Recursive Descent Parsing for Grammars with Contexts

Mikhail Barash\textsuperscript{1,2}

\textsuperscript{1} Turku Centre for Computer Science TUCS, Turku FI-20520, Finland, \textsuperscript{2} Department of Mathematics and Statistics, University of Turku, Turku FI-20014, Finland, mikhail.barash@utu.fi

Abstract. This paper undertakes to reconsider recursive descent parsing technique by using a general mathematical definition of contexts within grammar rules. This definition takes form of grammars with contexts (Barash, Okhotin, 2012). The model extends context-free grammars with quantifiers for referring to the context in which a substring being defined occurs. It has been shown that general parsing of such grammars is cubic-time, while their unambiguous subclass allows square-time parsing. The paper generalizes the recursive descent parsing to grammars with contexts. Right contexts, which can be explicitly specified, are used to choose the alternatives suitable during the parsing. The applicability of the proposed algorithm is restricted to a subset of grammars with contexts satisfying the properties of separability (Wood, 1971) and monotone decreasing prefixes. The latter allows proving the correctness of backtracking in this algorithm, which is defined similarly to Birman and Ullman (1973) and Ford (2004). A memoized version of the algorithm is guaranteed to have linear time complexity.

1 Introduction

Recursive descent parsing is probably the most well-known and intuitive technique applicable to a subclass of context-free grammars. A subroutine for each nonterminal should determine a rule according to which the substring shall be parsed. If such a rule can be determined using \( k \) next symbols of the input, then this is a standard LL(\( k \)) parser [1,13,14,15,28,29].

If several suitable alternatives exist, the issue of choosing one of them arises. Some of software implementations of recursive descent parser generators, such as ANTLR by Parr and Fisher [23,24], allow using ad hoc methods to make the parsing deterministic when it is inherently not. For instance, a unique rule according to which the string shall be parsed, may be chosen by scanning a right context of a string without actual consuming of the input. Another example of a mathematical model of making a choice of rule deterministic is LL-regular grammars [12], where a form of a right context can be specified by a regular language.

Unlike the deterministic predictive parsing, limited backtracking parsing uses prioritized choice of alternatives, going consequently over the set of suitable
rules. The parsing is successful when a string is parsed according to at least one of them. Birman and Ullman introduced the recognition schemata \(^4\) of top-down parsing languages (“TS/TDPL”) and generalized top-down parsing languages (“gTS/GTDPL”), which are capable of specifying recognizers for deterministic context-free languages with restricted backtracking. One of the software implementations of such approach is parsing expression grammars \(^7\), which additionally has a means to express Boolean operations over the right contexts of a string. The effective recognition power of these grammars has been proved \(^7\) equivalent to the one of the mentioned schemata.

Recently, a mathematical definition of contexts within grammar rules, called grammars with contexts \(^2\), has been introduced. The model extends context-free grammars with quantifiers for referring to the context (either left or right), in which a substring being defined occurs. For example, a rule \(A \rightarrow a \& \triangledown B\) defines a string \(a\) if it is followed by a string of the form \(B\). The conjunction operation is taken from conjunctive grammars \(^{17}\), which are an extension of standard context-free grammars with Boolean operation of intersection, where a rule, say, \(A \rightarrow B \& C\), assumes that a string has the property \(A\) if it has both properties \(B\) and \(C\).

The definition of grammars with contexts, their semantics and examples are given in Section \(^2\). Formal properties of grammars with contexts, including existence of a normal form, parsing algorithm working in cubic time, square-time algorithm for unambiguous subclass, and membership of languages generated by such grammars in \(\text{DSPACE}(n)\), are established in the literature \(^{2,3}\).

In this paper, the recursive descent parsing is extended to handle the case of grammars with contexts, additionally allowing a limited backtracking. Conjunction operation is implemented by scanning a single substring multiple times, as in the case of conjunctive grammars \(^{21}\). Possible rules are tested one by one in a given order; the applicability of the rule is determined by checking the contexts specifications. Thus, right contexts are used to choose the alternatives, suitable during the parsing.

