# On Families of Categorial Grammars of Bounded Value, Their Learnability and Related Complexity Questions

Christophe Costa Florêncio[a], Henning Fernau[b]

[a] *University of Amsterdam, Informatics Institute, ILPS, Amsterdam, the Netherlands*
[b] *Universität Trier, FB IV—Abteilung Informatik, D-54286 Trier, Germany*

**Abstract**

In [1], the learnability of several parameterized families of categorial grammar classes was studied. These classes were shown to be learnable in the technical sense of identifiability in the limit from positive data. They are defined in terms of bounds on parameters of the grammars which intuitively correspond to restrictions on linguistic aspects, such as the amount of lexical ambiguity.

The time complexity of learning these classes has been studied in [2]. It was shown that, for most of these classes, selecting a grammar from the class that is consistent with given data is NP-hard. In this paper existing complexity results are sharpened by demonstrating W[2]-hardness. Additionally, parameters are defined that allow FPT-results; roughly, this implies that if these parameters are fixed, these problems become tractable.

We also define the new family $\mathcal{G}_{k\text{-sum-val}}$, which is natural from the viewpoints of Parameterized Complexity, a flourishing area of Complexity Theory (see [3]) and from Descriptional Complexity, a sub-area of Formal Language Theory (see [4]). We prove its learnability, analyze its relation to other classes from the literature and prove a hierarchy theorem.

This approach is then generalized to a parameterized family defined in terms of a bound on the descriptional complexity expressed as a Hölder norm. We show that both the hierarchy result and the property of finite elasticity (and thus learnability) are preserved under this generalization.

*Keywords:* grammar induction, identification in the limit, finite elasticity, parameterized complexity, categorial grammar, regular tree languages, Hölder norms

## 1. Introduction

We consider the complexity of consistency problems for some family $(\mathcal{L}_k)$ of language classes ordered by inclusion, which is of the following form: Given a

*Email addresses:* C.CostaFlorencio@uva.nl (Christophe Costa Florêncio), fernau@uni-trier.de (Henning Fernau)

finite language sample $D$ and some parameter $k$, is there some language $L \supseteq D$ contained in $\mathcal{L}_k$?

This question shows up quite naturally in the context of (consistent) learning from text (details can be found below); the corresponding learning algorithms are often expressed as starting from a finite set $D$ of input strings, and the very first task is to find a grammar from the target class that is consistent with $D$.

For many families of language classes, this type of consistency problem is trivial. Consider for example the class $\mathcal{A}_k$ of languages that can be accepted by a finite automaton with at most $k$ states. In this case we can always answer YES to the consistency problem, since an automaton exists that accepts $\Sigma^*$.[1]

However, this trivial type of reply is no longer possible if the universal language (i.e., $\Sigma^*$ in the case of string languages) is not an element of each of the language classes of interest. Examples are provided by classes of classical categorial grammars; see [1].

The language families $\mathcal{L}_k$ are usually defined via grammar families $\mathcal{G}_k$. As a variant of the mentioned consistency problem, we may be given a finite set of derivation structures and some parameter $k$ and ask if there is a grammar $G \in \mathcal{G}_k$ that produces those structures (and possibly more). Note that this can also be seen as a special case of the first problem formulation, if we consider *languages of structures* (formally, labeled ordered trees).

We study the computational complexity of consistency problems both from a classical (P vs. NP) perspective, as well as from the perspective of parameterized complexity. As far as the authors are aware, we have obtained the first such results for grammar induction problems and, more generally, learning problems in the identification in the limit paradigm.[2]

Apart from these complexity results, we also introduce a new family of categorial grammars, based on the so-called sum-value of a categorial grammar (to be defined in the next section) which is interesting in its own right, since it constitutes a new strict hierarchy of categorial language classes, where each level is learnable from positive examples. We generalize this and other classes to a parameterized family defined in terms of a bound on the descriptional complexity expressed as a Hölder norm. Using elementary mathematical properties of Hilbert spaces, we demonstrate that both the hierarchy result and the property of finite elasticity (and thus learnability) are preserved under this generalization. Such a 'transfer' result has been hitherto unknown for the setting of learning from positive data.

Thus the new results presented in this paper fall within and bridge three areas of research:

[1] Such an automaton consists of just one state, which is accepting, and a transition for every element in $\Sigma$ which loops back to this state.

[2] The paper [5] deals strictly with the parameterized complexity of several flavors of exact learning of CNF/DNF formulas and certain types of graph learning problems.

Another relevant paper, [6], appeared after the publication of our first results in [7]. Among other things, it demonstrates W[1]-hardness of the consistency problem for the non-erasing pattern languages.

- Complexity Theory (more specifically, both classical aspects and aspects of Parameterized Complexity),
- Formal Languages (in particular, touching the sub-areas of Descriptional Complexity and classical questions as hierarchy theorems and algorithmic questions),
- and Learning Theory (primarily, Inductive Inference and Grammatical Inference).

Moreover, since categorial grammars are mainly studied within computational and mathematical linguistics, our results may be especially relevant for researchers from these fields.

In order to keep the paper as self-contained as possible, we will try to provide the necessary background of all the relevant fields. Readers familiar with this material can of course skip these sections.

Throughout the paper, we use standard notation from Formal Language Theory. For example, $\Sigma^*$ is the set of all words over the alphabet $\Sigma$, including the empty word $\varepsilon$. Any element from $\Sigma^*$ is called a *word* or a *string*. The length of a word $w$ is denoted by $|w|$. If $D$ is a finite language, then $\|D\|$ denotes the sum of all lengths of all words from $D$.

*Previous paper versions.* A part of this paper appeared in [7], covering only parameterized complexity results for the consistency problem of the family of (structure) languages of categorial grammars of bounded max-value. The hierarchy result and the results concerning the Hölder norm bound approach were first presented in [8]. Due to space restrictions, many proofs and claims were omitted in these conference papers.

## 2. Categorial Grammars—Definitions and Examples

The classes studied in [9, 10] which are also the focus of the present paper are based on a formalism for ($\varepsilon$-free) context-free languages called *classical categorial grammar* (CCG). In this section the relevant concepts of CCG will be defined.[3] We will use the same notation as found in [1].

