Fuzzy Context-Free Languages — Part 2: Recognition and Parsing Algorithms

Peter R.J. Asveld

Department of Computer Science, Twente University of Technology
P.O. Box 217, 7500 AE Enschede, the Netherlands

Abstract

In a companion paper [9] we used fuzzy context-free grammars in order to model grammatical errors resulting in erroneous inputs for robust recognizing and parsing algorithms for fuzzy context-free languages. In particular, this approach enables us to distinguish between small errors ("tiny mistakes") and big errors ("capital blunders").

In this paper we present some algorithms to recognize fuzzy context-free languages: particularly, a modification of Cocke–Younger–Kasami's algorithm and some recursive descent algorithms. Then we extend these recognition algorithms to corresponding parsing algorithms for fuzzy context-free languages. These parsing algorithms happen to be robust in some very elementary sense.

Key words: formal language, fuzzy context-free grammar, grammatical error, recognition algorithm, parsing algorithm, robust parsing.

1 Introduction

In a companion paper [9] we proposed a way to deal with correct as well as erroneous inputs to a recognizer or parser for context-free languages. The concept of fuzzy context-free grammar, as introduced in [18], provides an appropriate framework to model such a situation. In [9] these grammars have been generalized to the concept of fuzzy context-free K-grammars, i.e., a fuzzy context-free grammar with a countable rather than a finite number of grammar rules. This concept enables us to model the fact that, in general, there is an infinite number of ways to apply a rule erroneously; see [9] for more discussion, precise definitions and examples.

However, the main result of [9] implies that, in order to stay within the family of fuzzy context-free languages, the parameter K should have any value

Email address: infprja@cs.utwente.nl (Peter R.J. Asveld).

in between the family FIN_f of finite fuzzy languages and the family CF_f of fuzzy context-free languages. Thus, to keep matters as simple as possible, in the present paper we restrict ourselves to the case $K = \mathrm{FIN}_f$, i.e., to $\mathcal{L}$-fuzzy context-free grammars, where $\mathcal{L}$ is any type-00 lattice [9].

In this paper we address the problem of recognizing and parsing fuzzy context-free languages. So given a fuzzy context-free language L_0 over an alphabet Σ, generated by a fuzzy context-free grammar $G = (V, \Sigma, P, S)$, we will construct a recognizer or parser M for L_0. Of course, M is based on G, because G is the only finite description available for the countable set L_0. The set of productions P of G consists of: (i) rules p that fully belong to P, i.e., their degree of membership $\mu_P(p)$ —which we will also write as $\mu(p; P)$— satisfies $\mu(p; P) = 1$, and (ii) rules p' that are added to P to model grammatical errors: $0 < \mu(p'; P) < 1$. Applying rules of type (i) only, results in terminal strings x that fully belong to $L(G)$, i.e., $\mu(x; L(G)) = \mu(x; L_0) = 1$; so x belongs to the crisp part $c(L_0)$ of L_0. On the other hand making one or more grammatical errors, viz. applying type (ii) rules, results in a terminal string x' with $0 < \mu(x'; L(G)) < 1$, i.e., an erroneous input to the recognizer or parser M.

As argued in [9], the least we require of M is that M is able to compute the characteristic function $\mu_{c(L_0)} : \Sigma^\star \to \{0, 1\}$ of the crisp part $c(L_0)$ of the fuzzy language L_0. But the fuzzy set L_0 contains more than the contents of its crisp part $c(L_0)$, e.g., "tiny mistakes" (i.e., strings x over Σ with $\mu(x; L_0)$ unequal to, but in the neighborhood of 1: there is a threshold Δ such that $\Delta \leq \mu(x; L_0) < 1$), and "capital blunders" (i.e., strings x over Σ with $\mu(x; L_0)$ unequal to, but in the neighborhood of 0: there is another threshold δ such that $0 < \mu(x; L_0) \leq \delta$) with respect to $c(L_0)$; cf. [9]. Now it is natural to demand that a recognizer or parser M for L_0 computes the membership function $\mu_{L_0} : \Sigma^\star \to \mathcal{L}$ rather than the characteristic function of $c(L_0)$. In essence, this reflects the notion of robustness that we use in this paper, modeling the feature that M should be able to deal with "small errors" in its input.

In this paper we first restrict ourselves to recognizing rather than to parsing of fuzzy context-free languages as in [4,5], but —as we will see— our main results can be easily extended to corresponding robust parsing algorithms. In Section 3 we provide an alternative functional version of Cocke–Younger–Kasami's recognition algorithm for (ordinary, crisp, non-fuzzy) context-free languages. Top-down variants of the bottom-up algorithm of Section 3 are discussed in Section 4; this leads to some recursive descent recognizers. These functional versions happen to be a good starting point for the design of recognition algorithms for fuzzy context-free languages, which are discussed in Sections 5 (Cocke–Younger–Kasami's algorithm for fuzzy context-free languages) and 6 (functional recursive descent recognizers for fuzzy context-free languages). Then in Section 7 we modify the recognition algorithms from Sections 5 and 6 to corresponding parsing algorithms. The remaining two sections contain preliminaries and examples (Section 2) and concluding remarks (Section 8).

2 Preliminaries and Examples

We assume the reader to be familiar with the rudiments of formal languages, grammars and parsing as presented in standard texts like [1,14,15,24]. For the original source of fuzzy languages and grammars we refer to [18]. But the approach to fuzzy languages we will use is the one introduced in [6,7]; cf. also [9]. Obviously, the present paper relies to some extend on its predecessor; so we assume some familiarity with Sections 1–4 of [9].

For each crisp set X, the power set of X is denoted by $\mathbb{P}(X)$, i.e., $\mathbb{P}(X)$ is the set of all subsets of X: $\mathbb{P}(X) = \{S \mid S \subseteq X\}$.

A context-free grammar $G = (V, \Sigma, P, S)$ is called λ-*free* if the right-hand side of each rule is unequal to the empty word λ, i.e., if $A \to \omega$ is in P, then $\omega \neq \lambda$. Or, when we consider P as a substitution (cf. Section 4 in [9]), then P is a nested λ-free substitution over V.

Remember that a language L is called λ-*free* if it does not contain λ, i.e., if $L \subseteq \Sigma^+$. Of course, for a fuzzy language the requirement reads: $\mu(\lambda; L) = 0$ since $\mu(\lambda; L) \in \{0, 1\}$; cf. [9]. So a λ-free [fuzzy] context-free grammar generates a λ-free [fuzzy] language.

For our algorithms in subsequent sections, the fuzzy context-free grammars ought to possess a well-known special form. Recall that a λ-free context-free grammar $G = (V, \Sigma, P, S)$ is in *Chomsky normal form* if $P \subseteq N \times (\Sigma \cup N^2)$ where $N = V - \Sigma$. And G is in *Greibach 2-form* if $P \subseteq N \times \Sigma(\{\lambda\} \cup N \cup N^2)$. If P is viewed as a nested finite λ-free substitution over V (as in [9]), these conditions read as $P(\sigma) = \{\sigma\}$ ($\sigma \in \Sigma$), and for each $A \in N$, $P(A) = \{A\} \cup L_A$ for some L_A with $L_A \subseteq \Sigma \cup N^2$, and $L_A \subseteq \Sigma(\{\lambda\} \cup N \cup N^2)$, respectively.

Remark that our definitions of Chomsky normal form and of Greibach 2-form slightly differ from (particularly, they are weaker than) the familiar ones [1,14,15] in the sense that we allow the case in which the initial symbol S of the grammar is recursive.

Example 2.1. Let $G_1 = (V, \Sigma, P_1, S)$ and $G_2 = (V, \Sigma, P_2, S)$ be the context-free grammars of Examples 4.1 and 4.2 in [9]: $\Sigma = \{a, b\}$, $N = \{S, A, B\}$, $V = N \cup \Sigma$ with P_1 and P_2, viewed as λ-free nested substitutions, given by

$P_1(S) = \{S, AB, BA\}$,	$P_2(S) = \{S, aSB, aBS, bSA, bAS, aB, bA\}$,
$P_1(A) = \{A, AS, SA, a\}$,	$P_2(A) = \{A, aS, a\}$,
$P_1(B) = \{B, BS, SB, b\}$,	$P_2(B) = \{B, bS, b\}$,
$P_1(a) = \{a\}$,	$P_2(a) = \{a\}$,
$P_1(b) = \{b\}$,	$P_2(b) = \{b\}$.

Then both G_1 and G_2 are λ-free context-free grammars; G_1 is in Chomsky normal form and G_2 is in Greibach 2-form. In both grammars the initial symbol S is recursive. □

Throughout this paper, by "fuzzy" we mean "$\mathcal{L}$-fuzzy" as in [9], where $\mathcal{L}$ is an arbitrary type-00 lattice; cf. Definition 2.2 in [9]. All our algorithms for fuzzy context-free grammars refer to this general case. However, in examples we will restrict our attention to one of the following two special instances:

• $\mathcal{L} = \mathcal{I}$, where $\mathcal{I}$ —or the "interval"— equals the type-11 lattice of Example 2.3(4) in [9]: $\mathcal{I}$ is the closed real interval $[0, 1]$ provided with the operations max and min, whereas the third operation $\star$, being equal to min, is redundant.

• $\mathcal{L} = \mathcal{M}$, where $\mathcal{M}$ —or the "multiplicative interval"— is the type-10 lattice of Example 2.3(3) in [9]: $\mathcal{M}$ is the closed real interval $[0, 1]$ provided with the operations max, min and $\star$ (multiplication).

As in [9] we will not "use" all elements of $[0, 1]$: we restrict ourselves to the computable, or even to the rational elements in $[0, 1]$; cf. [12] for the effect of allowing noncomputable reals in the codomain of membership functions.

In order to represent fuzzy sets concisely, we use some additional notation:

• Let X be a finite $\mathcal{L}$-fuzzy set with support $s(X) = \{x_1, \ldots, x_n\}$ and $\mu(x_i; X) = r_i$ $(1 \leq i \leq n)$. Then we will represent X as $X = \{x_1/_{r_1}, \ldots, x_n/_{r_n}\}$.

• In case all elements of X have the same degree of membership r, i.e., $X = \{x_1/_r, \ldots, x_n/_r\}$, we write $\langle X \rangle_r$ or $X/_r$. Note that for each $r, s \in \mathcal{L}$, we have $\langle \langle X \rangle_r \rangle_s = \langle X \rangle_{r \star s}$.

• If $r = 1$, we write x, X, and X instead of $x/_1$, $\langle X \rangle_1$, and $X/_1$, respectively. And, of course, in case $r = 0$, we have $\langle X \rangle_0 = X/_0 = \varnothing$.

Example 2.2. Consider the $\mathcal{I}$-fuzzy context-free grammar $G_3 = (V, \Sigma, P_3, S)$ from Example 4.4 in [9]: $N = V - \Sigma = \{S, A, B\}$, $\Sigma = \{a, b\}$, and P_3 is defined by the following λ-free nested fuzzy substitution over V:

$$P_3(S) = \{S, AB, BA, AA/_{0.1}, BB/_{0.9}\},$$
$$P_3(A) = \{A, AS, SA, a\}, \qquad P_3(a) = \{a\},$$
$$P_3(B) = \{B, BS, SB, b\}, \qquad P_3(b) = \{b\}.$$

According to the notational convention above, we have $\mu(AA, P_3(S)) = 0.1$ and $\mu(BB, P_3(S)) = 0.9$, while all other degrees of membership are equal to 1. The crisp language $c(L(G_3))$ equals the language $L(G_1)$ of Example 2.1.