Applicability of the algorithm is restricted to a subset of grammar with contexts which satisfy the properties of separability \(^{27,29}\) and monotone decreasing prefixes, defined in Section \(^3\). The latter allows proving the correctness of the algorithm with limited backtracking even in such undesirable situations \(^{13}\) as one occurring in the grammar with the rules \(S \rightarrow Ac\) and \(A \rightarrow a \mid ab\). Given such a grammar and an input string \(abc\), the parser would fail when scanning symbol \(b\), since after having consumed the prefix \(a\) and returned according to the first alternative for \(A\), it would expect the symbol \(c\).

The recursive descent parsing algorithm for grammars with right contexts is described in Section \(^3\) and its correctness is proved. A direct implementation of the algorithm may use exponential time on some extreme grammars. However, using the memoization technique guarantees the linear time complexity of the parser.
2 Definitions

Definition 1 ([2]). A grammar with right contexts is a quadruple $G = (\Sigma, N, R, S)$, where
- $\Sigma$ is the alphabet of the language being defined;
- $N$ is a finite set of auxiliary symbols, disjoint with $\Sigma$, which denote the strings defined in the grammar;
- $R$ is a linearly ordered finite set of rules, each of the form
  \[ A \rightarrow Q_1\alpha_1 \& \ldots \& Q_n\alpha_n \]
  (1)
  with $A \in N$, $n \geq 1$, $\alpha_i \in (\Sigma \cup N)^*$, $Q_i \in \{\Box, \Box, \trianglerighteq\}$, $\{i \in \{1, \ldots, n\} \mid Q_i = \Box\} \neq \emptyset$ and $Q_i = \trianglerighteq$ implies that for some $j < i$ it holds that $Q_j = \Box$;
- $S \in N$ is a symbol representing correct sentences (“start symbol”).

For every grammar rule (1), a term $Q_i\alpha_i$ is called a conjunct. Each conjunct $\alpha_i$ with $Q_i = \Box$ (denoted as empty quantifier) gives a representation of the substring being defined. A conjunct $\trianglerighteq\alpha_i$ describes the form of the right context or the future of the substring. Conjuncts $\trianglerighteq\alpha_i$ refer to the form of the current substring and its right context, concatenated into a single string. A linear order is assumed on the set of conjuncts within each rule.

Definition [1] implies that every rule in the grammar should have at least one $\Box$-conjunct. Moreover, every $\trianglerighteq$-conjunct should be preceded by at least one $\Box$-conjunct, that is, the right context of a string can only be specified after the string itself has been specified. On the contrary, the same restriction is not needed for $\trianglerighteq$-conjuncts, since they describe the form of both the string and its future. Those conjuncts are extensively used to define the form of lookahead for the recursive descent parser.

The semantics of grammars with contexts can be defined in two equivalent ways [2]. One of them is by logical deduction of elementary propositions (items) of the form “a string $w \in \Sigma^*$ in the right context $v \in \Sigma^*$ has the property $\alpha \in (\Sigma \cup N)^*$”, denoted as $[\alpha, (w)v]$.

Definition 2 ([2]). Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Define the following system of items of the form $[X, (w)v]$, with $X \in \Sigma \cup N$ and $v, w \in \Sigma^*$ as follows. There is a single axiom scheme: $1 \vdash_G [a, (a)y]$, for all $a \in \Sigma$ and $y \in \Sigma^*$. Each rule $A \rightarrow Q_1\alpha_1 \& \ldots \& Q_n\alpha_n$ in the grammar defines a scheme $I \vdash_G [A, (w)v]$ for deduction rules, for all $v, w \in \Sigma^*$ and for every set of items $I$ satisfying the below properties:

- for every conjunct $Q_i\alpha_i$ with $Q_i = \Box$, $\alpha_i = X_1 \ldots X_\ell$, $\ell \geq 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $w = w_1 \ldots w_\ell$ with $[X_j, (w)jw_{j+1} \ldots w_\ell] \in I$ for all $j \in \{1, \ldots, \ell\}$;
- for every conjunct $Q_i\alpha_i$ with $Q_i = \trianglerighteq$, $\alpha_i = X_1 \ldots X_\ell$, $\ell \geq 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $v = v_1 \ldots v_\ell$, with $[X_j, (v)jv_{j+1} \ldots v_\ell] \in I$ for all $j \in \{1, \ldots, \ell\}$;
for every conjunct $Q_i\alpha_i$ with $Q_i = \triangleright$, $\alpha_i = X_1 \ldots X_\ell$, $\ell \geq 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $wv = x_1 \ldots x_\ell$ with $[X_j, (x_j)x_{j+1} \ldots x_\ell] \in I$ for all $j \in \{1, \ldots, \ell\}$. 