In CCG, each *symbol* (or *atom*) in some given alphabet $\Sigma$ is assigned a finite number of *types*. In the remainder, we assume $\Sigma$ to be fixed. This is technically convenient, and makes no difference in the context of learning, since only the subset of $\Sigma$ that actually appears in the data is relevant for the learner. Types are constructed from *primitive types* by the operators $\backslash$ and $/$. We let Pr denote the (countably infinite) set of primitive types. The set of types Tp is defined as the smallest set satisfying:

1. $\mathrm{Pr} \subseteq \mathrm{Tp}$,

[3] It is worth mentioning that the first paper on CCG [11] considerably pre-dates those on Chomsky-type grammars which are much better known in Computer Science.

2. if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $A\backslash B \in \mathrm{Tp}$,
3. if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $B/A \in \mathrm{Tp}$.

One member $t$ of Pr is called the *distinguished type*, and is considered a constant. In CCG there are only two modes of type combination, *backward application*, $A, A\backslash B \Rightarrow B$, and *forward application*, $B/A, A \Rightarrow B$. In both cases, type $A$ is the *argument*, the complex type is the *functor*.[4] Given an expression of the form $A/B$ ($B\backslash A$), its *main operator* is '/' ('\'). *Grammars* consist of type assignments to symbols, i.e., $\texttt{symbol} \mapsto T$, where $\texttt{symbol} \in \Sigma$ and $T \in \mathrm{Tp}$. A CCG $G$ can be hence viewed as a mapping from $\Sigma$ into finite subsets of types Tp.

**Def. 1** *A* derivation *of $B$ from $A_1, \ldots, A_n$ is a binary branching, labeled tree that encodes a proof of $A_1, \ldots, A_n \Rightarrow B$.*

Let $\mathrm{Tp}(G)$ denote the set of types that occur in a CCG $G$. A type from $\mathrm{Tp}(G)$ is called *useless* if it does not appear in any derivation of $G$. A grammar is called *reduced* if it contains no useless types, see [1]. The types appearing in the definition of a CCG are also called the *range* of the grammar. This should not be confused with the range of a type: the range of $A/B$ ($B\backslash A$) is $A$, its *domain* is $B$.

Through the notion of derivation the association between grammar and language is defined. All structures contained in some given structure language correspond to a derivation of type $t$ based solely on the type assignments contained in a given grammar. This is the *structure language* generated by $G$, denoted $\mathrm{FL}(G)$. The *string language* generated by $G$, $\mathrm{L}(G)$, consists of the strings corresponding to all the structures in its structure language, where the string corresponding to some derivation consists just of the leaves of that derivation (also known as the *yields*).

The symbol FL is an abbreviation of *functor-argument language*, the derivation language generated by a CCG that is obtained by suppressing types associated to inner nodes in the derivation (tree). Hence, structures correspond to terms. More precisely, structures are of the form `symbol`, `fa(s1,s2)` or `ba(s1,s2)`, where $\texttt{symbol} \in \Sigma$, `fa` stands for forward application, `ba` for backward application and `s1` and `s2` are also structures. The set of all functor-argument structures over the alphabet $\Sigma$ is denoted by $\Sigma^F$.

Note that structure languages as defined above are in fact *regular tree languages*. Thus the issues discussed in this paper could be relevant to the topic of learning (regular) tree languages, but we will not pursue this further here.

**Ex. 2** *The top left structure in Fig. 1 is a derivation for a proof of*

$$np, np\backslash(t/np), np/n, n/n, n \Rightarrow t.$$

[4] According to an alternative convention that is sometimes found in the literature, $A\backslash B$ is used where we write $B\backslash A$. We follow the original notation from [12].

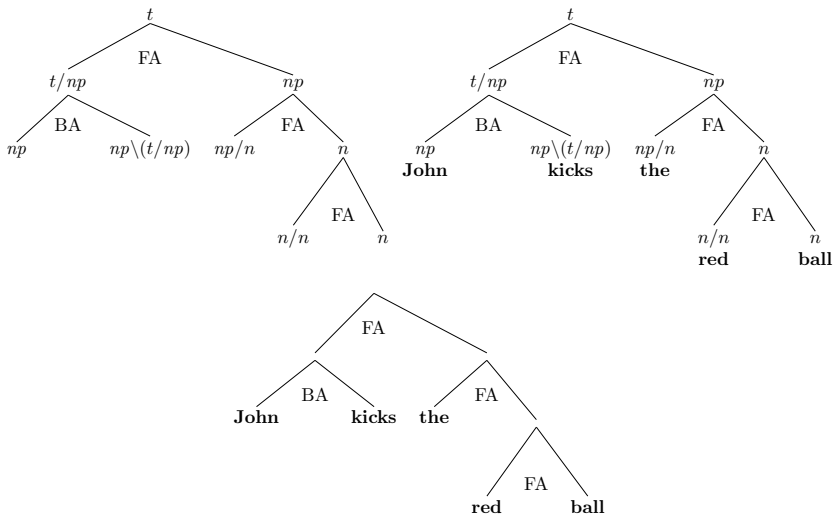


Figure 1: Parsing with categorial grammars.

*Note that all tree nodes carry types as labels, and that to inner nodes, in addition, labels* BA *and* FA *are associated, which indicate the operations applied at those points in the derivation.*
*The top right structure shows a CCG parse, which is a derivation where the leaves are labeled not just with types, but also with the lexical items (such as* **John***) that these types are assigned to in the grammar used for the parse. At the bottom, the corresponding functor-argument structure is shown.*

Classical categorial grammar can be extended to (various variants of) *general combinatory* grammars by introducing additional operators such as Forward Composition ($A/B, B/C \Rightarrow A/C$) or Forward Crossing Composition ($A/B, C\backslash B \Rightarrow C\backslash A$). These are linguistically motivated, and allow both a more flexible interface between surface structure and semantics, and extend the expressive power to mild context-sensitivity. Refer to [13, 14] for an overview.[5]

All learning functions in [1] are based on the function GF. This function receives a sample of structures $D$ as input and yields a set of assignments (i.e., a grammar) called the *general form* as output, which generates exactly $D$. It is a homomorphism and runs in linear time. GF assigns $t$ to each root node, assigns distinct variables to the argument nodes, and computes types for the functor nodes: if it is the case that `s1` $\mapsto A$, given `ba(s1,s2)` $\Rightarrow B$, then `s2` $\mapsto A\backslash B$. If `s1` $\mapsto A$, given `fa(s2,s1)` $\Rightarrow B$, then `s2` $\mapsto B/A$. When learning from strings,

[5] In addition to the classical/combinatorial approach, there also exists a type-logical approach to categorial grammar. This falls outside the scope of this paper, the interested reader is referred to [15] for a survey, and to [16, 17] for learnability results for classes of type-logical categorial grammars.

the structure language is not available to the learner, but given a set of strings there exist only finitely many possible sets of structures for the classes under discussion. These are then used to produce hypotheses.