Instead of the "correct" rules $S \to AB$ or $S \to BA$, now the "incorrect" rule $S \to BB$ may be applied too. Clearly, this is viewed as a minor error since $\mu(BB, P_3(S)) = 0.9$. An example of a corresponding derivation is

$$S \;\Rightarrow\; AB \;\Rightarrow\; ASB \;\Rightarrow\; ABBB \;\Rightarrow^4\; ab^3 \tag{1}$$

and so $\mu(ab^3; L(G_3)) = 0.9$. Making this "tiny mistake" twice results in

$$S \;\Rightarrow\; BB \;\Rightarrow\; BSB \;\Rightarrow\; BBBB \;\Rightarrow^4\; b^4 \tag{2}$$

with $\mu(b^4; L(G_3)) = 0.9$. Similarly, making one or more "capital blunders" (i.e., applying the incorrect rule $S \to AA$ rather than $S \to AB$ or $S \to BA$) results in derivations like

$$S \Rightarrow AB \Rightarrow ASB \Rightarrow AAAB \Rightarrow^4 a^3b \quad (3)$$
$$S \Rightarrow AA \Rightarrow ASA \Rightarrow AAAA \Rightarrow^4 a^4 \quad (4)$$
which yields words in $L(G_3)$ with degree of membership equal to 0.1 as $\mu(AA, P_3(S)) = 0.1$.

Summarizing we have that the fuzzy context-free grammar G_3 generates all nonempty even length strings over $\{a, b\}$ with preferably as many a's as b's (degree of membership equal to 1). Some a's in these nonempty even length strings may have changed into b's (tiny mistakes) or, conversely, some b's are erroneously rewritten into a's (capital blunders). □

Modeling grammatical errors as in Example 2.2 has a serious shortcoming. Indeed, when we compare derivations (1) and (2), then the resulting terminal strings both receive a degree of membership of 0.9. However, intuitively we have the idea that b^4 is "worse" than ab^3, since in (2) two grammatical errors have been made and so we expect that $\mu(b^4; L(G_3)) < \mu(ab^3; L(G_3))$. A similar observation can be made when we compare derivations (3) and (4).

But this inequality is impossible to obtain when $\mathcal{L}$ is a type-11 lattice such as $\mathcal{I}$: we only have the operations max and min. Applying these operations does not yield any membership value different from the finite number of values already present in P_3. Formally, for each type-11 lattice $\mathcal{L}$ and each $\mathcal{L}$-fuzzy context-free grammar $G = (V, \Sigma, P, S)$, we have the inclusion

$$\{\mu(x; L(G)) \mid x \in \Sigma^\star\} \subseteq \{\mu(\omega; P(\alpha)) \mid \omega \in V^\star,\ \alpha \in V\}.$$

Taking $\mathcal{L}$ equal to a type-10 lattice enables us to model the accumulation of errors properly, as this type of lattices possesses an additional multiplication operation $\star$. Repetitively applying this latter operation on the real interval $[0, 1]$ results in strictly decreasing degrees of membership (Lemma 2.4 in [9]).

Note that for each $\mathcal{M}$-fuzzy context-free grammar $G = (V, \Sigma, P, S)$, if the set of values $\{\mu(\omega; P(\alpha)) \mid \omega \in V^\star,\ \alpha \in V\}$ contains computable reals [rational numbers, respectively] only, then so does the set $\{\mu(x; L(G)) \mid x \in \Sigma^\star\}$.

Example 2.3. Let $G_4 = (V, \Sigma, P_4, S)$ be the $\mathcal{M}$-fuzzy context-free grammar which is equal to G_3 of Example 2.2 except that $\mathcal{I}$ has been replaced by $\mathcal{M}$.

For the derivations (1) and (2) of Example 2.2, we now have $\mu(ab^3; L(G_4)) = 0.9$ and $\mu(b^4; L(G_4)) = 0.81$, respectively. Clearly, this meets our intuition that $\mu(b^4; L(G_4)) < \mu(ab^3; L(G_4))$. Similarly, we have $\mu(a^3b; L(G_4)) = 0.1$ and $\mu(a^4; L(G_4)) = 0.01$ and so $\mu(a^4; L(G_4)) < \mu(a^3b; L(G_4))$; cf. (3) and (4).

In general, we have for each w over the alphabet $\{a, b\}$,

- $\mu(w; L(G_4)) = 1$ iff $\#_a(w) = \#_b(w)$ and $w \neq \lambda$,
- $\mu(w; L(G_4)) = (\frac{9}{10})^{(\#_b(w) - \#_a(w))/2}$ iff $\#_b(w) \geq \#_a(w) + 2$ and $|w|$ is even,
- $\mu(w; L(G_4)) = (\frac{1}{10})^{(\#_a(w) - \#_b(w))/2}$ iff $\#_a(w) \geq \#_b(w) + 2$ and $|w|$ is even,
- $\mu(w; L(G_4)) = 0$ iff either $w = \lambda$ or $|w|$ is odd. □

Example 2.4. Let $G_5 = (V_5, \Sigma_5, P_5, S)$ be the context-free grammar with $V_5 = \Sigma_5 \cup \{S\}$, $\Sigma_5 = \{\blacktriangleleft, \blacktriangleright, [\,,]\,\}$ and P_5 consists of the rules

$S \;\rightarrow\; [\,S]\,S \;\mid\; \blacktriangleleft S \blacktriangleright S \;\mid\; \lambda.$

The language $L(G_5)$ is called the *Dyck language over two types of parentheses* and it consists of all well-matched sequences over Σ_5. So $[\,[\,]\,\blacktriangleleft\blacktriangleright]$ and $\blacktriangleleft\blacktriangleleft\blacktriangleright[\,]\,\blacktriangleleft\blacktriangleright\blacktriangleright$ are in $L(G_5)$, but $[\,\blacktriangleleft\blacktriangleright[\,]$ and $[\,\blacktriangleleft]\,\blacktriangleright$ are not. This language $L(G_5)$ plays an important rôle in the theory of context-free languages, since any context-free language L_0 can be obtained from $L(G_5)$ by the application of an appropriate non-deterministic finite-state transducer T, i.e. $L_0 = T(L(G_5))$, where T depends on L_0; cf. e.g. [10,13,14].

When we view P_5 as a nested finite substitution over V_5, we have $P_5(S) = \{S, \blacktriangleleft S \blacktriangleright S, [\,S]\,S, \lambda\}$ and $P_5(\sigma) = \{\sigma\}$ for each σ in Σ_5. □

Example 2.5. Let $G_6 = (V_5, \Sigma_5, P_6, S)$ be the $\mathcal{I}$-fuzzy context-free grammar equal to G_5 of Example 2.4 except that $P_6(S) = P_5(S) \cup \{[\,S\blacktriangleright S/_{0.9}, [\,SS/_{0.1}\}$, and $P_6(\sigma) = \{\sigma\}$ for each σ in Σ.

The string $[\,S\blacktriangleright S$ gives rise to, e.g., the following derivation

$S \Rightarrow [\,S]\,S \Rightarrow [\,\blacktriangleleft S \blacktriangleright S]\,S \Rightarrow^2 [\,\blacktriangleleft\blacktriangleright S]\; \Rightarrow [\,\blacktriangleleft\blacktriangleright[\,S\blacktriangleright S]\; \Rightarrow^2 [\,\blacktriangleleft\blacktriangleright[\,\blacktriangleright]\,.$

Now $\mu([\,\blacktriangleleft\blacktriangleright[\,\blacktriangleright]\,; L(G_6)) = 0.9$ and so $[\,\blacktriangleleft\blacktriangleright[\,\blacktriangleright]$ is a tiny mistake from which it is easy to recover. However, the word $[\,SS$ causes more problems: we have $\mu([\,[\,]\;[\,]\,; L(G_6)) = 0.1$, but what is the corresponding correct word? There are three possibilities: $[\,]\;[\,]\;[\,]$, $[\,[\,]\,]\;[\,]$ and $[\,[\,]\;[\,]\,]$. The string $[\,[\,]\;[\,]$ is viewed as a capital blunder, when we choose, for instance, the thresholds δ to be equal to 0.2 and Δ to 0.8; cf. Section 1. □

Example 2.6. In G_6 of Example 2.5 we replace the type-11 lattice $\mathcal{I}$ by the type-10 lattice $\mathcal{M}$. The resulting $\mathcal{M}$-fuzzy context-free grammar is called G_7.

Then we have $\mu([\,[\,\blacktriangleright[\,[\,]\,\blacktriangleright\blacktriangleright; L(G_7)) = 0.729$ and $\mu([\,[\,[\,; L(G_7)) = 0.001$. In both these examples three grammatical errors have been made. Now additional grammatical errors do decrease the degree of membership. □

We need the following result, which has been established in [18] for $\mathcal{L}$-fuzzy context-free grammars provided $\mathcal{L}$ equals the type-11 lattice of Example 2.3.(4) in [9]. The authors remark that their proof can be generalized to type-01 lattices. However, a straightforward argument shows that their proof can be extended to arbitrary type-00 lattices as well; see also [16].

Remember that two $\mathcal{L}$-fuzzy context-free grammars G_1 and G_2 are *equivalent* [9], if $L(G_1) = L(G_2)$, i.e., if $\mu(x; L(G_1)) = \mu(x; L(G_2))$ for all strings x.

Theorem 2.7. [18] *Let $\mathcal{L}$ be a type-00 lattice and let $G = (V, \Sigma, P, S)$ be an arbitrary $\mathcal{L}$-fuzzy context-free grammar.*

(1) *We can effectively construct an equivalent $\mathcal{L}$-fuzzy context-free grammar G_1 in Chomsky normal form.*

(2) *We can effectively construct an equivalent $\mathcal{L}$-fuzzy context-free grammar G_2 in Greibach 2-form.* □

Example 2.8. Consider the $\mathcal{M}$-fuzzy context-free grammar G_8 with $G_8 = (V_8, \Sigma_8, P_8, S)$, $\Sigma_8 = \Sigma_5 = \{\not<, \not>, [\,,]\,\}$, $V_8 = \Sigma_8 \cup \{S, A, B, C, D, E, F\}$, and P_8 consists of the rules

$$S \to SS \mid AC \mid BC \mid DF \mid EF \mid AF \mid BF \mid BS \mid [\,,$$
$$A \to BS, \qquad B \to [\,, \qquad C \to]\,,$$
$$D \to ES, \qquad E \to \not<, \qquad F \to \not>,$$

with $\mu(AF; P_8(S)) = \mu(BF; P_8(S)) = 0.9$, $\mu(BS; P_8(S)) = \mu([\,; P_8(S)) = 0.1$ and μ equals 1 in all other cases. This G_8 is λ-free, in Chomsky normal form, and $L(G_8) = L(G_7) \cap \Sigma^+$ where G_7 is the grammar from Example 2.6. □

X	$\omega/_{\mu(\omega;P_9(X))}$
S	S, $[SA$, $[A$, $[SC$, $[C$, $\not<SB$, $\not<B$, $\not<SD$, $\not<D$ $[SB/_{0.9}$, $[SD/_{0.9}$, $[B/_{0.9}$, $[D/_{0.9}$, $[SS/_{0.1}$, $[S/_{0.1}$, $[/_{0.1}$
A	A, $]\,S$
B	B, $\not>S$
C	C, $]$
D	D, $\not>$

Table 1
The definition of P_9.

Example 2.9. Let G_9 be the $\mathcal{M}$-fuzzy context-free grammar with $G_9 = (V_9, \Sigma_9, P_9, S)$ with $\Sigma_9 = \Sigma_5 = \{\not<, \not>, [\,,]\,\}$, $V_9 = \Sigma_9 \cup \{S, C, F\}$, and P_9 consists of rules $X \to \omega$ which are displayed with their degree of membership in Table 1. This grammar is λ-free and in Greibach 2-form. We have $L(G_9) = L(G_7) \cap \Sigma^+$ where G_7 is the grammar of Example 2.6. □

3 A Functional Version of Cocke–Younger–Kasami's Algorithm

In this section we discuss a functional version of Cocke–Younger–Kasami's algorithm, or CYK-algorithm for short, for recognizing (ordinary, non-fuzzy) context-free languages; cf. [3]. This functional version (Algorithm 3.3 below) is a good starting point to develop a robust algorithm for recognizing fuzzy context-free languages; see also [4,5].

Usually, the CYK-algorithm is presented in terms of nested **for**-loops filling an upper-triangular matrix; cf. [1,14,15].