The language generated by a nonterminal symbol $A$ is defined as $L_G(A) = \{ \langle w\rangle v \mid v, w \in \Sigma^*, \rightharpoonup_G [A, \langle w\rangle v] \}$. 

The language generated by the grammar $G$ is the set of all strings with right context $\varepsilon$ generated by $S$: $L(G) = \{ w \mid w \in \Sigma^*, \rightharpoonup_G [S, \langle w\rangle \varepsilon] \}$. 

Note, that $\alpha_0 = \varepsilon$ in the first case implies $w = \varepsilon$. Similarly, $\alpha_i = \varepsilon$ in the second and third cases imply $v = \varepsilon$ and $w = v = \varepsilon$, respectively. 

Another definition of the semantics of grammars with contexts can be given by language equations over the languages of pairs [2]. 

Grammars with contexts can define languages which are not context-free [2]. Examples include the language $\{ a^n b^n c^n d^n \mid n \geq 0 \}$ and an abstract language representing declaration of identifiers before or after their use, for which no Boolean grammar (and hence no conjunctive grammar) is known. 

**Example 1.** Consider a language $\{ a^n b^n \mid n \geq 1 \} \cup \{ a^n d^n b^n \mid n \geq 1 \}$, which is not an LL($k$) language for any $k$ [1, Ex. 5.1.20]. 

This language can be generated by the following grammar with right contexts:

$$S \rightarrow \triangleright Ax & C | \triangleright Ax & A \\
A \rightarrow aA | \varepsilon \\
X \rightarrow aX | bX | cX | dX | \varepsilon \\
C \rightarrow aCb | c \\
D \rightarrow aDbb | d$$

Nonterminals $C$ and $D$ generate languages $a^n b^n$ (with $n \geq 1$) and $a^n d^n b^n$ (with $n \geq 1$) in any right context, respectively. Both rules for $S$ use right context specification of the form $a^n x \Sigma^*$ with $x = c$ or $x = d$. Depending on the symbol $x$, one of the nonterminals $C$ or $D$ is chosen to match the string. 

**Example 2.** The following grammar with right contexts generates the language $\{ a^n b^n c \mid n \geq 1 \} \cup \{ a^n b^n d \mid n \geq 0 \}$, which is not deterministic context-free:

$$S \rightarrow \triangleright Xc & A & C | \triangleright Xd & B & d \\
A \rightarrow aAb | \varepsilon \\
X \rightarrow aX | bX | \varepsilon \\
B \rightarrow aBbb | \varepsilon$$

Languages given by the symbols $A$ and $B$ are $\{ \langle a^n b^n \rangle y \mid n \geq 0, y \in \Sigma^* \}$ and $\{ \langle a^n b^n \rangle y \mid n \geq 0, y \in \Sigma^* \}$, respectively. The context quantifiers in the rules for $S$ check the very last symbol of the string, which can be either $c$ or $d$. 

## 3 Backtracking LL($k$) Grammars with Contexts

The computation of a recursive descent parser is guided by a parsing table, which, for every $A \in N$ and $w \in \Sigma^*$, determines the rules to apply when $A$ is to produce a string starting with $w$. A conjunctive grammar is called LL($k$) [21,22], if there is a deterministic LL($k$) table for it. Using the same definition for grammars with
contexts is inherently not possible, since the parsing table is nondeterministic and always exists. Thus, some additional conditions have to be imposed on the grammar.