Categorial types can be treated as terms, so natural definitions of substitution and unification apply. A substitution over a grammar is just a substitution over all of the types contained in its assignments. This notion can be used to unify distinct types assigned to the same word. Consider, for example, the following grammar, which the reader can verify generates just the words `ab`, `ba` and `babb`:

$$G = \begin{array}{lcl} \texttt{a} & \mapsto & t/A, B\backslash t, C/(E/E), (C/D)/D \\ \texttt{b} & \mapsto & A, B, t/C, D, (E/E)/D \end{array}$$

The types $t/A$ and $C/(E/E)$ can be unified to yield the single type $t/(E/E)$, obtained by applying the most general unifier $\sigma = \{C \leftarrow t, A \leftarrow (E/E)\}$. The type $B\backslash t$ cannot be unified with any of the other types, since they all have '/' as main operator, while $B\backslash t$ has '\' as main operator. The types $t/A$ and $(C/D)/D$ cannot be unified because their functors are a constant and a complex type, respectively. Finally, $C/(E/E)$ and $(C/D)/D$ cannot be unified because this would fail the so-called *occurs check*: it would require that $C$ is unified with $C/D$, which would result in a cyclic term.

We state without proof that $\mathrm{FL}(G) \subseteq \mathrm{FL}(\sigma[G])$ for each substitution $\sigma$, see [1] for details.

A CCG $G$ can be viewed as a mapping from $\Sigma$ into finite subsets of types Tp. Accordingly, we can associate a *value function* $v_G$ that maps $a \in \Sigma$ onto $|G(a)|$, i.e., the number of types that $G$ maps to $a$. If clear from context, the index $G$ will be suppressed. As discussed in [18], at least two natural size measures can be derived from $v$, depending on the chosen metric: the *max-value* of $G$ is $\max_{a\in\Sigma} v(a)$, while the *sum-value* of $G$ is $\sum_{a\in\Sigma} v(a)$. A categorial grammar $G$ is called *$k$-max-valued* (*$k$-sum-valued*, respectively) if its max-value (sum-value, respectively) is upper-bounded by $k$. The according grammar classes are denoted by $\mathcal{G}_{k\text{-max-val}}$ and $\mathcal{G}_{k\text{-sum-val}}$, resp. In the literature [2, 1], $k$-max-valued grammars are also known as $k$-valued grammars, and 1-max-valued grammars are also known as *rigid grammars*, and denoted as $\mathcal{G}_{\text{rigid}}$. This class is known to be learnable from structures with polynomial update-time, by simply unifying all types assigned to the same symbol in the general form ([1]). The other classes originally defined in [9, 10] are generalizations of this class. The $k$-sum-valued class has not been studied before.

The languages that can be described with $k$-max-valued ($k$-sum-valued, resp.) grammars are comprised in the classes $\mathcal{L}_{k\text{-max-val}}$ ($\mathcal{L}_{k\text{-sum-val}}$, resp). The structure languages that can be generated by grammars from $\mathcal{G}_{k\text{-max-val}}$ and $\mathcal{G}_{k\text{-sum-val}}$, resp., are written $\mathcal{FL}_{k\text{-max-val}}$ and $\mathcal{FL}_{k\text{-sum-val}}$, resp.

One more class discussed in [1] that we need to define is that of the least cardinality grammars, denoted by $\mathcal{G}_{\text{least-card}}$. A grammar of *least cardinality with respect to* $L$ is a partial function from $\Sigma$ to $\mathrm{Tp}^+$ such that $L \subseteq \mathrm{FL}(G)$ and there is no grammar $G'$ such that $\sum_{a\in\Sigma} v_{G'}(a) < \sum_{a\in\Sigma} v_G(a)$ and $L \subseteq \mathrm{FL}(G')$.

The following two propositions concern the relations between $\mathcal{G}_{k\text{-sum-val}}$ and the other classes. They follow directly from the definitions:

**Prop. 3** *In the case that* $k = |\Sigma|$, $\mathcal{G}_{k\text{-}sum\text{-}val} \supseteq \mathcal{G}_{rigid}$.

Observe that a grammar that assigns zero types to $a$ and two types to $b$ is 2-sum-valued, but not rigid.

**Prop. 4** *If a given sample* $D$ *of a structure language is not a subset of any language in* $\mathcal{FL}_{(k-1)\text{-}sum\text{-}val}$*, but is a subset of some language in* $\mathcal{FL}_{k\text{-}sum\text{-}val}$*, then* $\{G \mid G \in \mathcal{G}_{k\text{-}sum\text{-}val}, D \in \mathrm{FL}(G)\} = \{G \mid G \text{ is of least cardinality w.r.t. } D\}$.

For the intuition behind these definitions, we quote Buszkowski and Penn [10]:

> Following the general methodological principle of *economy*, we look for a simplest possible description of the initial data; simplicity may be interpreted here as minimization of the number of different types assigned to each atom.

## 3. Learning from Positive Samples

In [19] the notion of *identification in the limit* was introduced. In this model a learning function sees an infinite sequence which enumerates the target language, called a *text*, and hypothesizes a grammar for the target language at each time-step. A *class* of languages is called *learnable* if and only if there exists a learning function that, after a finite number of presentations, guesses the right language on every text for every language from that class and does not deviate from this hypothesis.

Research within this framework is known as *(formal) Learning Theory*. Only those aspects of Learning Theory that are relevant to our result will be discussed, see [20] for a comprehensive overview of the field.

Many different variants of the paradigm have been studied, amongst them the learning of *indexed families of computable languages*, i.e., learning in the presence of a uniform decision procedure for languages in the class (see [21]). This is a natural assumption when studying grammar induction, and we will assume it in the remainder of this paper.

Let the set $\Omega$ denote the hypothesis space, which can be any class of finitary objects. In the context of this paper, members of $\Omega$ are called *grammars*. The set $\mathbf{S}$ denotes the sample space, a recursive subset of $\Sigma^*$ or of $\Sigma^F$ for some fixed finite alphabet $\Sigma$. Elements of $\mathbf{S}$ are called *words* in the first case, subsets of $\mathbf{S}$ are then called *languages*. Elements of $\mathbf{S}$ are called *structures* in the second case, subsets of $\mathbf{S}$ are then called *structure languages*. The *naming function* L maps elements of $\Omega$ to subsets of $\mathbf{S}$. If $G$ is a grammar in $\Omega$, then $\mathrm{L}(G)$ is called the *language generated by (associated with) G*. The *universal membership problem* takes as input a word and a grammar and asks if that word belongs to the language generated by the given grammar.

