( p \) is successful. So, if parsing of an expression terminates in the strict semantics then it terminates with the same result in the standard semantics (but not necessarily vice versa). Therefore proving that parsing always gives a result in the strict semantics will establish this also for the standard semantics. In the rest, we sign strict semantics with exclamation mark, i.e., parsing assertions will be of the form \( e, \tau \vdash_G s \rightarrow o \).

\[ \text{Theorem 3.2.} \quad \text{Let } G = (N, T, \delta, s) \text{ be a well-formed PEG}^\ast \text{ and let } e \in E_G. \text{ Assume that all subexpressions of } e \text{ are well-formed. Then for every } \tau \in R^+(N) \text{ and } s \in \text{State}, \text{ there exists } o \in \text{State} + 1 \text{ such that } e, \tau \vdash_G s \rightarrow o. \]

\[ \text{Proof.} \quad \text{By induction on the length of the input string (i.e., the first component of } s). \text{ The induction step uses induction on the shape of the derivation tree of the assertion } e \in WF_G. \]

### 3.2 Splitting

As the repetition operator can always be eliminated (by adding a new non-terminal \( A_p \) with \( \delta(A_p) = pA_p/\varepsilon \) for each subexpression \( p \) that occurs under the star operator), we may assume that the input grammar \( G \) is repetition-free. The first stage of our 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. Achieving the last 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. Issues concerning this are discussed briefly in Subsect. 3.5.

We use in principle the same splitting algorithm as in stage 2 of the negation elimination process described by Ford [6], adding clauses for the extra operators in PEG\(^\ast\). The approach defines two functions \( \gamma_0 : WF_G \rightarrow E_G \) and \( \gamma_1 : E_G \rightarrow E_G \) as follows (\( F \) is a metavariable denoting any expression that always fails, e.g., \( \varepsilon \)):

\[
\gamma_0(\varepsilon) = \varepsilon \quad \gamma_0(a) = F \quad \gamma_0(X) = \gamma_0(\delta(X)) \quad \gamma_0(pq) = \begin{cases} \gamma_0(p)\gamma_0(q) & \text{if } p \rightarrow_G 0 \\ F & \text{otherwise} \end{cases} \quad \gamma_0(p/q) = \begin{cases} \gamma_0(p)/\gamma_0(q) & \text{if } p \rightarrow_G -1 \\ \gamma_0(p) & \text{otherwise} \end{cases} \quad \gamma_0(!p) = !\gamma_0(p) \quad \gamma_0(p^\ast) = (\gamma_0(p))^\ast \quad \gamma_0(p_a) = (\gamma_0(p))_a \quad \gamma_0(|p|) = |\gamma_0(p)| \quad \gamma_1(\varepsilon) = F \quad \gamma_1(a) = a \quad \gamma_1(X) = X \quad \gamma_1(pq) = \gamma_1(p)\gamma_1(q)/\gamma_1(p)\gamma_0(q)/\gamma_0(p)\gamma_1(q) \quad \gamma_1(p/q) = \begin{cases} \gamma_1(p)/\gamma_1(q) & \text{if } p \rightarrow_G -1 \\ \gamma_1(p) & \text{otherwise} \end{cases} \quad \gamma_1(!p) = F \quad \gamma_1(p^\ast) = (\gamma_1(p))^\ast \quad \gamma_1(p_a) = (\gamma_1(p))_a \quad \gamma_1(|p|) = |\gamma_1(p)|
\]

\( \Box \)
Correctness of the definition of $\gamma_0$ follows by induction on the shape of the derivation tree of the assertion $e \in WF_G$. In the negation case, we use that negations are applied to atomic expressions, whence the reference to $\gamma_1$ can be eliminated by a replacement from its definition. The definition of $\gamma_1$ is sound by induction on the shape of the expression $e$.

A new grammar $G' = (N, T, \delta', s')$ is defined using $\gamma_0, \gamma_1$ by equations $\delta'(X) = \gamma_1(\delta(X))$, $s' = \gamma_1(s)/\gamma_1(s)$. The equivalence of the input and output grammars relies on the splitting invariant established by Theorem 3.3 below which allows instead of each parsing expression $e$ with negations in front of atoms and disjoint choices in $G$ to equivalently use parsing expression $\gamma_1(e)/\gamma_0(e)$ in $G'$. The claim is analogous to the splitting invariant used by [6] but we can provide a simpler proof using the strict semantics (an analogous proof using the standard semantics would fail in the choice case).

**Theorem 3.3.** Let $e, \tau \vdash_G s \rightarrow o$ where $e \in \mathcal{E}_G$, $\tau \in \mathbb{R}^+ (N)$ and $s \in \text{State}$, $o \in \text{State} + 1$. Assuming that all choices in the rules of $G$ and expression $e$ are disjoint and the negations are applied to atoms, the following holds:

1. If $o = \top(s)$ then $\gamma_0(e), \tau \vdash_{G'} s \rightarrow \top(s)$ and $\gamma_1(e), \tau \vdash_{G'} s \rightarrow \bot$;
2. If $o = \top(s')$ where $s' \neq s$ then $\gamma_0(e), \tau \vdash_{G'} s \rightarrow \bot$ and $\gamma_1(e), \tau \vdash_{G'} s \rightarrow \top(s')$;
3. If $o = \bot$ then $\gamma_0(e), \tau \vdash_{G'} s \rightarrow \bot$ and $\gamma_1(e), \tau \vdash_{G'} s \rightarrow \bot$.