Consider the grammars from Examples 1 and 2. In the first of them, the contexts in the rules for \( S \) involve \( \Sigma^* \) at the very end of the string, while in the second one a similar check is made in the beginning. Taking in the latter case \( X \) same as in Example 1, which does not change the language generated by the second grammar, would make it impossible to distinguish between the end of the string generated by \( \{a, b, c, d\}^* \) and the string \( c \) (\( d \), respectively). Such intuitive notion of a “border” between two strings shall be now formalized.

**Definition 3** (27,29). Let \( K, L \subseteq \Sigma^* \times \Sigma^* \). A pair \( (K, L) \) is said to be separable, if \( K^{-1} \cdot K \setminus \{ (\epsilon)_y \mid y \in \Sigma^* \} \cap L \cdot L^{-1} \setminus \{ (\epsilon)_y \mid y \in \Sigma^* \} = \emptyset \), where \( X^{-1} \cdot Y = \{ (w_2)v_2 \mid (w_1)v_2 \in X, (w_1)w_2v_2 \in Y, w_1, w_2, v_1, v_2 \in \Sigma^* \} \) and \( X \cdot Y^{-1} = \{ (w_1)v_1 \mid (w_1)w_2v_2 \in X, (w_2)v_2 \in Y, w_1, w_2, v_1, v_2 \in \Sigma^* \} \), for \( X, Y \subseteq \Sigma^* \times \Sigma^* \), are left (right, respectively) quotients of \( X \) by \( Y \).

**Example 3.** Consider the following grammar generating the language \{ \( a^m a^n b^n \mid m, n \geq 0 \) \}, which is not \( LL(k) \) context-free for any \( k \geq 0 \):

\[
S \rightarrow AB \quad A \rightarrow aA \mid \epsilon \quad B \rightarrow aBb \mid \epsilon
\]

The language generated by nonterminal \( A \) is \( a^* \) (in any right context), while nonterminal \( B \) generates the language \( a^n b^n, \) with \( n \geq 0 \) (in any right context).

The languages \{ \( a^n y \mid n \geq 1, y \in \Sigma^* \) \} and \{ \( a^{n-1}b^n \mid n \geq 1 \) \} are not separable.

**Example 4.** Consider the following grammar with right contexts:

\[
S \rightarrow X & C \quad X \rightarrow aX \mid bX \mid \epsilon \quad C \rightarrow A & \epsilon \quad A \rightarrow aA \mid \epsilon
\]

Clearly, \( A \) defines \( a^* \) in any right context. The context specification in the rule for nonterminal \( C \) assumes that \( a^* \) should be followed by the string \( ab \). As in the previous example, the languages \{ \( a^n y \mid n \geq 1, y \in \Sigma^* \) \} and \{ \( ab \epsilon \) \} are not separable.

Backtracking in a recursive descent parser has to impose some restrictions on how the rules for a same nonterminal are ordered in a grammar.

**Example 5.** Consider a grammar with the rules \( S \rightarrow Ac \) and \( A \rightarrow a \mid ab \), demonstrating that the order in which the rules appear in a grammar is dramatically important for a recursive descent parser with backtracking \[13,23\]. When parsing an input string \( abc \), the parser would fail scanning symbol \( b \), since after having consumed the prefix \( a \) and returned according to the first alternative for \( A \), it would expect the symbol \( c \).

The following condition allows the parser to avoid situations as one described.
Definition 4. Let \( G = (\Sigma, N, R, S) \) be a grammar with right contexts, and let \( A \) be a nonterminal symbol such that \( A \rightarrow \varphi_1, \ldots, A \rightarrow \varphi_m \) (with \( \varphi_i = Q_i \alpha_{i,1} \& \ldots \& Q_i \alpha_{i,n} \)) are all rules for \( A \). Then \( A \) is said to have a monotone decreasing prefixes property if for all \( i, j \in \{1, \ldots, m\} \) such that \( i < j \), there do not exist strings \( \langle w_1 \rangle v_1 \in L_G(\varphi_i) \) and \( \langle w_2 \rangle v_2 \in L_G(\varphi_j) \) such that \( w_1 \in \text{Pref}(w_2v_2) \), where \( \text{Pref}(u) \), for a string \( u \in \Sigma^* \), denotes the set of all its prefixes.