A triple $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$, with $\Omega$, $\mathbf{S}$ and L as described above, is called a *grammar system*.[6] A class of grammars is generally denoted by $\mathcal{G}$, a class of languages is denoted by $\mathcal{L}$.

The class of all categorial grammars is denoted CatG, the grammar systems under discussion are hence $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL} \rangle$ and $\langle \mathrm{CatG}, \Sigma^*, \mathrm{L} \rangle$.

We will adopt notation from [1] and let $\mathcal{FL}$ denote a class of *structure languages*. The corresponding naming function is $\mathrm{FL}(G)$. Learning functions are written as $\varphi$.

The setting of learning from structures is closely related to learning from *meaning*, i.e., from strings together with their semantic types. This is outside the scope of this paper, the interested reader is referred to [23, 24].

Identifiability in the limit of a class guarantees the existence of a learning algorithm, but the learning problem is not necessarily tractable. In order to obtain a usable algorithm, further constraints on the learner need to be specified. Ideally we would use some notion of polynomial identification in the limit. There are several around, but they are all of a somewhat ad-hoc nature. Yokomori [25] defines a class as *polynomial-time identifiable in the limit from positive data* if a learning algorithm for that class exists such that both the number of explicit errors of prediction and the computation time it needs for any data sequence are bounded by polynomials over the complexity of the representation.

Obviously, this kind of definition implies that it must be possible to produce a hypothesis that is consistent with the data (and belongs to a given class) in polynomial time, this property is known as *polynomial update-time*. There seems to be some consensus that this is an important property for efficient learnability, but see [26, 27]. Given the class $C$, we will denote this problem as $C$-CONSISTENCY.

Note, however, that NP-completeness of $C$-CONSISTENCY does not exclude the existence of a learning algorithm for $C$ that is in some sense efficient. It is theoretically possible that in such a case there exists a learning algorithm that runs in polynomial time whenever it converges on the data set, and trades efficiency for consistency before convergence.

A language class $\mathcal{L}$ is said to have *infinite elasticity* (see [28, 29, 30]) if there exists an infinite sequence $(w_n)$ of words and an infinite sequence $(L_n)$ of languages in $\mathcal{L}$ such that, for all $n \in \mathbb{N}$, $w_n \notin L_n$, but $\{w_1, \ldots, w_n\} \subseteq L_{n+1}$. A language class $\mathcal{L}$ has *finite elasticity* if it does not possess infinite elasticity. If $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ is a grammar system and $\mathcal{G}$ is a recursively enumerable subset of grammars from $\Omega$, then the family of languages $L(\mathcal{G}) = \{L(G) \mid G \in \mathcal{G}\}$ is learnable (identifiable in the limit) if it has finite elasticity.

The language families that we defined in the preceding section are particularly interesting from the point of view of Learning Theory. They provide non-trivial examples of language families with finite elasticity, so that we can state:

[6] This notion should not be confused by that of a grammar system as defined in [22].

**Thm. 5** *Families from both the hierarchies $\mathcal{L}_{k\text{-}sum\text{-}val}$ and $\mathcal{FL}_{k\text{-}sum\text{-}val}$ have finite elasticity.*

**Proof.** The maximum number of types that can be assigned to any symbol in a $k$-max-valued grammar is $k$. Thus, $\mathcal{G}_{k\text{-sum-val}}(\Sigma) \subset \mathcal{G}_{k\text{-max-val}}(\Sigma)$, and $\mathcal{L}_{k\text{-sum-val}}(\Sigma) \subseteq \mathcal{L}_{k\text{-max-val}}(\Sigma)$.

Since $\mathcal{L}_{k\text{-max-val}}$ is known to have finite elasticity for every $k$, every subclass, including $\mathcal{L}_{k\text{-sum-val}}$, has finite elasticity (for every $k$), and is thus learnable.

An analogous argument can be given for $\mathcal{FL}_{k\text{-sum-val}}$. □

**Cor. 6** *Both families from the hierarchies $\mathcal{L}_{k\text{-}sum\text{-}val}$ and $\mathcal{FL}_{k\text{-}sum\text{-}val}$ are identifiable in the limit.*

## 4. Formal Language Results

The following hierarchy result was demonstrated in [1] (Theorem 5.5):

**Thm. 7** *For any $k \geq 1$, $\mathcal{L}_{k\text{-}max\text{-}val} \subsetneq \mathcal{L}_{(k+1)\text{-}max\text{-}val}$.*

The analogous result for $\mathcal{FL}_{k\text{-max-val}}$ follows directly from the definition of the class.

We complement this result by showing:

**Thm. 8** *For any $k \geq 1$, $\mathcal{L}_{k\text{-}sum\text{-}val} \subsetneq \mathcal{L}_{(k+1)\text{-}sum\text{-}val}$.*

Note that the proof of this theorem is quite unusual, since it heavily relies on the finite elasticity property that was originally defined for Learning Theory purposes.

**Proof.** This proof somewhat resembles the proof of the hierarchy theorem for $\mathcal{L}_{k\text{-max-val}}$. It is possible to provide a simple proof based on the relationship between $\mathcal{L}_{k\text{-sum-val}}$ and $\mathcal{L}_{k\text{-max-val}}$, but we prefer to give a constructive proof since it provides more insight and proves a hierarchy theorem for the corresponding class of structure languages as well.

By definition, $\mathcal{G}_{k\text{-sum-val}} \subset \mathcal{G}_{(k+1)\text{-sum-val}}$, which immediately implies the inclusion $\mathcal{L}_{k\text{-sum-val}} \subseteq \mathcal{L}_{(k+1)\text{-sum-val}}$. It remains to show its strictness, i.e., that $\mathcal{L}_{(k+1)\text{-sum-val}} - \mathcal{L}_{k\text{-sum-val}} \neq \emptyset$.

In the case that $k = 2$, the grammar $\mathtt{a} \mapsto A, t/A$ is in $\mathcal{G}_{k\text{-sum-val}}$, and thus the language $\{\mathtt{aa}\}$ is in $\mathcal{L}_{k\text{-sum-val}}$. In the case that $k = 1$, only grammars of the form $\mathtt{a} \mapsto t$ are in the class, so only languages of the form $\{\mathtt{a}\}$ are in $\mathcal{L}_{k\text{-sum-val}}$. This demonstrates that $\mathcal{L}_{(k+1)\text{-sum-val}} - \mathcal{L}_{k\text{-sum-val}} \neq \emptyset$ for $k = 1$.