Algorithm 3.1. [14] Given a λ-free context-free grammar $G = (V, \Sigma, P, S)$ in Chomsky normal form and a word $a_1 a_2 \cdots a_n$ $(n \geq 1)$ with $a_k \in \Sigma$ $(1 \leq k \leq n)$. Fill the strictly upper-triangular $(n+1) \times (n+1)$ recognition matrix T by the program of Figure 1, where each element $t_{i,j}$ is a subset of $N = V - \Sigma$ and is initially empty.

```
begin
  for i := 0 to n − 1 do  t_{i,i+1} := {A | a_{i+1} ∈ P(A)};
  for d := 2 to n do
    for i := 0 to n − d do
      begin j := d + i;
        t_{i,j} := {A | ∃k(i + 1 ≤ k ≤ j − 1) : ∃B(B ∈ t_{i,k}) :
                                          ∃C(C ∈ t_{k,j}) : BC ∈ P(A)}
      end
end.
```

Fig. 1. Algorithm 3.1.

Then $a_1 a_2 \cdots a_n \in L(G)$ if and only if $S \in t_{0,n}$. □

Example 3.2. Consider the context-free grammar G_1 in Chomsky normal form of Example 2.1 and the string *abba* over Σ. The initialization phase of

$i\backslash j$	1	2	3	4
0	$\{A\}$	$\{S\}$	$\{B\}$	$\{S\}$
1		$\{B\}$	$\varnothing$	$\{B\}$
2			$\{B\}$	$\{S\}$
3				$\{A\}$

Table 2
Recognition of *abba* by Algorithm 3.1.

Algorithm 3.1 yields: $t_{0,1} = \{A\}$, $t_{1,2} = \{B\}$, $t_{2,3} = \{B\}$ and $t_{3,4} = \{A\}$. Applying the iteration phase —i.e., the nested **for**-loops— results in the recognition matrix for *abba*; cf. Table 2. Since $S \in t_{0,4}$ we have $abba \in L(G_1)$. Next we

$i\backslash j$	1	2	3	4
0	$\{B\}$	$\varnothing$	$\varnothing$	$\varnothing$
1		$\{B\}$	$\varnothing$	$\{B\}$
2			$\{B\}$	$\{S\}$
3				$\{A\}$

Table 3
Recognition of *bbba* by Algorithm 3.1.

take the string *bbba* as input to this algorithm. Then $t_{0,1} = \{B\}$, $t_{1,2} = \{B\}$, $t_{2,3} = \{B\}$ and $t_{3,4} = \{A\}$, whereas the recognition matrix for this input is as in Table 3. Now we have $bbba \notin L(G_1)$, as $S \notin t_{0,4}$. □

The formulation of Algorithms 3.1, as given above, is well known and stems from [14]. But in [3] an alternative, functional version of this algorithm has been proposed. A pleasant feature of this functional formulation is the omission of implementation details like the data structure, the indices i, j and k and the length n of the input string.

In this alternative formulation we need two functions f and g that correspond to the initialization phase and the iteration phase, respectively. These functions $f : \Sigma^+ \to \mathbb{P}(N^+)$ and $g : \mathbb{P}(N^+) \to \mathbb{P}(N)$ are defined by:

- For each w in Σ^+, f is defined as the length-preserving finite substitution generated by

$$f(a) = \{A \mid a \in P(A)\} \tag{5}$$

and extended to nonempty words over Σ by

$$f(w) = f(a_1)f(a_2)\cdots f(a_n) \quad \text{if} \quad w = a_1a_2\cdots a_n \ \ (a_k \in \Sigma, \ \ 1 \le k \le n). \tag{6}$$

- The function g is defined in two steps. First, $g : N^+ \to \mathbb{P}(N)$ is defined by

$$g(A) = \{A\} \qquad (A \in N) \text{ and} \tag{7}$$

$$g(\omega) = \bigcup\{g(\chi) \otimes g(\eta) \mid \chi, \eta \in N^+, \ \omega = \chi\eta\} \qquad (\omega \in N^+, \ \ |\omega| \ge 2) \tag{8}$$

where for X and Y in $\mathbb{P}(N)$ the binary operation $\otimes$ is defined by

$$X \otimes Y = \{A \mid BC \in P(A), \text{ with } B \in X \text{ and } C \in Y\}. \tag{9}$$

In the second step we extend g to $g : \mathbb{P}(N^+) \to \mathbb{P}(N)$; viz. for each (finite) language M over N, $g(M)$ is defined by

$$g(M) = \bigcup\{g(\omega) \mid \omega \in M\}. \tag{10}$$

The functional version of the CYK-algorithm from [3] now reads as follows.

Algorithm 3.3. Let $G = (V, \Sigma, P, S)$ be a λ-free context-free grammar in Chomsky normal form and let w be a nonempty string over Σ. Compute $g(f(w))$ and determine whether S belongs to the set $g(f(w))$.

Clearly, we have $w \in L(G)$ if and only if $S \in g(f(w))$. □

Note that the iteration in Algorithm 3.1 has been replaced by recursion in Algorithm 3.3, since g is recursive; cf. (8).

Example 3.4. Applying Algorithm 3.3 to the grammar G_1 of Example 2.1 and to input words *abba* and *bbba* yields:

$$\begin{aligned}
g(f(abba)) &= g(f(a)f(b)f(b)f(a)) = g(\{A\}\{B\}\{B\}\{A\}) = g(\{ABBA\}) \\
&= g(ABBA) = g(ABB) \otimes g(A) \cup g(AB) \otimes g(BA) \cup g(A) \otimes g(BBA) \\
&= (g(AB) \otimes g(B) \cup g(A) \otimes g(BB)) \otimes g(A) \cup g(AB) \otimes g(BA) \cup g(A) \otimes \\
&\quad (g(BB) \otimes g(A) \cup g(B) \otimes g(BA)) = ((g(A) \otimes g(B)) \otimes g(B) \cup \\
&\quad g(A) \otimes (g(B) \otimes g(B)) \otimes g(A) \cup (g(A) \otimes g(B)) \otimes (g(B) \otimes g(A)) \cup \\
&\quad g(A) \otimes ((g(B) \otimes g(B)) \otimes g(A) \cup \cup g(B) \otimes (g(B) \otimes g(A))) \\
&= (((\{A\} \otimes \{B\}) \otimes \{B\} \cup \{A\} \otimes (\{B\} \otimes \{B\}) \otimes \{A\} \cup (\{A\} \otimes \{B\}) \otimes \\
&\quad (\{B\} \otimes \{A\}) \cup \{A\} \otimes ((\{B\} \otimes \{B\}) \otimes \{A\} \cup \{B\} \otimes (\{B\} \otimes \{A\})) \\
&= (\{S\} \otimes \{B\} \cup \{A\} \otimes \varnothing) \otimes \{A\} \cup \{S\} \otimes \{S\} \cup \{A\} \otimes (\varnothing \otimes \{A\} \cup \\
&\quad \{B\} \otimes \{S\}) = (\{B\} \cup \varnothing) \otimes \{A\} \cup \varnothing \cup \{A\} \otimes (\varnothing \cup \{B\}) \\
&= \{B\} \otimes \{A\} \cup \{A\} \otimes \{B\} = \{S\}, \text{ and}
\end{aligned}$$

$$
\begin{aligned}
g(f(bbba)) &= g(f(b)f(b)f(b)f(a)) = g(\{B\}\{B\}\{B\}\{A\}) = g(\{BBBA\}) \\
&= g(BBBA) = g(BBB) \otimes g(A) \cup g(BB) \otimes g(BA) \cup g(B) \otimes g(BBA) \\
&= \ \cdots \ = (\varnothing \otimes \{B\} \cup \{B\} \otimes \varnothing) \otimes \{A\} \cup \varnothing \otimes \{S\} \cup \{B\} \otimes \\
&\quad (\varnothing \otimes \{A\} \cup \{B\} \otimes \{S\}) = (\varnothing \cup \varnothing) \otimes \{A\} \cup \varnothing \cup \{B\} \otimes (\varnothing \cup \{B\}) \\
&= \varnothing \otimes \{A\} \cup \{B\} \otimes \{B\} = \varnothing \cup \varnothing = \varnothing.
\end{aligned}
$$

We conclude again that $abba \in L(G_1)$ and $bbba \notin L(G_1)$, as $S \in g(f(abba)) = \{S\}$ and $S \notin g(f(bbba)) = \varnothing$, respectively. □

4 Recursive Descent Recognizers

Clearly, Cocke–Younger–Kasami's algorithm is a bottom-up algorithm for recognizing λ-free context-free languages. Now the question naturally arises whether there exists a (functional) top-down analogue of the CYK-algorithm. This question has been answered affirmatively in [3], from which we quote the following definition and algorithms.

Definition 4.1. Let $G = (V, \Sigma, P, S)$ be a context-free grammar and $N = V - \Sigma$. The set $T(\Sigma, N)$ of *terms* over (Σ, N) is the smallest set satisfying

- λ is a term in $T(\Sigma, N)$ and each a ($a \in \Sigma$) is a term in $T(\Sigma, N)$.
- For each A in N and each term t in $T(\Sigma, N)$, $A(t)$ is a term in $T(\Sigma, N)$.
- If t_1 and t_2 are in $T(\Sigma, N)$, then $t_1 t_2$ is also a term in $T(\Sigma, N)$. □

This definition implies that if S_1 and S_2 are sets of terms over (Σ, N), then so is the set $S_1 S_2$ defined by $S_1 S_2 = \{t_1 t_2 \mid t_1 \in S_1,\ t_2 \in S_2\}$.

Algorithm 4.2. Let $G = (V, \Sigma, P, S)$ be a λ-free context-free grammar in Chomsky normal form and let w be a string in Σ^+. To each nonterminal symbol A in N we associate a function $\tilde{A} : \Sigma^\star \cup \{\perp\} \to \mathbb{P}(T(\Sigma, N))$ defined as follows. (We use the symbol $\perp$ to denote "undefined".)

First, $\tilde{A}(\perp) = \varnothing$ and $\tilde{A}(\lambda) = \{\lambda\}$ for each A in N. If the argument x of $\tilde{A}$ is a word of length 1 (i.e., x is in Σ), then

$$\tilde{A}(x) = \{\lambda \mid x \in P(A)\} \qquad (x \in \Sigma) \tag{11}$$

and in case the length $|x|$ of the word x is 2 or more, then

$$\tilde{A}(x) = \bigcup\{\tilde{B}(y)\tilde{C}(z) \mid BC \in P(A),\ \ y, z \in \Sigma^+,\ \ x = yz\}. \tag{12}$$

Finally, we compute $\tilde{S}(w)$ and determine whether λ belongs to $\tilde{S}(w)$.

It is straightforward to show that $w \in L(G)$ if and only if $\lambda \in \tilde{S}(w)$. □

Example 4.3. Let $G_{10} = (V_{10}, \Sigma_{10}, P_{10}, S)$ be defined by $\Sigma_{10} = \{$ ⧏, ⧐, [,] $\}$, $V_{10} = \Sigma_{10} \cup \{S, A, B, C, D, E, F\}$, and P_{10}, viewed as a λ-free nested substitution, is $P_{10}(\sigma) = \{\sigma\}$ ($\sigma \in \Sigma_{10}$), $P_{10}(S) = \{S, SS, AC, BC, DF, EF\}$, $P_{10}(A) = \{A, BS\}$, $P_{10}(B) = \{B, [\}$, $P_{10}(C) = \{C,]\}$, $P_{10}(D) = \{D, ES\}$, $P_{10}(E) = \{E, ⧏\}$ and $P_{10}(F) = \{F, ⧐\}$.