Context-free recursive descent for standard context-free grammars, as well as its generalizations for conjunctive and Boolean grammars [21], require the grammar to be free of left recursion, which means that no nonterminal \( A \) can derive \( A \xi \) for any \( \xi \in (\Sigma \cup N)^* \). The reason for that is that a parser can enter an infinite loop otherwise. Non-left-recursive grammars with contexts can be defined similarly to the case of conjunctive grammars [21], using the relation of the context-free reachability in one step, which is an adaption of context-free derivation [1] to specify dependence of nonterminals in a grammar.

Definition 5 ([21]). Let \( G = (\Sigma, N, R, S) \) be a grammar with right contexts. Define the relation of context-free reachability in one step, \( \rightsquigarrow \), a binary relation on the set of strings with a marked substring \( \{ \alpha[\beta\gamma] \mid \alpha, \beta, \gamma \in (\Sigma \cup N)^* \} \) as: \( \alpha[\beta A\gamma] \mu \rightsquigarrow \alpha\beta[\sigma]\gamma\mu \), for all \( \alpha, \beta, \gamma, \mu, \xi, \sigma, \eta \in (\Sigma \cup N)^* \), \( A \in N \), and for all conjuncts \( \xi \sigma \eta \) in the rules for the nonterminal \( A \).

Now the definition of a backtracking LL(\( k \)) grammar with contexts can be given.

Definition 6. A non-left-recursive grammar with contexts \( G = (\Sigma, N, R, S) \) is said to be backtracking LL(\( k \)) ("BLL(\( k \))" in short) if the following conditions hold:

I. each nonterminal has the property of monotone decreasing prefixes;
II. a pair \( (L_G(X), L_G(Y)) \) is separable for all \( X, Y \in \Sigma \cup N \) such that \( \varepsilon[\delta]\varepsilon \rightsquigarrow^* \xi[X]Y\eta \), where \( \delta = S \) or \( \delta = \alpha \), \( \alpha \in (\Sigma \cup N)^* \), \( Q\alpha \) (\( Q \in \{\trianglerighteq, \triangleright\} \)) is a quantified conjunct in a rule for some nonterminal in the grammar.

The grammars in Examples 1 and 2 are BLL(\( k \)) for every \( k \geq 0 \), unlike the grammars in Examples 3 and 5 which do not satisfy Definition 6. Note, that all the languages generated by these grammars are not context-free LL(\( k \)) for any \( k \).

In general, checking whether an arbitrary grammar with contexts satisfies Definition 6 is undecidable. This is caused by the undecidability of the property of separability, which, in its turn, follows from the undecidability of emptiness of intersection [1].

Before a parsing table can be formally defined, the following technical notions are needed.

Definition 7. Let \( v \in \Sigma^* \) be a string, and let \( k \geq 1 \). Define

\[
\text{First}_k(w) = \begin{cases} w, & \text{if } |w| \leq k \\ \text{first } k \text{ symbols of } w, & \text{if } |w| > k \end{cases}
\]
Definition 8. Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. A string $v \in \Sigma^*$ is said to follow $\sigma \in (\Sigma \cup N)^*$ if $\varepsilon[\delta]\varepsilon \twoheadrightarrow^* \xi[\sigma]\eta$ where $\delta = S$ or $\delta = \alpha$, $\alpha \in (\Sigma \cup N)^*$, $Q\alpha$ ($Q \in \{\trianglerighteq, \trianglerighteqslant\}$) is a quantified conjunct in a rule for some nonterminal in the grammar, with $\xi, \eta \in (\Sigma \cup N)^*$, and $\langle v \rangle y \in L_G(\eta)$.

Now the definition of the parsing table can be given.

Definition 9. Let $G = (\Sigma, N, R, S)$ be a BLL($k$) grammar with right contexts, and let $k \geq 0$. A nondeterministic BLL($k$) table for $G$ is a function $T_k : N \times \Sigma^k \rightarrow 2^R$, such that for every rule $A \rightarrow \varphi$, and $u, v \in \Sigma^*$, for which $\langle u \rangle vy \in L_G(\varphi)$ (with $y \in \Sigma^*$) and $v$ follows $A$, it holds that $A \rightarrow \varphi \in T_k[A, \text{First}_k(uv)]$.