It remains to show that $\mathcal{L}_{(k+1)\text{-sum-val}} - \mathcal{L}_{k\text{-sum-val}} \neq \emptyset$ for $k \geq 2$. Let $L_n = \{a^i \mid 1 \leq i \leq n\}$, and $\mathcal{L}_{iac} = \{L_n \mid n \in \mathbb{N}^+\}$. Note that $\mathcal{L}_{iac}$ is an infinite ascending chain, so, since $\mathcal{L}_{k\text{-sum-val}}$ has finite elasticity (Theorem 5), for any fixed $k$, $\mathcal{L}_{iac} \not\subseteq \mathcal{L}_{k\text{-sum-val}}$. Thus, there exists an $n \in \mathbb{N}$ such that $L_n \notin \mathcal{L}_{k\text{-sum-val}}$. We will show that for the least such $n$, $L_n \in \mathcal{L}_{(k+1)\text{-sum-val}}$.

Let $G_n$ be a $k$-sum-valued grammar such that $\mathrm{L}(G_n) = L_n$.[7] There must exist a type $A \in \mathrm{range}(G_n) - \mathrm{Pr}$ such that $A$ is not a proper subtype of any type in $\mathrm{range}(G_n)$. Let $B = (\ldots(t/\underbrace{A)/\ldots)/A}_{n}$, and let $G_{n+1} = G_n \cup \{\langle a, B\rangle\}$.

We will now show that $\mathrm{L}(G_n) = L_n$.

The range of $B$ can only be $t$. Since $B$ is not the proper subtype of any type in $\mathrm{range}(G_{n+1})$, the only derivation that can yield type $B$ consists of just a leaf, and this type is assigned to $\mathtt{a}$ in $G_n$. Therefore, there exists a derivation proving $B, \underbrace{A \ldots A}_{n} \Rightarrow t$, and $G_{n+1}$ allows a CCG parse corresponding to this derivation. The yield of the parse is $a^{n+1}$, and since $B$ is not a proper subtype of any type in the range of $G_{n+1}$, the only other derivations that $G_{n+1}$ allows are exactly the derivations that $G_n$ allows. Thus $\mathrm{L}(G_{n+1}) = \mathrm{L}(G_n) \cup \{a^{n+1}\} = L_n$. □

From this proof it immediately follows that:

**Thm. 9** *For any $k \geq 1$, $\mathcal{FL}_{k\text{-}sum\text{-}val} \subsetneq \mathcal{FL}_{(k+1)\text{-}sum\text{-}val}$.*

## 5. Hölder Norms

The bounds defining the $k$-max-valued and $k$-sum-valued classes are in fact special cases of *Hölder norms*. These are of the form $\| \vec{x} \|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$ for $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $p \geq 1$. For $p = 1$, this is the sum norm bound, for $p \to \infty$ this is the maximum norm bound, and for $p = 2$ we obviously obtain the Euclidian distance bound.

**Prop. 10** *[Folklore]*

1. $\| \vec{x} \|_\infty \leq \| \vec{x} \|_p$ *for any* $p \geq 1$.
2. *For 1-dimensional spaces,* $\| \vec{x} \|_\infty = \| \vec{x} \|_p$ *for any* $p \geq 1$.

Thus, a natural question presents itself: do learnability and hierarchy theorems hold for every class of CCGs defined in terms of Hölder norms? Let $\mathcal{G}_{H(k,p)}$ collect those grammars $G$ which satisfy $\| \vec{v}(G) \|_p \leq k$. We write $\mathcal{L}_{H(k,p)}$ and $\mathcal{FL}_{H(k,p)}$ for the corresponding string- and structure languages. Recall that we assume the alphabet $\Sigma$ to be fixed. By definition, we have:

**Lem. 11** *For every $k, p \geq 1$, $\mathcal{G}_{H(k,p)} \subseteq \mathcal{G}_{H(k+1,p)}$.*

Proposition 10 (1) entails, completely analogous to Theorem 5:

**Thm. 12** *Let $p \geq 1$. Families from both the hierarchies $\mathcal{L}_{H(k,p)}$ and $\mathcal{FL}_{H(k,p)}$ have finite elasticity.*

**Thm. 13** *For every $k, p \geq 1$, $\mathcal{L}_{H(k,p)} \subsetneq \mathcal{L}_{H(k+1,p)}$.*

[7] Note that when we write $G_n$, $n$ is simply an index, not necessarily the value of $k$.

**Proof.** From Lemma 11, it immediately follows that $\mathcal{L}_{H(k,p)} \subseteq \mathcal{L}_{H(k+1,p)}$. Since the strictness (for the case $p = 1$) was shown in Theorem 8 by a unary example, based on the finite elasticity of that class, Proposition 10 (2), together with Theorem 12 yield the claim. □

From this proof, it immediately follows that:

**Thm. 14** *For every* $k, p \geq 1$, $\mathcal{FL}_{H(k,p)} \subsetneq \mathcal{FL}_{H(k+1,p)}$.

More generally, if $\mathcal{L}_k$ now denotes any language class that has finite elasticity and whose definition is obtained by restricting the $L_\infty$ norm of some vector $\vec{v}(G)$ associated to some grammar $G$ for $L \in \mathcal{L}_k$, then, for any $p \geq 1$, $\mathcal{L}_{k,p}$ is a hierarchy with finite elasticity, as well, where $\mathcal{L}_{k,p}$ consists of all languages that can be described by grammars $G$ with $\parallel \vec{G} \parallel_p \leq k$.

This might allow to identify the language class that comes closest to one's needs for, say, linguistic reasons, as all these classes enjoy similar positive learnability properties.

## 6. Complexity Notions

We assume some familiarity with the basic notions of classical complexity on the side of the reader.

There has been recent interest in the development of parameterized complexity results to allow for a more fine-grained analysis of NP-hard problems. So, a problem (parameterized by $k$) is in FPT if we can develop an algorithm with running time $O(f(k)p(n))$, where $n$ is the overall input size and $k$ is the (size of the) parameter. $f$ is an arbitrary function (depending on $k$ but independent of $n$) and $p$ is a polynomial. An algorithm that proves FPT-ness is also called an FPT-*algorithm*, or a *parameterized algorithm*. More details can be found in the monograph [3].