Clearly, G_{10} is in Chomsky normal form and $L(G_{10}) = L(G_5)$; cf. Examples 2.8 and 2.4. Applying Algorithm 4.2 to G_{10} yields for inputs `[]‹›` and `[‹]` :

$$\begin{aligned}\tilde{S}(\texttt{[]‹›}) = {} & \tilde{S}(\texttt{[‹}\,)\tilde{S}(\texttt{›}) \cup \tilde{S}(\texttt{[}\,)\tilde{S}(\texttt{‹›}) \cup \tilde{S}(\texttt{[}\,)\tilde{S}(\texttt{]‹›}) \cup \tilde{A}(\texttt{[‹}\,)\tilde{C}(\texttt{›})\cup \\ & \tilde{A}(\texttt{[}\,)\tilde{C}(\texttt{‹›}) \cup \tilde{A}(\texttt{[}\,)\tilde{C}(\texttt{]‹›}) \cup \tilde{B}(\texttt{[‹}\,)\tilde{C}(\texttt{›}) \cup \tilde{B}(\texttt{[}\,)\tilde{C}(\texttt{‹›})\cup \\ & \tilde{B}(\texttt{[}\,)\tilde{C}(\texttt{]‹›}) \cup \tilde{D}(\texttt{[‹}\,)\tilde{F}(\texttt{›}) \cup \tilde{D}(\texttt{[}\,)\tilde{F}(\texttt{‹›}) \cup \tilde{D}(\texttt{[}\,)\tilde{F}(\texttt{]‹›})\cup \\ & \tilde{E}(\texttt{[‹}\,)\tilde{F}(\texttt{›}) \cup \tilde{E}(\texttt{[}\,)\tilde{F}(\texttt{‹›}) \cup \tilde{E}(\texttt{[}\,)\tilde{F}(\texttt{]‹›}) = \ \cdots \ = \\ = {} & (\tilde{B}(\texttt{[}\,)\tilde{C}(\texttt{]}\,))(\tilde{E}(\texttt{‹}\,)\tilde{F}(\texttt{›})) = (\{\lambda\}\{\lambda\})(\{\lambda\}\{\lambda\}) = \{\lambda\}\{\lambda\} = \{\lambda\}, \text{ and}\end{aligned}$$

$$\begin{aligned}\tilde{S}(\texttt{[‹]}\,) = {} & \tilde{S}(\texttt{[‹}\,)\tilde{S}(\texttt{]}\,) \cup \tilde{S}(\texttt{[}\,)\tilde{S}(\texttt{‹]}\,) \cup \tilde{A}(\texttt{[‹}\,)\tilde{C}(\texttt{]}\,) \cup \tilde{A}(\texttt{[}\,)\tilde{C}(\texttt{‹]}\,)\cup \\ & \tilde{B}(\texttt{[‹}\,)\tilde{C}(\texttt{]}\,) \cup \tilde{B}(\texttt{[}\,)\tilde{C}(\texttt{‹]}\,) \cup \tilde{D}(\texttt{[‹}\,)\tilde{F}(\texttt{]}\,) \cup \tilde{D}(\texttt{[}\,)\tilde{F}(\texttt{‹]}\,)\cup \\ & \tilde{E}(\texttt{[‹}\,)\tilde{F}(\texttt{]}\,) \cup \tilde{E}(\texttt{[}\,)\tilde{F}(\texttt{‹]}\,) = \tilde{A}(\texttt{[‹}\,) \cup \tilde{C}(\texttt{‹]}\,) = \tilde{B}(\texttt{[}\,)\tilde{S}(\texttt{‹}) = \varnothing.\end{aligned}$$

Here we used equalities like $\tilde{S}(\sigma) = \tilde{A}(\sigma) = \tilde{D}(\sigma) = \varnothing$ ($\sigma \in \Sigma_{10}$), $\tilde{B}(x) = \tilde{C}(x) = \tilde{E}(x) = \tilde{F}(x) = \varnothing$ ($x \in \Sigma_{10}^+$ with $|x| \geq 2$) and, of course, $X \cdot \varnothing = \varnothing \cdot X = \varnothing$ ($X \subseteq T(\Sigma_{10}, V_{10} - \Sigma_{10})$).

Now we conclude that `[]‹›` $\in L(G_{10})$ and `[‹]` $\notin L(G_{10})$, since we have $\lambda \in \tilde{S}(\texttt{[]‹›})$ and $\lambda \notin \tilde{S}(\texttt{[‹]}\,)$, respectively. □

Starting from Greibach 2-form instead of Chomsky normal form (as in Algorithm 4.2) yields the following recursive descent recognition algorithm.

Algorithm 4.4. Let $G = (V, \Sigma, P, S)$ be a λ-free context-free grammar in Greibach 2-form and let $w \in \Sigma^+$. The algorithm is as Algorithm 4.2, but (12) is replaced by

$$\begin{aligned}\tilde{A}(x) = {} & \bigcup\{\tilde{B}(y)\tilde{C}(z) \mid aBC \in P(A), \ \ y, z \in \Sigma^+, \ \ x = ayz\}\cup \\ & \bigcup\{\tilde{B}(y) \mid aB \in P(A), \ \ y \in \Sigma^+, \ \ x = ay\}. \qquad (13)\end{aligned}$$

Still we have that $w \in L(G)$ if and only if $\lambda \in \tilde{S}(w)$. □

Example 4.5. Consider the context-free grammar G_2 of Example 2.1. This grammar is in Greibach 2-form; so we may apply Algorithm 4.4: $\tilde{A}(x) = \tilde{S}(a\backslash x)$, $\tilde{B}(x) = \tilde{S}(b\backslash x)$ and

$$\begin{aligned}\tilde{S}(x) = {} & \bigcup\{\tilde{S}(y)\tilde{B}(z) \mid y, z \in \Sigma^+, \ x = ayz\}\cup \\ & \bigcup\{\tilde{B}(y)\tilde{S}(z) \mid y, z \in \Sigma^+, \ x = ayz\} \cup \bigcup\{\tilde{S}(y)\tilde{A}(z) \mid y, z \in \Sigma^+, \ x = byz\}\cup \\ & \bigcup\{\tilde{A}(y)\tilde{S}(z) \mid y, z \in \Sigma^+, \ x = byz\} \cup \tilde{B}(a\backslash x) \cup \tilde{A}(b\backslash x),\end{aligned}$$

where $u\backslash v = w$ if $v = uw$, and $\perp$ otherwise ($u, v, w \in \Sigma^\star$). Similarly, we define $u/v = w$ if $u = wv$, and $\perp$ otherwise. Remember that for each nonterminal symbol A, we have $\tilde{A}(\perp) = \varnothing$.

These three equalities reduce to

$$\begin{aligned}\tilde{S}(x) = {} & \bigcup\{\tilde{S}(a\backslash x/b), \tilde{S}(ab\backslash x), \tilde{S}(b\backslash x/a), \tilde{S}(ba\backslash x)\}\cup \\ & \bigcup\{\tilde{S}(y)\tilde{S}(b\backslash z), \tilde{S}(b\backslash y)\tilde{S}(z) \mid \ y, z \in \Sigma^+, \ x = ayz\}\cup \\ & \bigcup\{\tilde{S}(y)\tilde{S}(a\backslash z), \tilde{S}(a\backslash y)\tilde{S}(z) \mid \ y, z \in \Sigma^+, \ x = byz\}.\end{aligned}$$

As examples consider $\tilde{S}(abba) = \tilde{S}(ba) \cup \tilde{S}(b)\tilde{S}(a) = \{\lambda\} \cup \tilde{S}(b)\tilde{S}(a) = \{\lambda\} \cup \varnothing = \{\lambda\}$, and $\tilde{S}(aba) = \tilde{S}(a) \cup \tilde{S}(a)\tilde{S}(\lambda) = \tilde{S}(a) = \varnothing$. Since $\lambda \in \tilde{S}(abba)$ and $\lambda \notin \tilde{S}(aba)$, we have $abba \in L(G_2)$ and $aba \notin L(G_2)$, respectively. □

5 The CYK-Algorithm Adapted for Fuzzy Context-Free Languages

As pointed out in Section 1, a minimal requirement for a recognition or parsing algorithm to be called *robust*—in the context of fuzzy context-free languages—is, that the algorithm is able to compute the degree of membership of its input with respect to a given fuzzy context-free grammar. Henceforth in this paper, we will use this minimal notion of robustness.

Once we have the CYK-algorithm in the functional form of Section 3, it is easy to obtain a robust modification for recognizing fuzzy context-free languages.

Algorithm 5.1. Let $G = (V, \Sigma, P, S)$ be a λ-free fuzzy context-free grammar in Chomsky normal form and let w be a string over Σ. Extend (5)–(10) in Algorithm 3.3 with

$$\mu(A; f(a)) = \mu(a; P(A)),$$
$$\mu(A; g(A)) = 1, \qquad (A \in N),$$
$$\mu(A; g(\omega)) = \bigvee\{\mu(A; g(\chi) \otimes g(\eta)) \mid \chi, \eta \in N^+,\ \omega = \chi\eta\}, \quad (\omega \in N^+,\ |\omega| \geq 2),$$
$$\mu(A; X \otimes Y) = \bigvee\{\mu(BC; P(A)) \star \mu(B; X) \star \mu(C; Y) \ \mid \ B, C \in N\},$$

for (5), (7), (8) and (9) respectively, whereas corresponding equalities for (6) and (10) can be obtained by the definitions for concatenation and finite union, respectively; cf. Section 2 of [9]. Finally, compute $\mu(S; g(f(w))$.

Then we have $\mu(w; L(G)) = \mu(S; g(f(w))$ for each w in Σ^+. □

Example 5.2. Consider the $\mathcal{I}$-fuzzy context-free grammar G_3 of Example 2.2. Applying Algorithm 5.1 yields

$$\begin{aligned}\mu(abba; L(G_3)) &= \mu(S; g(f(abba))) = \mu(S; g(ABBA)) = \\ &= \mu(S; \{g(A) \otimes g(BBA), g(AB) \otimes g(BA), g(ABB) \otimes g(A)\}) = \cdots = 1,\end{aligned}$$

$$\begin{aligned}\mu(abbb; L(G_3)) &= \mu(S; g(f(abbb))) = \mu(S; g(ABBB)) = \\ &= \mu(S; \{g(A) \otimes g(BBB), g(AB) \otimes g(BB), g(ABB) \otimes g(B)\}) = \\ &= \cdots = 0.9, \text{ and}\end{aligned}$$

$$\begin{aligned}\mu(aaab; L(G_3)) &= \mu(S; g(f(aaab))) = \mu(S; g(AAAB)) = \\ &= \mu(S; \{g(A) \otimes g(AAB), g(AA) \otimes g(AB), g(AAA) \otimes g(B)\}) = \\ &= \cdots = 0.1.\end{aligned}$$

Since G_3 is $\mathcal{I}$-fuzzy, the accumulation of grammatical errors does not result in decreasing degrees of membership. Since $\{\mu(x; L(G_3)) \mid x \in \Sigma^\star\} = \{\mu(\omega; P_3(\alpha)) \mid \omega \in V^\star, \alpha \in V\} = \{0, 0.1, 0.9, 1\}$, we have $\mu(bbbb; L(G_3) = 0.9$ and $\mu(aaaa; L(G_3) = 0.1$. □

Example 5.3. Now we apply Algorithm 5.1 to the $\mathcal{M}$-fuzzy context-free grammar G_4 of Example 2.3. Then $\mu(abba; L(G_4)) = 1$, $\mu(abbb; L(G_4)) = 0.9$, $\mu(aaab; L(G_4)) = 0.1$, but

$$\mu(bbbb; L(G_4)) = \mu(S; g(f(bbbb))) = \mu(S; g(BBBB)) = \cdots = 0.81,$$

$$\mu(aaaa; L(G_4)) = \mu(S; g(f(aaaa))) = \mu(S; g(AAAA)) = \cdots = 0.01, \text{ and}$$

$$\mu(aab; L(G_4)) = \mu(S; g(f(aab))) = \mu(S; g(AAB)) = \cdots = 0.$$

Now repetitively making grammatical errors does decrease the degree of membership and $\{\mu(x; L(G_4)) \mid x \in \Sigma^\star\}$ is a subset of the closure of $\{\mu(\omega; P_4(\alpha)) \mid \omega \in V^\star,\ \alpha \in V\} = \{0,\ 0.1,\ 0.9,\ 1\}$ under the $\star$-operation (multiplication). □

When we compare Algorithms 3.3 and 5.1, it is easy to design an $\mathcal{L}$-fuzzy counterpart of Algorithm 3.1 (the "traditional", non-functional CYK-algorithm); viz. Algorithm 5.4.

begin
  for $i := 0$ **to** $n-1$ **do** $t_{i,i+1} := \{A/_m \mid A \in N,\ \mu(a_{i+1}; P(A)) = m > 0\}$
  for $d := 2$ **to** n **do**
    for $i := 0$ **to** $n-d$ **do**
      begin $j := d + i$;
        $t_{i,j} := \{A/_m \mid A \in N,\ m = \bigvee\{r \star p \star q \mid B/_p \in t_{i,k},\ C/_q \in t_{k,j},$
                $\mu(BC, P(A)) = r > 0;\ i+1 \le k \le j-1\ \}\ \}$
      end
end.