Note, that the definition allows $k$ to be zero. In that case, the function $T_k$ maps the set of nonterminals to the powerset of the rules: $T_k : N \rightarrow 2^R$. This means, that whenever more than one rule is given for some nonterminal, all of them should be tried (until the first of them, matching the substring, is found) without using any lookahead symbols.

The construction of the parsing table is similar to the case of conjunctive grammars [21] and is based on the variants of the sets First$_k(A)$ and Follow$_k(A)$ [1], which can be computed by the appropriate algorithms.

Example 6. Consider the grammar from Example 1. The BLL(1) parsing table for it is as follows:

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$\emptyset$</td>
<td>${S \rightarrow \trianglerighteq AcX &amp; C, S \rightarrow \trianglerighteq AdX &amp; D}$</td>
<td>$\emptyset$</td>
<td>$S \rightarrow \trianglerighteq AcX &amp; C$</td>
<td>$S \rightarrow \trianglerighteq AdX &amp; D$</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \rightarrow \varepsilon$</td>
<td>$A \rightarrow aA$</td>
<td>$\emptyset$</td>
<td>$A \rightarrow \varepsilon$</td>
<td>$A \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$X$</td>
<td>$X \rightarrow \varepsilon$</td>
<td>$X \rightarrow aX$</td>
<td>$X \rightarrow bX$</td>
<td>$X \rightarrow cX$</td>
<td>$X \rightarrow dX$</td>
</tr>
<tr>
<td>$C$</td>
<td>$\emptyset$</td>
<td>$C \rightarrow aCb$</td>
<td>$\emptyset$</td>
<td>$C \rightarrow c$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$D$</td>
<td>$\emptyset$</td>
<td>$D \rightarrow aDb$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$D \rightarrow d$</td>
</tr>
</tbody>
</table>

The table is nondeterministic, as the cell $(S, a)$ contains two rules. Thus, both of the rules shall be tried by the parsing algorithm in the given order.

4 Recursive Descent Parser for BLL($k$) Grammars with Contexts

Recursive descent parser enables to perform the analysis of an input string, using the collection of recursive subroutines constructed upon the form of the rules of the grammar. For each symbol of the grammar, the recursive descent parser contains a subroutine, which, having an access to the input string and the current position of the parser, tries to recognize this symbol and advance the current position, until the whole input string is consumed.

In the case of context-free grammars, for which recursive descent parsing is possible, namely LL($k$) grammars [14,15,16,26], the code of each procedure $a()$,
with $a \in \Sigma$, checks whether the next symbol in the input string is $a$, and, if so, advances the current position of the parser. Otherwise, an error is reported.

For each nonterminal symbol $A \in N$, the corresponding procedure $A()$ shall deterministically choose one of the rules with $A$ on their left-hand side, only using $k$ look-ahead symbols of the input. Once a rule $A \rightarrow X_1 \ldots X_n$ is chosen, the code $X_1(); \ldots ;X_n();$ parses the following substring according to this rule, i.e. by invoking corresponding procedures for each of the symbols $X_i, i \in \{1, \ldots, n\}$.

In the extension of recursive descent parsing to conjunctive grammars [21], the conjunction operation is implemented by scanning a single substring multiple times. If a parser chooses a rule, say, $A \rightarrow B \& C$, it first stores its current (initial) position in a local variable, and then invokes the procedure $B()$. So far, the substring is parsed according to the first conjunct. Then the parser again stores its (final) position in another local variable, thus remembering the position it gained after the first parse of the substring. Next, the current position of the parser is rewound back to the stored initial position and parsing of the substring according to the second conjunct is performed by invoking the procedure $C()$. Once it returns, the parser checks whether its current position is identical to what it had after the first parse of the string. If the two positions are identical, the procedure $A()$ returns; if not, an error is reported.