The classical example for a problem in FPT is the VERTEX COVER problem on undirected graphs. So, given a graph $G$ and a parameter $k$, the question is whether a *vertex cover* $C$ exists with $|C| \leq k$, where a vertex cover set can be characterized by the fact that, upon removing it together with all incident edges, no edges will remain in the graph. One simple FPT-algorithm is a branching algorithm that picks an edge $\{x, y\}$, as long as this is possible and as long as the parameter value $k$ is bigger than zero, and branches into the two possibilities of covering that edge. In each of the branches, one more vertex is taken into the proposed partial cover, the graph is modified by deleting that cover-vertex from the instance (together with its incident edges), and the parameter value is decremented by one. If this binary search tree arrives at a graph with no edges, then a cover with at most $k$ vertices was found on that path of the search tree and this can be accordingly reported. If all branches of the search tree are cut because the parameter value was decreased down to zero and no solution was found so far, then we face a NO-instance. In that (worst) case, the search tree will have $2^k$ leaves and twice as many nodes in total. Since all work that arises in the nodes can be done in polynomial time, this is an FPT-algorithm.

It is worth mentioning that the very first analysis that is, to our knowledge, published on the sketched simple search tree algorithm was in the algorithms textbook of Mehlhorn [31], predating any other publication on parameterized algorithms. It has become the standard example, including the discussion of several improvements, in all textbooks on Parameterized Complexity [3, 32, 33]. The current record-holder has brought the basis of the expression estimating the size of the search tree (and hence the running time) down to less than 1.28, see [34].

This approach makes sense in particular if the parameter of interest can be assumed to be small. The hierarchy level $k$ in our formulation of the consistency problem might be such a small parameter: in linguistics, the amount of lexical ambiguity for natural language is assumed to be very small in relation to the number of symbols found in any realistic lexicon. We thus arrive at problems that we call, for instance, $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY in order to make the parameter explicit. However, we cannot always hope to find nice parameterized algorithms. More specifically, we can derive as a corollary from the fact that $\mathcal{L}_{1\text{-VALUED}}$-CONSISTENCY is NP-hard ([2, Theorem 5.32] or alternatively, [35])):

**Cor. 15** *Unless* P = NP*, there is no* FPT *algorithm that decides* $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY*.*

**Proof.** If this were not the case, there would be an algorithm that decides the consistency problem in time $O(f(k)p(n))$. Setting $k = 1$, we would arrive at a polynomial-time algorithm that decides if, given a finite set $D$ of strings, there exists a 1-max-valued (rigid) categorial grammar $G$ such that $D \subseteq \mathrm{L}(G)$. This problem is known to be NP-hard. □

This corollary is surely disappointing from a parameterized point of view, since it seems to rule out the use of the natural parameter $k$ as a good choice of a parameter for the consistency problem for $k$-valued categorial grammars.

The proof of NP-hardness of $\mathcal{L}_{1\text{-VALUED}}$-CONSISTENCY makes use of the fact that there is no bound on the size of the alphabet, so we still might hope for better parameterized results when we restrict our attention to languages over alphabets of size three, for example.

As we will see, this hope will not be fulfilled. However, in order to formulate and establish the indicated result, we need some more notions from parameterized complexity.

As with classical complexity, we need an appropriate notion of reduction to prove hardness results, and some knowledge about classes of parameterized problems that are believed not to possess FPT-algorithms. Actually, there is a whole hierarchy of parameterized problems that is believed to be strict, the so-called W-hierarchy. Usually, its lowest level, W[0], is called FPT (which we have already defined).

The W-hierarchy is usually defined in terms of circuits. However, the lowest three levels of this hierarchy are the most important ones for this paper, and they can be characterized by halting problems of Turing machines.

Namely, if a parameterized problem (with parameter $k$) is in W[1], then it can be solved by a nondeterministic one-tape Turing machine within $f(k)$ steps for some function $f$. A typical W[1]-complete problem is the following one: given a nondeterministic one-tape Turing machine $M$ and a parameter $k$, does $M$ halt within $k$ steps (when starting on the empty tape)? There are also many graph-theoretic problems that belong to this class. For example, recall that $C$ is a minimum vertex cover in a graph $G = (V, E)$ iff the subgraph induced by $V \setminus C$ is an *independent set* (also known as a *stable set*) of $G$, i.e., a set of vertices having no edge between any pair of them. This relation is also known as the Gallai identity, see [36]. So, the question of finding an independent vertex set $I$ of size $k$ is intimately related to the question of finding a vertex cover $C$ of size $k'$. However, while the latter problem is in FPT, the first question is W[1]-complete.

If a parameterized problem (with parameter $k$) is in W[2], then it can be solved by a nondeterministic multi-tape Turing machine within $f(k)$ steps for some function $f$; it is important that this Turing machine has parallel access to its working tapes. A typical W[2]-complete problem is the following one: given a nondeterministic multi-tape Turing machine $M$ and a parameter $k$, does $M$ halt within $k$ steps (when starting on the empty tape)? Another problem related to VERTEX COVER in graphs is the analogous question for hypergraphs. This problem is also known as HITTING SET: given a hypergraph $G$ and a parameter $k$, we ask for a vertex set $C$ of size at most $k$ such that each edge $e$ of $G$ is hit, i.e., $e \cap C \neq \emptyset$ (note that a hyperedge $e$ is simply the set of vertices it connects). HITTING SET is known to be W[2]-complete.

We still need a satisfying notion of reduction in order to define hardness (and completeness) for parameterized complexity classes. Given two parameterized problems $P$ and $P'$ with parameterizations $k$ and $k'$, resp., a *parameterized (many-one) reduction* translates an instance $(I, k)$ of $P$ in polynomial time into an instance $(I', k')$ such that $k' = f(k)$ for some function $f$. Obviously, if $P'$ is in $W[i]$, $i = 0, 1, 2$, then so is $P$. So, if $P$ is W[2]-hard and we can provide such a reduction that translates $P$ into $P'$, then $P'$ is W[2]-hard, as well. As an example, consider the reductions presented in Sections 5.3-5.5 from [2] that show NP-hardness of several variants of 'consistency problems'. These reductions use VERTEX COVER, and the reductions are actually parameterized reductions in the following sense: they show how an instance $(G, k)$ of VERTEX COVER can be transformed in polynomial time into an instance $(F, k)$ of $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY. The fact that the same parameter appears in both problems is one of the reasons we originally hoped for FPT-ness results for this type of problem.