Fig. 2. Algorithm 5.4.

Algorithm 5.4. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Chomsky normal form and let $a_1 a_2 \cdots a_n$ $(n \ge 1)$ with $a_k \in \Sigma$ $(1 \le k \le n)$ be a string. Construct the strictly upper-triangular $(n+1)\times(n+1)$ recognition matrix T as in Figure 2, where each element $t_{i,j}$ is a finite subset of $N \times \mathcal{L}$ with $N = V - \Sigma$. As usual, each $t_{i,j}$ is initially empty.

Then for each $m > 0$ $(m \in \mathcal{L})$, $\mu(a_1 a_2 \cdots a_n; L(G)) = m$ iff $S/_m \in t_{0,n}$. □

$i\backslash j$	1	2	3	4
0	$\{S/_{0.1}, B/_1\}$	$\{S/_{0.01}, A/_{0.1}\}$	$\{S/_{0.09}, A/_{0.9}\}$	$\{S/_{0.81}\}$
1		$\{S/_{0.1}, B/_1\}$	$\{S/_{0.9}\}$	$\varnothing$
2			$\{F/_1\}$	$\varnothing$
3				$\{F/_1\}$

Table 4
Recognition of [[⦘⦘ by Algorithm 5.4.

Example 5.5. Consider the $\mathcal{M}$-fuzzy context-free grammar G_8 in Chomsky normal form of Example 2.8 and the string [[⦘⦘ over Σ_8. Clearly, during the generation of [[⦘⦘ two small grammatical errors ("tiny mistakes") occurred.

$i\backslash j$	1	2	3	4
0	$\{S/_{0.1}, B/_1\}$	$\{S/_{0.01}, A/_{0.1}\}$	$\{S/_{0.09}, A/_{0.9}\}$	$\{S/_{0.009}, A/_{0.9}\}$
1		$\{S/_{0.1}, B/_1\}$	$\{S/_{0.9}\}$	$\{S/_{0.09}\}$
2			$\{F/_1\}$	$\varnothing$
3				$\{S/_{0.1}, B/_1\}$

Table 5
Recognition matrix for [[⋗ [by Algorithm 5.4.

Algorithm 5.4 starts with computing $t_{0,1} = \{S/_{0.1}, B/_1\}$, $t_{1,2} = \{S/_{0.1}, B/_1\}$, $t_{2,3} = \{F/_1\}$ and $t_{3,4} = \{F/_1\}$. Since we have $\mu(AF; P_8(S)) = \mu(BF; P_8(S)) = 0.9$, applying the nested **for**-loops results in the recognition matrix for [[⋗⋗ of Table 4. Since $S/_{0.81} \in t_{0,4}$ we have $\mu([\,[\,\gtrdot\gtrdot; L(G_8)) = 0.81$.

The word [[⋗ [is obtained by making two "capital blunders" and one "tiny mistake". First, we obtain: $t_{0,1} = \{S/_{0.1}, B/_1\}$, $t_{1,2} = \{S/_{0.1}, B/_1\}$, $t_{2,3} = \{F/_1\}$ and $t_{3,4} = \{S/_{0.1}, B/_1\}$, whereas the recognition matrix for this input is in Table 5. So we have $\mu([\,[\,\gtrdot\,[\,; L(G_8)) = 0.009$, as $S/_{0.009} \in t_{0,4}$. □

6 Recursive Descent Recognizers for Fuzzy Context-Free Languages

This section is devoted to robust versions of Algorithms 4.2 and 4.4. So we obtain algorithms to recognize λ-free $\mathcal{L}$-fuzzy context-free languages generated by grammars in Chomsky normal form (Algorithm 6.1) and in Greibach 2-form (Algorithm 6.3).

Algorithm 6.1. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Chomsky normal form and let w be a string in Σ^+. For all A in N, $\mu(\lambda; \tilde{A}(\lambda)) = 1$ and $\mu(t; \tilde{A}(\perp)) = 0$ for each t in $T(\Sigma, N)$. Extend (11)–(12) in Algorithm 4.2 with

$$\mu(\lambda; \tilde{A}(x)) = \mu(x; P(A)), \qquad (x \in \Sigma),$$
$$\mu(\lambda; \tilde{A}(x)) = \bigvee\{\mu(\lambda; \tilde{B}(y)) \star \mu(\lambda; \tilde{C}(z)) \star \mu(BC; P(A)) \mid x = yz,\ y, z \in \Sigma^+\}.$$

Finally, we compute $\mu(\lambda; \tilde{S}(w))$. Then we have $\mu(w; L(G)) = \mu(\lambda; \tilde{S}(w))$. □

Example 6.2. Applying Algorithm 6.1 to the $\mathcal{M}$-fuzzy context-free grammar G_4 of Example 2.3 results, for example, in

$$\begin{aligned}
&\mu(bbbb; L(G_4)) = \mu(\lambda; \tilde{S}(bbbb)) = \\
&= \mu(\lambda; \tilde{A}(bbb)\tilde{B}(b) \cup \tilde{A}(bb)\tilde{B}(bb) \cup \tilde{A}(b)\tilde{B}(bbb) \cup \tilde{B}(bbb)\tilde{A}(b) \cup \tilde{B}(bb)\tilde{A}(bb) \cup \\
&\qquad \tilde{B}(b)\tilde{A}(bbb) \cup \langle \tilde{A}(bbb)\tilde{A}(b) \cup \tilde{A}(bb)\tilde{A}(bb) \cup \tilde{A}(b)\tilde{A}(bbb)\rangle_{0.1} \cup \\
&\qquad \langle \tilde{B}(bbb)\tilde{B}(b) \cup \tilde{B}(bb)\tilde{B}(bb) \cup \tilde{B}(b)\tilde{B}(bbb)\rangle_{0.9}) = \ \cdots \ = \\
&= \mu(\lambda; \langle \tilde{B}(b)\tilde{S}(bb) \cup \tilde{S}(bb)\tilde{B}(b)\rangle_{0.9}) = \mu(\lambda; \langle \tilde{S}(bb)\rangle_{0.9}) = \\
&= \mu(\lambda; \langle \tilde{A}(b)\tilde{B}(b) \cup \tilde{B}(b)\tilde{A}(b) \cup \langle \tilde{A}(b)\tilde{A}(b)\rangle_{0.1} \cup \langle \tilde{B}(b)\tilde{B}(b)\rangle_{0.9}\rangle_{0.9}) = \\
&= \mu(\lambda; \langle \langle \tilde{B}(b)\tilde{B}(b)\rangle_{0.9}\rangle_{0.9}) = \mu(\lambda; \{\lambda/_{0.81}\}) = 0.81,
\end{aligned}$$

Here we used equalities like $\tilde{A}(b) = \tilde{B}(a) = \varnothing$ and $\tilde{A}(a) = \tilde{B}(b) = \{\lambda/_1\}$.

In a similar way we can derive that

$$\mu(aabb; L(G_4)) = \mu(\lambda; \tilde{S}(aabb)) = \ \cdots \ = 1,$$
$$\mu(aaaa; L(G_4)) = \mu(\lambda; \tilde{S}(aaaa)) = \ \cdots \ = 0.01,$$
$$\mu(abb; L(G_4)) = \mu(\lambda; \tilde{S}(abb)) = \ \cdots \ = 0.$$

It is useful to compare these computations with those in Example 5.3. □

Algorithm 6.3. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Greibach 2-form and let w be a word in Σ^+. For all A in N, $\mu(\lambda; \tilde{A}(\lambda)) = 1$ and $\mu(t; \tilde{A}(\bot)) = 0$ for each t in $T(\Sigma, N)$. Extend (13) in Algorithm 4.4 with

$$\begin{aligned}&\mu(\lambda; \tilde{A}(x)) = \\ &= \bigvee\{\mu(\lambda; \tilde{B}(y)) \star \mu(\lambda; \tilde{C}(z)) \star \mu(aBC; P(A)) \mid x = ayz,\ a \in \Sigma,\ y, z \in \Sigma^+\} \vee \\ &\quad \bigvee\{\mu(\lambda; \tilde{B}(y)) \star \mu(aB; P(A)) \mid x = ay,\ a \in \Sigma,\ \ \in \Sigma^+\}.\end{aligned}$$

Finally, we compute $\mu(\lambda; \tilde{S}(w))$. Then we have $\mu(w; L(G)) = \mu(\lambda; \tilde{S}(w))$. □

Example 6.4. The $\mathcal{M}$-fuzzy context-free grammar of Example 2.9 is in Greibach 2-form. Some sample computations according to Algorithm 6.3 are:

$$\begin{aligned}&\mu(\lambda; \tilde{S}(\texttt{[}\blacktriangleright\texttt{[]}\,)) = \\ &= \mu(\lambda; \tilde{S}(\blacktriangleright)\tilde{A}(\texttt{[]}\,) \cup \tilde{S}(\blacktriangleright\texttt{[}\,)\tilde{A}(\texttt{]}\,) \cup \tilde{A}(\blacktriangleright\texttt{[]}\,) \cup \tilde{S}(\blacktriangleright)\tilde{C}(\texttt{[]}\,) \cup \tilde{S}(\blacktriangleright\texttt{[}\,)\tilde{C}(\texttt{]}\,) \cup \\ &\qquad \tilde{C}(\blacktriangleright\texttt{[]}\,) \cup \langle \tilde{S}(\blacktriangleright)B(\texttt{[]}\,) \cup \tilde{S}(\blacktriangleright\texttt{[}\,)\tilde{B}(\texttt{]}\,) \cup \tilde{S}(\blacktriangleright)\tilde{D}(\texttt{[]}\,) \cup \\ &\qquad \tilde{S}(\blacktriangleright\texttt{[}\,)\tilde{D}(\texttt{]}\,) \cup \tilde{B}(\blacktriangleright\texttt{[]}\,) \cup \tilde{D}(\blacktriangleright\texttt{[]}\,)\rangle_{0.9} \cup \\ &\qquad \langle \tilde{S}(\blacktriangleright)\tilde{S}(\texttt{[]}\,) \cup \tilde{S}(\blacktriangleright\texttt{[}\,)\tilde{S}(\texttt{]}\,) \cup \tilde{S}(\blacktriangleright\texttt{[]}\,)\rangle_{0.1}) = \\ &= \ \cdots \ = \mu(\lambda; \langle\{\lambda\} \cup \langle\varnothing\rangle_{0.9} \cup \langle\varnothing\rangle_{0.1}\rangle_{0.9}) = \mu(\lambda; \{\lambda/_{0.9}\}) = 0.9\end{aligned}$$

$$\begin{aligned}&\mu(\lambda; \tilde{S}(\texttt{[[}\blacktriangleright\texttt{[}\,)) = \\ &= \mu(\lambda; \tilde{S}(\texttt{[}\,)\tilde{A}(\blacktriangleright\texttt{[}\,) \cup \tilde{S}(\texttt{[}\blacktriangleright)\tilde{A}(\texttt{[}\,) \cup \tilde{A}(\texttt{[}\blacktriangleright\texttt{[}\,) \cup \tilde{S}(\texttt{[}\,)\tilde{C}(\blacktriangleright\texttt{[}\,) \cup \tilde{S}(\texttt{[}\blacktriangleright)\tilde{C}(\texttt{[}\,) \cup \\ &\qquad \tilde{C}(\texttt{[}\blacktriangleright\texttt{[}\,) \cup \langle \tilde{S}(\texttt{[}\,)\tilde{B}(\blacktriangleright\texttt{[}\,) \cup \tilde{S}(\texttt{[}\blacktriangleright)\tilde{B}(\texttt{[}\,) \cup \tilde{S}(\texttt{[}\,)\tilde{D}(\blacktriangleright\texttt{[}\,) \cup \\ &\qquad \tilde{S}(\texttt{[}\blacktriangleright)\tilde{D}(\texttt{[}\,) \cup \tilde{B}(\texttt{[}\blacktriangleright\texttt{[}\,) \cup \tilde{D}(\texttt{[}\blacktriangleright\texttt{[}\,)\rangle_{0.9} \cup \langle \tilde{S}(\texttt{[}\,)\tilde{S}(\blacktriangleright\texttt{[}\,) \cup \\ &\qquad \tilde{S}(\texttt{[}\blacktriangleright)\tilde{S}(\texttt{[}\,) \cup \tilde{S}(\texttt{[}\blacktriangleright\texttt{[}\,)\rangle_{0.1}) = \\ &= \ \cdots \ = \mu(\lambda; \langle \tilde{S}(\texttt{[}\,)\rangle_{0.09} \cup \langle \tilde{D}(\blacktriangleright)\rangle_{0.009} \cup \langle \tilde{S}(\texttt{[}\,)\rangle_{0.09}) = \\ &= \mu(\lambda; \{\lambda/_{0.009}\} \cup \{\lambda/_{0.009}\} \cup \{\lambda/_{0.009}\}) = \mu(\lambda; \{\lambda/_{0.009}\}) = 0.009\end{aligned}$$

Note that the string `[[▶[` is obtained by three grammatical errors: one tiny mistake and two capital blunders. The algorithm finds three derivations corresponding to the "correct strings" `[][][]`, `[[]][]` and `[[][]]`. □

7 Parsing Fuzzy Context-Free Languages

In this section we show that our recognition algorithms from Sections 5–6 can be extended to parsing algorithms. In order to parse a string with respect to a given context-free grammar we need the notion of derivation tree. However, we

will use a one-dimensional representation of these trees as expressions rather than picturing them two-dimensionally.