Generalization of the recursive descent parser method to the case of grammars with right contexts, additionally to implementation of the conjunction operation, shall check that the $\trianglerighteq$-and $\triangleright$-contexts are satisfied. If a parser chooses a rule $A \rightarrow B \& C \& F \trianglerighteq H$, then initially it behaves exactly the same as the parser for conjunctive grammars. After the two successful parses of the substring (according to $B$ and $C$), the parser stores its current position in a local variable and invokes the procedure $F()$. If the position after the procedure $F$ returns is identical to the length of the string, the context $\trianglerighteq F$ is deemed satisfied and the parser continues with the context $\trianglerighteq H$. If not, an error is reported. When checking the context $\trianglerighteq H$, the parser rewinds its position to the beginning of the substring and invokes the procedure $H()$. Similarly to the case of the $\trianglerighteq F$ context, either an error is reported or the parser continues its work. In the latter case, since no other contexts are left to check, the parser reports a successful parsing of the substring. Since the conjuncts in each rule are linearly ordered, they are parsed in the order they are given. This allows one to check first the $\trianglerighteq$-conjuncts, which represent a form of a lookahead. If some $\trianglerighteq$-conjunct of a rule is not satisfied, then the whole rule is considered mismatching and the actual parsing of the substring shall not be done.

When the parser can not deterministically choose the rule to apply, it shall use the backtracking technique, which is fairly standard in recursive descent parsing [4][13][25]. Thus, if several rules $r_1, \ldots, r_n$ for a nonterminal symbol $A$ are suitable, the parser first tries to parse the substring according to the first rule, $r_1$, having stored its (initial) position in a local variable beforehand. If no error has been reported, the substring is deemed recognized according to the rule $r_1$, and the parser moves on to the next symbol to parse. Otherwise, the parser shall rewind its current position to the stored one, and try the next rule,
r_2. The parser continues working in such a manner until either after some rule r_i no error is reported (then the substring is recognized according to this rule), or all possible rules for nonterminal A are exhausted. In the latter case, an error is reported. Such behaviour of the parser can be implemented with the mechanism of structured exception handling [9].

Backtracking recursive descent parser with k look-ahead symbols is defined below for grammars with right contexts. It shall use two global variables accessible to all subroutines: the input string w and the current position p of the parser within that string (a positive integer).

Let G = (Σ, N, R, S) be a grammar with contexts, and let T_k be a BLL(k) table. The subroutine corresponding to every terminal a ∈ Σ is defined as follows:

\[
a() \begin{cases} 
  \text{if } w_p = a \text{ then} \\
  p := p + 1; \\
  \text{else} \\
  \quad \text{raise;}
\end{cases}
\]

For every nonterminal A ∈ N the code is

\[
A() \begin{cases} 
  \text{let } R_A = T_k[A, First_k(w_pw_{p+1} \ldots)] \\
  \quad \text{boolean failed := false;} \\
  \quad \text{integer pos = p; (omit this if } n = 1) \\
  \quad \text{try} \\
  \quad \quad \text{(code for the first rule in } R_A) \\
  \quad \text{catch} \\
  \quad \quad \text{try} \\
  \quad \quad \quad \text{p := pos;} \\
  \quad \quad \quad \text{(code for the next rule in } R_A) \\
  \quad \quad \quad \quad \text{try} \\
  \quad \quad \quad \quad \quad \text{p := pos;} \\
  \quad \quad \quad \quad \quad \text{(code for the last rule in } R_A) \\
  \quad \quad \quad \text{catch} \quad \text{failed := true;} \\
  \quad \quad \text{.;} \\
  \quad \text{if } \text{failed then raise;}
\end{cases}
\]

The code for a rule of the form A → Q_1α_1 & … & Q_nα_n consists of code fragments for each of the conjuncts. The code for a conjunct Q_iα_i (with α_i = s_1…s_ℓ), depending on the quantifier, is defined as follows:
– if \( Q_i = \triangleright \), then the code is
\[
\text{start} := p; \quad (\text{if this is the very first conjunct in the rule})
\]
\[
p := \text{start}; \quad (\text{otherwise})
\]
\[
s_1();
\]
\[
\ldots
\]
\[
s_\ell();
\]
\[
\text{if } p \neq |w| \text{ then raise;}
\]