The hunt for this type of results can be also seen under a broader perspective. For decades, the P-Cognition thesis held that cognitive capacities are limited to those functions that can be computed in polynomial time. More recently, the FPT-Cognition thesis has been proposed as an alternative. In short, this means that one would concede more computing power to the brain and other computational cognitive devices to cope with certain problems, thereby widening the territory of tractability. Detailed discussions on this matter can be found

in [37, 38]. Naturally, learning is one of the basic cognitive tasks and hence fits into these discussions.

## 7. Complexity Results

### *7.1. k-Maximum-Value Problems*

In [2], only hardness results were shown. We first complement these results by demonstrating membership in NP. This was originally neglected, since the consistency problem was studied as an aspect of learning problems, specifically of identification in the limit. In this paradigm, the length of the input sequence before convergence is inherently unbounded. Thus, it makes little sense to consider questions such as membership in NP, which would require a polynomial number of steps before convergence.

Lemmas 6.1 and 6.2 from [1] (attributed to Buszkowski and Penn) underline the importance of the concept of the general form $\mathrm{GF}(D)$ (as discussed above), a CCG associated to a finite structure language $D$. Without giving details here, note that it is further known that any reduced CCG $G'$ consistent with $D$ can be obtained from $\mathrm{GF}(D)$ by unification (implied by Prop 6.35 from [1]). Moreover, the size of $\mathrm{GF}(D)$ (and hence the size of $G'$) is bounded by a polynomial over $\|D\|$. If $D$ is a finite string language, then any finite structure language $D'$ that yields $D$ is of size polynomial in $\|D\|$. Hence, any $\mathrm{GF}(D')$ of interest is also of size polynomial in $\|D\|$.

**Thm. 16** $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY *is* NP*-complete (for every fixed* $\Sigma$ *with* $|\Sigma| \geq 3$*).*

**Proof.** NP-hardness of $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is shown in [2] (Theorem 5.16, which holds for the case where $|\Sigma| = 3$ and any larger alphabet). To see membership in NP, let $(D, k)$ be an instance of $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY. The nondeterministic procedure we propose first generates some $G \in \mathcal{G}_{k\text{-max-val}}$, by unifying types in $\mathrm{GF}(D)$ that are assigned to the same symbol. Then, for each structure $s \in D$, the procedure tests whether $s \in \mathrm{FL}(G)$ (which can be done in polynomial time). If (and only if) all these tests are passed, the algorithm returns YES. □

**Thm. 17** $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY *is* NP*-complete. Here, the alphabet* $\Sigma$ *is not fixed.*

**Proof.** NP-hardness of $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY is shown in [2] (Theorem 5.32, which holds for the case where $k = 1$ and $|\Sigma|$ is unbounded). To see membership in NP, let $(D, k)$ be an instance of $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY. The nondeterministic procedure we propose consists of two parts; first, for each string in $D$, a derivation is chosen. Then, the union of the resulting structures, $D'$ (note that $\|D'\|$ is obviously polynomial in $\|D\|$), is used as a sample for learning from structures, so that some $G \in \mathcal{G}_{k\text{-max-val}}$ is generated by unifying types in $\mathrm{GF}(D')$ that are assigned to the same symbol. Then, for each string

$w \in D$, the procedure tests whether $w \in \mathrm{L}(G)$ (which can be done in polynomial time). If (and only if) all these tests are passed, the algorithm returns YES. □

We can actually sharpen the hardness assertion in the sense of parameterized complexity, by defining a polynomial-time transformation from HITTING SET to a dataset for a language in $\mathcal{FL}_{k\text{-max-val}}$. Deciding that the data is consistent with a language in that class, i.e., $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY, then corresponds to deciding the existence of a cover of a specified size $c$. We will call any grammar that generates a language in the class consistent with the given input a *consistent grammar*.

Since the construction is somewhat involved, an example is included in Appendix Appendix A.

*7.2. Explanation of the construction*

The construction makes use of several techniques that we now discuss informally to make things more accessible.

**Padding**. At certain places in the construction we may want types that are assigned to the same symbol in the general form to be unified in the consistent grammar. To this end, elements can be added to $D$ such that extra types are assigned to the same symbol that are pairwise non-unifiable, and not unifiable to the types to be unified. This can be achieved by assigning the right (in this case, depending on $k$) number of such types (which we call padding types).

**Multiple occurrences of the same primitive type**. At other places in the construction we may need multiple occurrences of the same primitive type. Padding cannot be used directly, since primitive types can often be unified with complex types, and thus with the padding types, which is undesirable. Instead, for primitive type $P$ assigned to type `p`, we can replace all occurrences of `p` in a given sample $D$ with, say, $\mathtt{fa}(\mathtt{p}, \mathtt{x})$. We thus have (compound) types $P_i/X_i$ assigned to `p` in $\mathrm{GF}(D)$, which makes the use of padding types possible.

**Excluding all but one / Selection gadget**. For our construction we need at some points to exclude all types but one that are assigned to the same symbol to be unified with some other type.

Let $n$ distinct types $T_i$, some type $U$, and $n$ padding types be assigned to `t`. Given a sufficiently small $k$, at least one type $T_i$ has to be unified with $U$ in order to obtain a consistent grammar that is in $\mathcal{G}_{k\text{-max-val}}$. Let all types $T_i$ and $U$ be extended as follows: let $\{\Gamma_1, \ldots, \Gamma_n\}$ be a set of pairwise non-unifiable types, and let $T_i' = T_i/X_{1,i}/\cdots/X_{n,i}$, [8] where all $X$s are primitive types, except $X_{i,i} = \Gamma_i$. Let $U' = U/Y_1/\cdots/Y_n$, where the $Y_i$s are multiple occurrences of the same primitive type.

Now, let some $T_i'$ be unified with $U'$. This will yield a substitution such that $\Gamma_i$ is substituted for $Y_i$ and thus for all $Y_1, \ldots, Y_n$.

This immediately excludes the unification of any other $T_j'$ with $U'$, since that would require unifying $\Gamma_j$ with $\Gamma_i$.

[8] Here, we assume left-associativity of / to simplify our formulas.

**Overlapping types**. In our construction we use an 'overlapping types' approach to exclude exactly one arbitrary step out of $n$ possible unification steps. Let $n$ types $T_i$, as well some type $U$, and $k-2$ padding types, be assigned to $\mathtt{t}$.