Definition 7.1. Let $G = (V, \Sigma, P, S)$ be an $\mathcal{L}$-fuzzy context-free grammar with $N = V - \Sigma$. The fuzzy sets $\mathcal{D}_\alpha$ ($\alpha \in V$) of *derivation trees with root* α are defined as follows.

(a) For each σ in Σ, the expression σ belongs to $\mathcal{D}_\sigma$ with $\mu(\sigma; \mathcal{D}_\sigma) = 1$.

(b) For each A in N, the fuzzy set $\mathcal{D}_A$ is defined inductively by:

• If $\lambda \in P(A)$, i.e., if $\mu(\lambda; P(A)) = 1$, then the expression $A(\lambda)$ is in $\mathcal{D}_A$ with $\mu(A(\lambda); \mathcal{D}_A) = \mu(\lambda; P(A)) = 1$.

• If $\alpha_1 \cdots \alpha_n \in P(A)$ and $t_i \in \mathcal{D}_{\alpha_i}$ ($\alpha_i \in V$, $1 \leq i \leq n$), then $A(t_1, \ldots, t_n)$ belongs to $\mathcal{D}_A$ with

$$\mu(A(t_1, \ldots, t_n); \mathcal{D}_A) = \mu(\alpha_1 \cdots \alpha_n; P(A)) \star \mu(t_1; \mathcal{D}_{\alpha_1}) \star \cdots \star \mu(t_n; \mathcal{D}_{\alpha_n}).$$

(c) Finally, let $\mathcal{D}$ be defined by $\mathcal{D} = \bigcup\{\mathcal{D}_\alpha \mid \alpha \in V\}$.

The function $\Upsilon : \mathcal{D} \to \Sigma^\star$ is defined recursively by:

• For each t in $\mathcal{D}_\sigma$ ($\sigma \in \Sigma$), $\Upsilon(t) = \sigma$.

• For each $A(t_1, \ldots, t_n)$ in $\mathcal{D}_A$, $\Upsilon(A(t_1, \ldots, t_n)) = \Upsilon(t_1) \cdots \Upsilon(t_n)$.

For each t in $\mathcal{D}$, $\Upsilon(t)$ is called the *yield* of the derivation tree t. □

Corollary 7.2. *Let $G = (V, \Sigma, P, S)$ be an $\mathcal{L}$-fuzzy context-free grammar and let $\{\mathcal{D}_\alpha \mid \alpha \in V\}$ be the corresponding family of fuzzy sets of derivation trees. Then for each $x \in \Sigma^\star$, we have $\mu(x; L(G)) = \bigvee\{\mu(t; \mathcal{D}_S) \mid \Upsilon(t) = x\}$.* □

We called the elements of $\mathcal{D}$ expressions rather than terms, because symbols from N usually do not possess a single, unique airity (in case $P(A)$ contains two or more strings of different length). Remark that Definition 7.1 also applies to ordinary context-free grammars (Viz. take $\mathcal{L}$ equal to the two-element Boolean lattice $\{0, 1\}$.) and to ($\mathcal{L}$-fuzzy) context-free grammars in Chomsky normal form or in Greibach 2-form.

Example 7.3. (1) The derivation tree t with root S corresponding to derivation (1) of Example 2.2 is $t = S(A(A(a), S(B(b), B(b))), B(b))$. Note that in this expression the symbol A has airity 2 as well as 1. Clearly, we have $\Upsilon(t) = abbb$.

(2) Consider the grammar G_9 in Greibach 2-form of Example 2.9. For the string [[⫸ [we have the following derivation trees: $t_1 = S([, S([, B(⫸, S([))))$, $t_2 = S([, S([, D(⫸)), S([)))$ and $t_3 = S([, S([), B(⫸, S([)))$. Obviously, $\Upsilon(t_i) =$ [[⫸ [and $\mu(t_i; \mathcal{D}_S) = 0.009$ ($1 \leq i \leq 3$); cf. Example 6.4. In t_1 the symbol S has airity 2 and 1, in t_2 airity 3, 2 and 1, and in t_3 airity 3 and 1. □

First we modify Algorithm 5.1 into a functional parsing algorithm for fuzzy context-free grammars; cf. Algorithm 3.3. Let $\mathbb{F}(X)$ denote the power set of the fuzzy set X.

Algorithm 7.4. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Chomsky normal form and let x be a string in Σ^+.

Define the functions $\hat{f} : \Sigma^+ \to \mathcal{D}^+$ and $\hat{g} : \mathcal{D}^+ \to \mathbb{F}(\mathcal{D})$ by

$\hat{f}(a) = \{A(a) \mid a \in P(A)\}$ with $\mu(A(a); \hat{f}(a)) = \mu(a; P(A)) \quad (a \in \Sigma)$,

$\hat{f}(w) = \hat{f}(a_1)\hat{f}(a_2)\cdots\hat{f}(a_n)$ if $w = a_1a_2\cdots a_n \ \ (a_k \in \Sigma, \ \ 1 \leq k \leq n)$,

$\hat{g}(A(a)) = \{A(a)\}$ with $\mu(A(a); \hat{g}(A(a))) = 1 \quad (A \in N, \ a \in \Sigma)$,

$\hat{g}(\omega) = \bigcup\{\hat{g}(\chi)\boxtimes\hat{g}(\eta) \mid \chi, \eta \in \mathcal{D}^+, \ \omega = \chi\eta\} \qquad (\omega \in \mathcal{D}^+, \ \ |\omega| \geq 2)$ with

$\mu(t; \hat{g}(\omega)) =$

$\quad \bigvee\{\mu(t; \hat{g}(\chi)\boxtimes\hat{g}(\eta)) \mid \chi, \eta \in \mathcal{D}^+, \ \omega = \chi\eta\} \quad (t \in \mathcal{D}, \ \omega \in \mathcal{D}^+, \ |\omega| \geq 2)$,

where for X and Y in $\mathbb{F}(\mathcal{D})$ the binary operation $\boxtimes$ is defined by

$X\boxtimes Y = \{A(t_1, t_2) \mid BC \in P(A), \ t_1 \in X \cap \mathcal{D}_B, \ t_2 \in Y \cap \mathcal{D}_C, \ \ A, B, C \in N\}$,

$\mu(A(t_1, t_2); X\boxtimes Y) =$

$\quad \bigvee\{\mu(BC; P(A)) \star \mu(t_1; X \cap \mathcal{D}_B) \star \mu(t_2; Y \cap \mathcal{D}_C) \mid A, B, C \in N\}$.

Using $\hat{f}$, $\hat{g}$ and $\boxtimes$ we compute the $\mathcal{L}$-fuzzy subset $\hat{g}(\hat{f}(x))$ of $\mathcal{D}$. Then we have

- $\hat{g}(\hat{f}(x)) \cap \mathcal{D}_S$ is the $\mathcal{L}$-fuzzy set of all derivation trees of x according to G,
- $\mu(x; L(G)) = \bigvee\{\mu(t; \hat{g}(\hat{f}(x)) \cap \mathcal{D}_S) \mid \Upsilon(t) = x\}$. □

Example 7.5. We apply Algorithm 7.5 to the $\mathcal{M}$-fuzzy context-free grammar G_4 of Example 2.3 and to the input b^4. Computing $\hat{g}(\hat{f}(b^4))$ yields

$$\begin{aligned}
\hat{g}(\hat{f}(b^4)) &= \hat{g}(B(b)B(b)B(b)B(b)) = \hat{g}(B(b)B(b)B(b))\boxtimes\hat{g}(B(b))\cup \\
&\quad \hat{g}(B(b)B(b))\boxtimes\hat{g}(B(b)B(b)) \cup \hat{g}(B(b))\boxtimes\hat{g}(B(b)B(b)B(b)) = \\
&= (\hat{g}(B(b)B(b))\boxtimes\hat{g}(B(b)) \cup \hat{g}(B(b))\boxtimes\hat{g}(B(b)B(b)))\boxtimes B(b)\cup \\
&\quad (\hat{g}(B(b))\boxtimes\hat{g}(B(b)))\boxtimes(\hat{g}(B(b))\boxtimes\hat{g}(B(b)))\cup \\
&\quad B(b)\boxtimes(\hat{g}(B(b)B(b))\boxtimes\hat{g}(B(b)) \cup \hat{g}(B(b))\boxtimes\hat{g}(B(b)B(b))) = \ \cdots \ = \\
&= (\langle B(S(B(b), B(b)), B(b))\rangle_{0.9} \cup \langle B(B(b), S(B(b), B(b)))\rangle_{0.9})\boxtimes B(b)\cup \\
&\quad \varnothing \cup B(b)\boxtimes(\langle B(S(B(b), B(b)), B(b))\rangle_{0.9} \cup \langle B(B(b), S(B(b), B(b)))\rangle_{0.9}) = \\
&= \langle S(B(S(B(b), B(b)), B(b)), B(b)) \cup S(B(B(b), S(B(b), B(b))), B(b))\cup \\
&\quad S(B(b), B(S(B(b), B(b)), B(b))) \cup S(B(b), B(B(b), S(B(b), B(b))))\rangle_{0.81}.
\end{aligned}$$

So the fuzzy set $\hat{g}(\hat{f}(x)) \cap \mathcal{D}_S$ consists of the four derivation trees

$$\begin{aligned}
&S(B(S(B(b), B(b)), B(b)), B(b)), && S(B(B(b), S(B(b), B(b))), B(b)), \\
&S(B(b), B(S(B(b), B(b)), B(b))), && S(B(b), B(B(b), S(B(b), B(b)))),
\end{aligned}$$

each of which has degree of membership 0.81. Consequently, $\mu(b^4; L(G_4)) = 0.81$ which we already knew from Example 6.2. □

Next we turn to a tabular version of the CYK-algorithm for parsing $\mathcal{L}$-fuzzy context-free languages.

Algorithm 7.6. Given a λ-free $\mathcal{L}$-fuzzy context-free grammar $G = (V, \Sigma, P, S)$ in Chomsky normal form and a string $a_1 a_2 \cdots a_n$ $(n \geq 1)$ with $a_k \in \Sigma$ $(1 \leq k \leq n)$. Construct the strictly upper-triangular $(n+1) \times (n+1)$ parsing matrix M as in Figure 3, where each element $m_{i,j}$ is a finite subset of $\mathcal{D} \times \mathcal{L}$ with $N = V - \Sigma$. As usual, each $m_{i,j}$ is initially empty.