– if \( Q_i = \boxtimes \), then the code is
\[
\text{start} := p; \quad (\text{if this is first } \boxtimes \text{-conjunct in the rule})
\]
\[
p := \text{start}; \quad (\text{otherwise})
\]
\[
s_1();
\]
\[
\ldots
\]
\[
s_\ell();
\]
\[
\text{end} := p; \quad (\text{if this is first } \boxtimes \text{-conjunct in the rule})
\]
\[
\text{if } p \neq \text{end then raise; (otherwise)}
\]

– if \( Q_i = \triangleright \), then the code is
\[
p := \text{end};
\]
\[
s_1();
\]
\[
\ldots
\]
\[
s_\ell();
\]
\[
\text{if } p \neq |w| \text{ then raise;}
\]

The procedure \( S() \) is called in the main procedure of the parser.

The algorithm in the form it is presented, has exponential time complexity, which is caused by recomputing the same procedures with the same value of the pointer in different branches of the computation. The technique of memoization \cite{7,21,25}, that is, storing the result of the first computation in memory and then looking it up for every subsequent call the same procedure with the same arguments, can be used to avoid such behaviour of the parser. Such a modification to a parser reduces its complexity to linear.

Now the correctness of the algorithm shall be stated. It should be shown that the algorithm always terminates and accepts a string if and only if this string is in the language.

**Lemma 1.** Let \( G = (\Sigma, N, R, S) \) be BLL\((k)\) grammar with right contexts. Let \( k \geq 0 \). Let \( T : N \times \Sigma^{\leq k} \rightarrow 2^R \) be a BLL\((k)\) table for \( G \), and let the set of procedures be constructed with respect to \( G \) and \( T \). Let \( w, v \in \Sigma^* \) and \( s_1, \ldots, s_\ell \in \Sigma \cup N \) (with \( \ell \geq 0 \)), and assume that there exists \( \hat{v} \in \Sigma^* \), such that \( \hat{v} \) follows \( s_1 \ldots s_\ell \) and \( \text{First}_k(\hat{v}) = \text{First}_k(v) \). Then the code \( s_1(); \ldots s_\ell() \) returns on the input \( wv \), consuming \( w \), if and only if \( \langle w \rangle v \in L_G(s_1 \ldots s_\ell) \).

Taking \( v = \hat{v} = \varepsilon, \ell = 1 \) and \( s_1 = S \) in Lemma 1 gives the following theorem.

**Theorem 1.** Let \( G = (\Sigma, N, R, S) \) be a BLL\((k)\) grammar with contexts. Let \( k \geq 0 \). Let \( T : N \times \Sigma^{\leq k} \rightarrow 2^R \) be the corresponding BLL\((k)\) parsing table, and let the collection of procedures of the parser be constructed with respect to \( G \) and \( T \). Then, if \( w \in L(G) \), for every string \( w \in \Sigma^* \), the procedure \( S() \) executed on \( w \) returns, consuming the whole input. If \( w \notin L(G) \), it either returns, consuming a proper prefix of the input, or raises an exception.

The proofs are omitted due to the space constraints.
Conclusion

Yet another model of recursive descent parsing technique has been proposed. Unlike some other extensions of the recursive descent, used in software implementations, the proposed approach has a theoretical basis represented by grammars with right contexts, for which a sound semantics has been defined. In spite of the fact that grammars with contexts have higher expressive power than context-free grammars, the complexity of the algorithm remains linear. Thus, it seems worth trying to implement the proposed technique in a practical software.

A prototype implementation of a parser generator for grammars with contexts, which further allows regular expressions in the right-hand sides of the rules, is available at the author’s homepage (http://users.utu.fi/mikbar/gwrc/). The expressive means of the grammars are experimentally augmented with the negation operation.

The expressive power of BLL($k$) grammars with contexts remains uninvestigated so far. For example, conjunctive LL($k$) and Boolean LL($k$) grammars have been proved to generate only regular languages over a one-letter alphabet. Obtaining a similar result for BLL($k$) grammars with contexts, as well as some other negative results, is a possible direction of further research on the model.

Acknowledgments

The author expresses his deep gratitude to Dr. Alexander Okhotin for valuable suggestions and numerous discussions about the paper. The author is thankful to the anonymous referees for a number of valuable comments.

References