**Proof.** We don’t use the repetition operator, whence all expressions in well-formed grammars are well-formed (this fact follows from an easy induction on the expression structure). By Theorems 3.2 and 2.1, $e, \tau \vdash_G s \rightarrow o$. The desired result follows by induction of the shape of the derivation tree of $e, \tau \vdash_G s \rightarrow o$, using the disjointness assumption in the choice case. □

As the result of this transformation, the sizes of the right-hand sides of the productions can grow exponentially though the number of productions 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. The subsequent transformations cause at most a linear growth of right-hand sides.

### 3.3 Alignment elimination

In a grammar $G = (N, T, \delta, s)$ obtained via splitting, we can eliminate alignments using the following three steps:

1. Introduce a copy $X'$ of each non-terminal $X$ and define $\delta(X') = |\delta(X)|$.
2. In all right-hand sides of productions and the start expression, apply distributivity laws (Theorem 2.3 (3), Theorem 2.4 (3), Theorem 2.5) and idempotence (Theorem 2.4 (4)) to bring all alignment operators down to terminals and non-terminals. Replace alignment of terminals by position (Theorem 2.4 (6)).
3. In all right-hand sides of productions and the start expression, replace all subexpressions of the form $\{X\}$ with the corresponding new non-terminal $X'$.

For establishing the equivalence of the original and the obtained grammar, the following general theorem can be used.

**Theorem 3.4.** Let $G_1 = (N, T, \delta_1, s_1)$ and $G_2 = (N, T, \delta_2, s_2)$ be PEG$^>$ s. If for every $X \in N$, $\delta_1(X) \sim_{G_1} \delta_2(X)$ then $e, \tau \vdash_{G_2} s \rightarrow o$ always implies $e, \tau \vdash_{G_1} s \rightarrow o$.

**Proof.** Easy induction on the shape of the derivation tree of $e, \tau \vdash_{G_2} s \rightarrow o$. □
Denote by $\varphi$ the function defined on $E_G$ that performs transformations of steps 2–3, i.e., distributes alignment operators to the non-terminals and replaces aligned non-terminals with corresponding new non-terminals. Denote by $G'$ the grammar obtained after step 3. Note that step 1 does not change the semantics of expressions written in the original grammar. Steps 2 and 3 replace the right-hand sides of productions with expressions that are semantically equivalent with them in the grammar obtained after step 1. By Theorem 3.4, this implies that whenever parsing of some $e \in E_G$ in the final grammar $G'$ produces some result then the same result is obtained when parsing $e$ with the same input state and token mode in the original grammar $G$. In order to be able to apply Theorem 3.4 with grammars interchanged, we need the equivalence of the right-hand sides of productions also in grammar $G'$. For this, it is sufficient to show $|X| \sim_{G'} |X'|$ for every $X \in N$, which in turn would follow from the statement $|\varphi(\delta(X))| \sim_{G'} |\varphi(\delta(X))|$. Consequently, the equivalence of the initial and final grammars is implied by the following theorem.

Theorem 3.5. For every $e \in E_G$, $|\varphi(e)| \equiv_{G'} |\varphi(e)|$.

Proof. The claim is a direct consequence of the following two lemmas, both holding for arbitrary $s \in \text{State}$, $o \in \text{State} + 1$:

1. If $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$ then $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$ and $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$;

2. If $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$ or $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$ then $|\varphi(e)|, \tau \vdash_{G'} s \rightarrow o$.

Both lemmas are proven by induction on the shape of derivation trees. The assertion with two alignments (both outside and inside) is needed in the case where $e$ itself is of the form $|p|$. ▶

3.4 Elimination of position operators

In an alignment-free PEG $G = (N, T, \delta, s)$, we can get rid of position operations using a process largely analogous to the alignment elimination, consisting of the following four steps:

1. Introduce a new non-terminal $(X, \tau)$ for each existing non-terminal $X$ and relation $\tau$ used by a position operator, with $\delta(X, \tau) = (\delta(X)\tau)$.

2. Apply distributivity laws (Theorem 2.3(1)) and cancellation (Theorem 2.4(5)) to bring all position operators down to terminals and non-terminals.

3. Replace all subexpressions of the form $X_\tau$ with corresponding new non-terminals $(X, \tau)$.

4. Replace all subexpressions of the form $a_\tau$ with $a^\tau$.

Again, denote by $\varphi$ the function defined on $E_G$ that performs transformations of steps 2–3, i.e., distributes position operators to the terminals and non-terminals and replaces non-terminals under position operators with corresponding new non-terminals. Denote by $G'$ the grammar obtained after step 3. Theorem 3.4 applies here as well, whence the equivalence of the grammar obtained after step 3 and the initial grammar is implied by the following theorem.