Then $m_{0,n} \cap (\mathcal{D}_S \times \mathcal{L})$ consists of all pairs (t, s) such that t is a derivation tree of G with $\Upsilon(t) = a_1 a_2 \cdots a_n$ and $\mu(t; \mathcal{D}_S) = s$. □

begin
  for $i := 0$ **to** $n - 1$ **do**
    $m_{i,i+1} := \{A(a_{i+1})/_s \mid A \in N,\ s = \mu(a_{i+1};\ P(A)) > 0\}$;
  for $d := 2$ **to** n **do**
    for $i := 0$ **to** $n - d$ **do**
      begin $j := d + i$;
        $m_{i,j} := \{A(t_1, t_2)/_s \mid A \in N,\ t_1/_p \in m_{i,k} \cap \mathcal{D}_B,\ t_2/_q \in m_{k,j} \cap \mathcal{D}_C,$
          $i + 1 \leq k \leq j - 1;\ s = p \star q \star \mu(BC, P(A)) > 0\ \}$
      end
end.

Fig. 3. Algorithm 7.6.

Example 7.7. Consider the $\mathcal{M}$-fuzzy context-free grammar G_8 in Chomsky normal form of Example 2.8 and the string [[⚡ over Σ_8.

$i \backslash j$	1	2	3
0	$\{S([)/_{0.1}$, $B([)/_1\}$	$\{S(B([), S([))/_{0.01}$, $A(B([), S([))/_{0.1}\}$	$\{S(A(B([), S([)), F(⚡))/_{0.09}$, $A(B([), S(B([), F(⚡)))/_{0.9}\}$
1		$\{S([)/_{0.1}, B([)/_1\}$	$\{S(B([), F(⚡))/_{0.9}\}$
2			$\{F(⚡)/_1\}$

Table 6
Parsing matrix for [[⚡ using Algorithm 7.6.

The initialization phase of Algorithm 7.6 yields: $m_{0,1} = \{S([)/_{0.1}, B([)/_1\}$, $m_{1,2} = \{S([)/_{0.1}, B([)/_1\}$ and $m_{2,3} = \{F(⚡)/_1\}$; the parsing matrix is in Table 6.

Now $m_{0,3} \cap (\mathcal{D}_S \times \mathcal{M}) = \{S(A(B([), S([)), F(⚡))/_{0.09}\}$, and the only derivation tree of [[⚡ is $S(A(B([), S([)), F(⚡))$, while $\mu([\ [\ ⚡; L(G_8)) = 0.09$. □

Extending Algorithms 4.2 and 6.1 to a parsing algorithm for $\mathcal{L}$-fuzzy context-free languages yields the following recursive descent parsing algorithm.

Algorithm 7.8. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Chomsky normal form and let w be a string in Σ^+. To each nonterminal symbol A in N, we associate a function $\hat{A} : \Sigma^\star \to \mathbb{F}(\mathcal{D}))$ defined as follows.

If the argument x of $\hat{A}$ is a word of length 1 (i.e., x is in Σ), then

$\hat{A}(x) = \{A(x) \mid x \in P(A)\}$ with $\mu(A(x); \hat{A}(x)) = \mu(x; P(A))$ $\qquad (x \in \Sigma)$,

and in case the length $|x|$ of the word x is 2 or more, then

$\hat{A}(x) = \{A(t_1, t_2) \mid t_1 \in \hat{B}(y),\ t_2 \in \hat{C}(z),\ BC \in P(A),\ y, z \in \Sigma^+,\ x = yz\}$,

$\mu(A(t_1, t_2); \hat{A}(x)) =$

$\quad \bigvee\{\mu(t_1; \hat{B}(y)) \star \mu(t_2; \hat{C}(z)) \star \mu(BC; P(A)) \mid x = yz,\ y, z \in \Sigma^+\}$.

Using these functions, we compute the fuzzy subset $\hat{S}(w)$ of $\mathcal{D}$. Then $\hat{S}(w) = \{t \in \mathcal{D}_S \mid \Upsilon(t) = w\}$ and $\mu(w; L(G)) = \bigvee\{\mu(t; \hat{S}(w)) \mid t \in \mathcal{D}_S,\ \Upsilon(t) = w\}$. □

Example 7.9. We apply Algorithm 7.8 to the $\mathcal{M}$-fuzzy context-free grammar G_4 of Example 2.3 and to the input b^4.

$\hat{S}(b^4) = \{S(t_1, t_2) \mid t_1 \in \hat{A}(b^3), t_2 \in \hat{B}(b)\} \cup \{S(t_1, t_2) \mid t_1 \in \hat{A}(b^2), t_2 \in \hat{B}(b^2)\} \cup$
$\{S(t_1, t_2) \mid t_1 \in \hat{A}(b), t_2 \in \hat{B}(b^3)\} \cup \{S(t_1, t_2) \mid t_1 \in \hat{B}(b^3), t_2 \in \hat{A}(b)\} \cup$
$\{S(t_1, t_2) \mid t_1 \in \hat{B}(b^2), t_2 \in \hat{A}(b^2)\} \cup \{S(t_1, t_2) \mid t_1 \in \hat{B}(b), t_2 \in \hat{A}(b^3)\} \cup$
$\{S(t_1, t_2) \mid t_1 \in \hat{A}(b^3), t_2 \in \hat{A}(b)\}/_{0.1} \cup \{S(t_1, t_2) \mid t_1 \in \hat{A}(b^2), t_2 \in \hat{A}(b^2)\}/_{0.1} \cup$
$\{S(t_1, t_2) \mid t_1 \in \hat{A}(b), t_2 \in \hat{A}(b^3)\}/_{0.1} \cup \{S(t_1, t_2) \mid t_1 \in \hat{B}(b^3), t_2 \in \hat{B}(b)\}/_{0.9} \cup$
$\{S(t_1, t_2) \mid t_1 \in \hat{B}(b^2), t_2 \in \hat{B}(b^2)\}/_{0.9} \cup \{S(t_1, t_2) \mid t_1 \in \hat{B}(b), t_2 \in \hat{B}(b^3)\}/_{0.9}$.

Since $\hat{A}(b) = \varnothing$ and $\hat{B}(b) = \{B(b)\}$, we need $\hat{A}(b^2)$, $\hat{A}(b^3)$, $\hat{B}(b^2)$, and $\hat{B}(b^3)$. Using $\hat{S}(b) = \varnothing$, we obtain by some subcomputations that $\hat{A}(b^2) = \hat{A}(b^3) = \hat{B}(b^2) = \varnothing$, and $\hat{B}(b^3) = \{B(S(B(b), B(b)), B(b)), B(B(b), S(B(b), B(b)))\}/_{0.9}$. Then $\hat{S}(b^4)$ reduces to

$\{S(t_1, t_2) \mid t_1 \in \hat{B}(b^3), t_2 \in \hat{B}(b)\}/_{0.9} \cup \{S(t_1, t_2) \mid t_1 \in \hat{B}(b), t_2 \in \hat{B}(b^3)\}/_{0.9} =$
$= \{S(B(S(B(b), B(b)), B(b)), B(b)), S(B(B(b), S(B(b), B(b))), B(b)),$
$\quad S(B(b), B(S(B(b), B(b)), B(b))), S(B(b), B(B(b), S(B(b), B(b))))\}/_{0.81}$.

So we obtain four derivation trees together with their degree of membership as in Example 7.5. □

Finally, we provide a robust parsing algorithm for λ-free $\mathcal{L}$-fuzzy context-free grammars in Greibach 2-form; cf. Algorithms 4.4 and 6.3.

Algorithm 7.10. Let $G = (V, \Sigma, P, S)$ be a λ-free $\mathcal{L}$-fuzzy context-free grammar in Greibach 2-form and let w be a string in Σ^+. To each nonterminal symbol A in N we associate a function $\hat{A} : \Sigma^\star \to \mathbb{F}(\mathcal{D}))$ defined as follows.

If the argument x of $\hat{A}$ is a word of length 1 (i.e., x is in Σ) then

$\hat{A}(x) = \{A(x) \mid x \in P(A)\}$ with $\mu(A(x); \hat{A}(x)) = \mu(x; P(A))$ $\qquad (x \in \Sigma)$,

and in case the length $|x|$ of the word x is 2 or more, then

$\hat{A}(x) = \{A(a, t_1, t_2) \mid x = ayz,\ a \in \Sigma,\ y, z \in \Sigma^+,\ aBC \in P(A),\ t_1 \in \hat{B}(y),$
$\quad t_2 \in \hat{C}(z)\} \cup \{A(a, t_1) \mid x = ay,\ a \in \Sigma,\ y \in \Sigma^+,\ t_1 \in \hat{B}(y),\ aB \in P(A)\}$,

$$\mu(A(a,t_1,t_2);\hat{A}(x)) = \bigvee\{\mu(t_1;\hat{B}(y)) \star \mu(t_2;\hat{C}(z)) \star \mu(aBC;P(A)) \mid a \in \Sigma, \\ B,C \in N,\ x = ayz,\ y,z \in \Sigma^+\}$$

$$\mu(A(a,t_1);\hat{A}(x)) = \bigvee\{\mu(t_1;\hat{B}(y)) \star \mu(aB;P(A)) \mid a \in \Sigma,\ B \in N,\ x = ay, \\ y \in \Sigma^+\}.$$

As usual, we compute the fuzzy subset $\hat{S}(w)$ of $\mathcal{D}$. Then we have that $\hat{S}(w) = \{t \in \mathcal{D}_S \mid \Upsilon(t) = w\}$ and $\mu(w;L(G)) = \bigvee\{\mu(t;\hat{S}(w)) \mid t \in \mathcal{D}_S,\ \Upsilon(t) = w\}$. □

Example 7.11. Consider the $\mathcal{M}$-fuzzy context-free grammar of Example 2.9. Since G is in Greibach 2-form, we may apply Algorithm 7.10. For input `[[⋗[` we obtain

$$\begin{aligned}
\hat{S}(\texttt{[[⋗[}) = {} & \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{A}(\texttt{[})\} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{A}(\texttt{⋗[})\} \cup \\
& \{S(\texttt{[},t) \mid t \in \hat{A}(\texttt{[⋗[})\} \cup \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{C}(\texttt{[})\} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{C}(\texttt{⋗[})\} \cup \{S(\texttt{[},t) \mid t \in \hat{C}(\texttt{[⋗[})\} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{B}(\texttt{[})\}/_{0.9} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{B}(\texttt{⋗[})\}/_{0.9} \cup \{S(\texttt{[},t) \mid t \in \hat{B}(\texttt{[⋗[})\}/_{0.9} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{D}(\texttt{[})\}/_{0.9} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{D}(\texttt{⋗[})\}/_{0.9} \cup \{S(\texttt{[},t) \mid t \in \hat{D}(\texttt{[⋗[})\}/_{0.9} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{S}(\texttt{[})\}/_{0.1} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{S}(\texttt{⋗[})\}/_{0.1} \cup \{S(\texttt{[},t) \mid t \in \hat{S}(\texttt{[⋗[})\}/_{0.1}.
\end{aligned}$$

Since $\hat{A}(\texttt{[}) = \hat{A}(\texttt{⋗[}) = \hat{A}(\texttt{[⋗[}) = \hat{C}(\texttt{[}) = \hat{C}(\texttt{⋗[}) = \hat{C}(\texttt{[⋗[}) = \hat{B}(\texttt{[}) = \hat{B}(\texttt{[⋗[}) = \hat{D}(\texttt{[}) = \hat{D}(\texttt{⋗[}) = \hat{D}(\texttt{[⋗[}) = \hat{S}(\texttt{⋗[}) = \varnothing$, this reduces to

$$\begin{aligned}
\hat{S}(\texttt{[[⋗[}) = {} & \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[}), t_2 \in \hat{B}(\texttt{⋗[})\}/_{0.9} \cup \\
& \{S(\texttt{[},t_1,t_2) \mid t_1 \in \hat{S}(\texttt{[⋗}), t_2 \in \hat{S}(\texttt{[})\}/_{0.1} \cup \{S(\texttt{[},t) \mid t \in \hat{S}(\texttt{[⋗[})\}/_{0.1}.
\end{aligned}$$

Computing the other sets yield $\hat{S}(\texttt{[}) = \{S(\texttt{[})\}/_{0.1}$, $\hat{S}(\texttt{[⋗}) = \{S(\texttt{[},D(\texttt{⋗}))\}/_{0.9}$, $\hat{B}(\texttt{⋗[}) = \{B(\texttt{⋗},S(\texttt{[}))\}/_{0.1}$ and $\hat{S}(\texttt{[⋗[}) = \{S(\texttt{[},B(\texttt{⋗},S(\texttt{[})))\}/_{0.09}$.

Substituting these results in the expression for $\hat{S}(\texttt{[[⋗[})$ yields a set $\hat{S}(\texttt{[[⋗[})$ with three elements: viz. $S(\texttt{[},S(\texttt{[}),B(\texttt{⋗},S(\texttt{[}))))$, $S(\texttt{[},S(\texttt{[},D(\texttt{⋗})),S(\texttt{[}))$, and $S(\texttt{[},S(\texttt{[},B(\texttt{⋗},S(\texttt{[})))))$, each of which has 0.009 as degree of membership.

Remember that `[[⋗[` has been obtained by three grammatical errors; viz. by one tiny mistake and two capital blunders. Therefore the degree of membership of each derivation tree is $0.9 \star 0.1 \star 0.1 = 0.009$; cf. Example 6.4. □

8 Concluding Remarks

In Sections 3 and 4 we considered functional versions of the CYK-algorithm and of some recursive descent recognizers. These functional versions have been obtained by removing details that refer to possible implementations or to data structures. For a functional version of Earley's algorithm we refer to [19].

Although some of these functional algorithms are rather inefficient, they are still worth to be studied. Firstly, they may serve as a basis for a general approach to recognition and parsing algorithms for context-free languages [24]. Secondly, these functional algorithms are good starting points ("prototypes") for designing more efficient algorithms based on dynamic programming, parallelism or on a systolic approach [21,22]. Particularly, Algorithms 4.2 and 4.4 are inefficient; they can be implemented efficiently in many different ways; cf. e.g., the divide-and-conquer approach in [11]. Since the "calls" of $\tilde{B}(y)$ and $\tilde{C}(z)$ in (12) are mutually independent, a parallel implementation (e.g. on a parallel random access machine [23]) is a suitable choice. Due to the fact that the context-free grammar is λ-free, the total number of recursive calls during the computation of $\tilde{S}(w)$ is at least $2 \cdot |w| - 1$ for Algorithm 4.2 and at least $|w|$ for Algorithm 4.4.

All our algorithms are based on Chomsky normal form or Greibach 2-form. Of course, one can drop these conditions, e.g., it is possible to generalize the CYK-algorithm to arbitrary context-free grammars [2]. And instead of the Greibach 2-form, we may use the Greibach normal form or the super normal form of [20]. But the price we have to pay is that the resulting algorithms become more complicated, less transparent, and more difficult to analyze.

The algorithms in Sections 3–4 served as starting point to develop recognizers and parsers for fuzzy context-free languages in Sections 5–7. Obviously, the complexity of these algorithms increases when we add modifications to deal with the degree of membership. So the ultimate complexity of the algorithms in Sections 5–7 heavily depends on the costs of the "arithmetic" in the codomain $\mathcal{L}$ of the membership functions. If we restrict ourselves to the cases $\mathcal{L} = \mathcal{I}$ or $\mathcal{L} = \mathcal{M}$ (Section 2), the change in the complexity will not be dramatic, particularly when a restricted accuracy in $\mathcal{L}$ is sufficient; cf. the rôle of the thresholds δ and Δ (Section 1).

These thresholds can also be used to prune the recognition or parsing process: adding the condition "$r \star p \star q > \delta$" or even "$r \star p \star q \geq \Delta$" in computing $t_{i,j}$ in Algorithm 5.4, speeds up the computation but we loose some less desirable derivations (corresponding to "capital blunders"). Clearly, this approach can be applied to other algorithms in Sections 5–7 as well.

Our algorithms in Section 5–7 are robust in the very simple sense that they are able to report grammatical errors and the extend in which these errors affected the input string. To correct these grammatical errors is quite a different and much more difficult problem. Repairing a wrong terminal symbol (as in Examples 2.5 and 2.6: writing $[\, S \not\rightarrow S$ instead of $[\, S]\, S$) is rather easy, the deletion of a terminal symbol causes more problems (Examples 2.5 and 2.6: writing $[\, SS$ rather than $[\, S]\, S$), but recovery from the deletion or incorrect substitution of a nonterminal symbol (Example 2.2) is anything but straightforward.

In this paper we used in our examples either $\mathcal{I}$ or $\mathcal{M}$ as codomain of the membership functions. Since both $\mathcal{I}$ and $\mathcal{M}$ are linearly ordered, we will briefly sketch what might happen when the codomain $\mathcal{L}$ is not linearly ordered.

Example 8.1. Cf. [17] and also Examples 3.3.(1) and 6.4.(2) in [8]. Let $\mathcal{L}$ be the 4-element distributive lattice that is not a linearly ordered set, i.e, $\mathcal{L} = (\{0, \xi, \eta, 1\}, \wedge, \vee, 0, 1, \wedge)$ with $0 < \xi < 1$, $0 < \eta < 1$, and ξ and η are incomparable.

Consider the following $\mathcal{L}$-fuzzy context-free grammar $G_{11} = (V_{11}, \Sigma_{11}, P_{11}, S)$ with $\Sigma_{11} = \{a, b\}$, $V_{11} - \Sigma_{11} = \{S, A, B, C, D, \overline{D}, E, \overline{E}\}$ and P_{11}, viewed as a λ-free nested fuzzy substitution, is defined by

$$
\begin{array}{ll}
P_{11}(S) = \{S, CD/_\xi, EC/_\eta\}, & P_{11}(C) = \{C, AC, a\}, \\
P_{11}(D) = \{D, BA, \overline{D}A\}, & P_{11}(\overline{D}) = \{\overline{D}, BD\}, \\
P_{11}(E) = \{E, AB, A\overline{E}\}, & P_{11}(\overline{E}) = \{\overline{E}, EB\}, \\
P_{11}(A) = \{A, a\}, & P_{11}(B) = \{B, b\}, \\
P_{11}(a) = \{a\}, & P_{11}(b) = \{b\}.
\end{array}
$$

The crisp part $c(L(G_{11}))$ of $L(G_{11})$ satisfies $c(L(G_{11})) = \{a^n b^n a^n \mid n \geq 1\}$, and for the fuzzy language $L(G_{11})$ we have

$$
\begin{aligned}
L(G_{11}) = c(L(G_{11})) \cup \{a^m b^n a^n \mid m, n \geq 1,\ m \neq n\}/_\xi \cup \\
\{a^n b^n a^m \mid m, n \geq 1,\ m \neq n\}/_\eta.
\end{aligned}
$$

Applying any of the recognition algorithms from Section 5–6 yields: for each n ($n \geq 1$), $\mu(a^n b^n a^n; L(G_{11})) = 1$, although there does not exists a completely correct derivation for $a^n b^n a^n$ according to G_{11} (i.e., a derivation without grammatical errors).

Now parsing a string $a^n b^n a^n$ ($n \geq 1$) is much more revealing. Viz. when we apply, for instance, Algorithm 7.4 or 7.6, we obtain sets of the form $\{t_1/_\xi, t_2/_\eta\}$ or, equivalently, $\{(t_1, \xi), (t_2, \eta)\}$. So we get two different derivation trees t_i ($i = 1, 2$) with different, incomparable degrees of membership. Since $\Upsilon(t_i) = a^n b^n a^n$ and $\xi \vee \eta = 1$, we have $\mu(a^n b^n a^n; L(G_{11})) = 1$ as the recognition algorithms showed. □

The phenomenon showed in this example ("strings without a completely correct derivation, that still do possess a degree of membership equal to 1") partially explains the popularity of linearly ordered codomains for membership functions.

Finally, we recall that we used fuzzy context-free grammars (at a syntactical level) to model the effect of grammatical errors in relation to robustness in recognizing and parsing. Of course, it is also possible to add fuzziness (at a semantical level) to a crisp context-free grammar in order to incorporate "vagueness" or "uncertainty" in natural language processing. An example of this latter approach can be found in [25].

References

[1] A.V. Aho & J.D. Ullman: *The Theory of Parsing, Translation and Compiling — Volume I: Parsing* (1972), Prentice-Hall, Englewood Cliffs, NJ.

[2] H.J.A. op den Akker: Recognition methods for context-free languages (1991), Memoranda Informatica 91-30, Dept. of Comp. Sci., Twente University of Technology, Enschede, the Netherlands.

[3] P.R.J. Asveld: An alternative formulation of Cocke–Younger–Kasami's algorithm, *Bull. Europ. Assoc. for Theor. Comp. Sci.* (1994) no. 53, 213–216.

[4] P.R.J. Asveld: Towards robustness in parsing — Fuzzifying context-free language recognition, pp. 443–453 in: J. Dassow, G. Rozenberg & A. Salomaa (eds.): *Developments in Language Theory II — At the Crossroads of Mathematics, Computer Science and Biology* (1996), World Scientific, Singapore.

[5] P.R.J. Asveld: A fuzzy approach to erroneous inputs in context-free language recognition, pp. 14–25 in: *Proc. 4th Internat. Workshop on Parsing Technologies* (1995), Prague / Karlovy Vary, Czech Republic.

[6] P.R.J. Asveld: Controlled fuzzy parallel rewriting, pp. 49–70 in: Gh. Păun & A. Salomaa (eds.): *New Trends in Formal Languages — Control, Cooperation, and Combinatorics* (1997), Lect. Notes in Comp. Sci. **1218**, Springer, Berlin, etc.

[7] P.R.J. Asveld: The non-self-embedding property for generalized fuzzy context-free grammars, *Publ. Math. Debrecen* **54** *Suppl.* (1999) 553–573.

[8] P.R.J. Asveld: Algebraic aspects of families of fuzzy languages, *Theor. Comp. Sci.* **293** (2003) 417–445.

[9] P.R.J. Asveld: Fuzzy context-free languages — Part 1: Generalized fuzzy context-free grammars, *Theor. Comp. Sci.* **347** (2005) 167–190.

[10] J. Berstel: *Transductions and Context-Free Languages* (1979), Teubner, Stuttgart.

[11] A. Bossi, N. Cocco & L. Colussi: A divide-and-conquer approach to general context-free parsing, *Inform. Process. Lett.* **16** (1983) 203–208.

[12] G. Gerla: Fuzzy grammars and recursively enumerable fuzzy languages, *Inform. Sci.* **60** (1992) 137–143.

[13] S. Ginsburg: *Algebraic and Automata-Theoretic Properties of Formal Languages* (1975), North-Holland, Amsterdam.

[14] M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.

[15] J.E. Hopcroft & J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation* (1979), Addison-Wesley, Reading, Mass.

[16] H.H. Kim, M. Mizumoto, J. Toyoda & K. Tanaka: L-fuzzy grammars, *Inform. Sci.* **8** (1975) 123–140.

[17] M. Mizumoto, J. Toyoda & K. Tanaka, Examples of formal grammars with weights, *Inform. Process. Lett.* **2** (1973) 74–78.

[18] E.T. Lee & L.A. Zadeh: Note on fuzzy languages, *Inform. Sci.* **1** (1969) 421–434.

[19] R. Leermakers: A recursive ascent Earley parser, *Inform. Process. Lett.* **41** (1992) 87–91.

[20] H.A. Maurer, A. Salomaa & D. Wood: A supernormal-form theorem for context-free grammars, *J. Assoc. Comp. Mach.* **30** (1983) 95–102.

[21] A. Nijholt: Overview of parallel parsing strategies, Chapter 14 (pp. 207–229) in M. Tomita (ed.): *Current Issues in Parsing Technology* (1991), Kluwer, Boston.

[22] A. Nijholt: The CYK-approach to serial and parallel parsing, *Proc. Seoul Internat. Conf. on Natural Language Processing SICONLP'90* (1990) 144–155.

[23] W.J. Savitch & M.J. Stimson: Time bounded random access machines with parallel processing, *J. Assoc. Comp. Mach.* **26** (1979) 103–118.

[24] K. Sikkel: *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms* (1997), Springer-Verlag, Berlin, etc.

[25] F. Wang: A fuzzy grammar and possibility theory-based natural language use interface for spatial queries, *Fuzzy Sets and Systems* **113** (2000) 147–159.