Theorem 3.6. For every $e \in E_G$ and $\sigma \in R^+(N)$, $|\varphi(e_\sigma)| \equiv_{G'} |\varphi(e_\sigma)|$.

Proof. The claim is a direct consequence of the following two lemmas, both holding for arbitrary $s \in \text{State}$, $o \in \text{State} + 1$ and $\tau, \nu \in R^+(N)$:

1. If $|\varphi(e_\sigma)|, \tau \vdash_{G'} s \rightarrow o$ then $(\varphi(e_\sigma)_\sigma, \tau \vdash_{G'} s \rightarrow o$ and $(\varphi(e_\sigma))_\nu, \tau \vdash_{G'} s \rightarrow o$;

2. If $(\varphi(e_\sigma)_\sigma, \tau \vdash_{G'} s \rightarrow o$ or $(\varphi(e_\sigma))_\nu, \tau \vdash_{G'} s \rightarrow o$ then $\varphi(e_\sigma), \tau \vdash_{G'} s \rightarrow o$.

Both lemmas are proven by induction on the shape of derivation trees. The claim with position operator both outside and inside ($(\varphi(e_\sigma))_\sigma$) is needed in the case when $e$ itself is an application of the position operator. ▶
Correctness of step 4 can be proven by induction on the shape of the derivation trees, using Theorem 2.6. Note that here we must assume that parsing according to the final grammar is performed with the alignment flag false (a natural assumption as the grammar is alignment-free) and the token mode $\triangle$.

3.5 Discussion on the preconditions

Alignment elimination was correctly defined under the assumption that the input grammar is well-formed, has negations only in front of atoms, and disjoint choices (all these conditions are needed at stage 1 only). The second assumption can be easily established by introducing a new non-terminal for each expression $p$ such that $!p$ occurs in the productions or in the start expression. This can be done in the lines of the first stage of the negation elimination process described by Ford [6]. This transformation preserves well-formedness of the grammar. Achieving disjoint choices is a more subtle topic. A straightforward way would be replacing choices of the form $p/q$ with disjoint choices $p!/pq$ which seems to work well as $p/q$ and $p!/pq$ are equivalent in the standard semantics.

Alas, $p/q$ and $p!/pq$ are not equivalent in the approximation semantics, because if $p \rightarrow_G 1$, $p \rightarrow_G -1$, $q \rightarrow_G 0$ but $q \not\rightarrow_G -1$, then $p!/pq \not\rightarrow_G -1$ while $p/q \not\rightarrow_G -1$. Due to this, replacing $p/q$ with $p!/pq$ can break well-formedness. Take $X \in N$ such that $\delta(X) = !(a/\varepsilon)X$. Then $X \in WF_G$ due to $!(a/\varepsilon) \in WF_G$ alone, no recursive call to $X \in WF_G$ arises as $!(a/\varepsilon) \not\rightarrow_G 0$. However, if $\delta'(X) = !(a/3a\varepsilon)X$ in $G'$ then $!(a/3a\varepsilon) \rightarrow_{G'} 0$ whence well-formedness of $X$ now recursively requires well-formedness of $X$. Thus $X \notin WF_{G'}$. (An argument similar to this shows that the first stage of the negation elimination process in Ford [5] also can break well-formedness. As the second stage is correctly defined only for well-formed grammars, the whole process fails.)

One solution would be changing the approximation semantics by adding, to the inductive definition in Subsect. 3.1, a general clause

0. $e \rightarrow_G -1$ if $e \rightarrow_G 0$ or $e \rightarrow_G 1$.

This forces $e \rightarrow_G -1$ to hold whenever an assertion of the form $e \rightarrow_G n$ holds, and in particular, $p/q$ becomes equivalent to $p!/pq$. Then replacing $p/q$ with $p!/pq$ preserves well-formedness. Although well-formedness predicate becomes more restrictive and rejects more safe grammars, the loss seems to be little and acceptable in practice (expressions $q$ such that $q \rightarrow_G 0$ or $q \rightarrow_G 1$ while $q \not\rightarrow_G -1$ seem to occur not very commonly in influenced productions such as $X \mapsto !(p/q)X$, but a further investigation is needed to clarify this).

4 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 [3], as well as to their generalized forms [3, 4].

Adams [2] and Adams and Aşçan [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.

5 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.

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. URL: http://doi.acm.org/10.1145/2429069.2429129 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. URL: http://doi.acm.org/10.1145/2633357.2633369 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. URL: http://dx.doi.org/10.1016/S0019-9958(73)90851-6 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. URL: http://doi.acm.org/10.1145/964001.964011 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. URL: http://dx.doi.org/10.2197/ipsjjip.24.256 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. URL: http://dx.doi.org/10.1007/978-3-642-33182-4_4 doi:10.1007/978-3-642-33182-4_4