In the case that $n \geq 2$, at least $n-1$ of the types $T_i$ have to be unified with $U$. In the case that we want this to be *exactly* $n-1$ types, i.e., exclude one arbitrary of these types from unification with $U$, we can extend both all $T_i$ and $U$ in the following way: let $T'_i = T_i/X_{1,i}/\cdots/X_{n+1,i}$, where all $X$s are pairwise distinct primitive types, except for $X_{i+1,i}$ and $X_{i,i}$, which are the same primitive type. The fact that these two domain types are the same causes an 'overlap'. Let $U' = U/Y_1/\cdots/Y_{n+1}$, where all $Y$s are distinct primitive types, except for $Y_1$ and $Y_{n+1}$, which should form a non-unifiable pair and thus cannot be both primitive types. Now, let $T'_1$ be unified with $U'$. Since $X_{2,1} = X_{1,1}$, $Y_2 = Y_1$. It should be clear that sequentially unifying types $T'_2, T'_3 \ldots$ will yield a substitution $\sigma$ such that $\sigma[Y_1] = \sigma[Y_2] = \cdots = \sigma[Y_{n+1}]$. Since $Y_1$ cannot be unified with $Y_{n+1}$, it follows that it is not possible to unify all types $T'_1, T'_2, \ldots$ with $U'$. However, unifying all types of this form, save one single $T'_i$, with $U'$ is possible, since the resulting substitution $\sigma$ need not be such that $\sigma[X_{i+1,i}] = \sigma[X_{i,i}]$, and hence it is not required that $\sigma[Y_{i+1,i}] = \sigma[Y_{i,i}]$, which, by definition, is not possible.

*7.3. The construction*

We now define a construction that is based on these ideas.

**Def. 18** *Let $hg(HG, c)$ be the algorithm that maps instances of the vertex cover problem for hypergraphs to samples of structure languages defined in the following way:*

*The hypergraph $HG = (V, E)$ consists of a set $V$ of vertices numbered $1, \ldots, v$ and a set $E$ of (hyper)edges numbered $1, \ldots, e$. It is characterized by the functions $d(i)$, $1 \leq i \leq e$, which gives the degree (or size) of edge $E_i$, i.e., the number of vertices that belong to that edge, and $n(i, j)$, which gives the index of the $j$th vertex that edge $i$ is incident on. The constant $c$ specifies the maximal size of the cover.*

*The sample $D$ output by $hg$ is the smallest set that fulfills the following requirements:*

*For each $i$, $1 \leq i \leq e$, the structures*
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, \underbrace{\mathtt{fa}(\mathtt{c}_i, \mathtt{x})) \ldots \mathtt{fa}(\mathtt{c}_i, \mathtt{x}))}_{d(i)\ times}, \underbrace{\mathtt{u}_i), \ldots \mathtt{u}_i)}_{d(i)+1\ times}$ *and*
$\mathtt{fa}(\mathtt{fa}(\mathtt{fa}(\mathtt{fa}(\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, \mathtt{f}_{(i,1)}), \ldots \mathtt{f}_{(i,d(i))}), \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{s}_{(i,1)}), \ldots \mathtt{s}_{(i,d(i)-1)}), \mathtt{fa}(\mathtt{ttt}, \mathtt{t}))$
*are in $D$. Additionally, for each $j$, $1 \leq j \leq d(i)$,*
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_i, G_{(j,1)}), \ldots G_{(j,d(i))}), T_{(j,1)}), \ldots T_{(j,d(i)+1)})$ *where*

$$G_{(j,x)} = \begin{cases} \mathtt{fa}(\mathtt{v}_{n(i,x)}, \mathtt{x}) & \text{if } x = j \text{ and} \\ \mathtt{g}_{(i,x)}, & \text{otherwise} \end{cases}$$

*where*

$$T_{(j,x)} = \begin{cases} \mathtt{fa}(\mathtt{t}_{j,j}, \mathtt{x}) & \text{if } x = j \vee x = j+1 \text{ and} \\ \mathtt{t}_{(j,x)}, & \text{otherwise.} \end{cases}$$

*For each* $i$, $1 \leq i \leq e$, $\mathtt{fa}(\mathtt{c}, \mathtt{fa}(\mathtt{c}_i, \mathtt{x}))$ *is in* $D$.
*If* $c = 1$, *the padding structure* $\mathtt{ba}(\mathtt{x}, \mathtt{c})$ *is in* $D$.
*For each* $i$, $1 \leq i \leq c$, *the padding structure* $\mathtt{ba}(\mathtt{x}, \mathtt{c}_i)$ *is in* $D$.
*For each* $i$, $1 \leq i \leq v$, *the padding structure* $\mathtt{ba}(\mathtt{x}, \mathtt{v}_i)$ *is in* $D$.
*For each* $\mathtt{t}_{i,j}$, *the padding structure* $\mathtt{ba}(\mathtt{x}, \mathtt{t}_{i,j})$ *is in* $D$.
*We add* $k-2$ *padding structures for each* $\mathtt{e}_i$, *and* $k-1$ *such structures for each* $\mathtt{v}_i$, $\mathtt{c}_i$, *and for* `tt` *and* `ttt`.

In order to make clear why the sample is built up in this way, we now discuss the types as they occur in any grammar in $\mathcal{G}_{k\text{-max-val}}$ that is consistent with this sample. For the sake of readability, we first introduce shorthand notation in the form of $\Upsilon$, $\Gamma$ and $\Sigma$, which represent complex types that will occur in the general form (types written as capital letters are primitive types, unless otherwise stated). Let $1 \leq i \leq e$ in the following.

Let $\Upsilon_i = C_{i,1}/\cdots/C_{i,v}/(U_{s(1,i,0)}/U_{s(1,i,1)})/\cdots/(U_{s(d(i),i,0)}/U_{s(d(i),i,1)})$, where the $C$ and $U$ types are primitive types.

Define $\Sigma_i = F_{i,1}/\cdots/F_{i,v}/(S_{g(1,i,0)}/S_{g(1,i,1)})/\cdots/(S_{g(v,i,0)}/S_{g(v,i,1)})$. Here, the $F$ and $S$ types are primitive types.

Let $\Gamma_{n(i,j)} = G_{i,1}/\cdots/G_{i,v}/\ (T_{t(1,i,0)}/T_{t(1,i,1)})/\cdots/(T_{t(d(i),i,0)}/T_{t(d(i),i,1)})$ for $1 \leq j \leq d(i)$, $T_{t(k,i,0)} = T_{t(\ell,i,1)}$ if $\ell = n(i,j)$, and the $T_{t(\ell,i,j)}$s are distinct (primitive) types, otherwise.

Every type $G_{i,j}$, in the case that $i = j$, is equal to some type $\Delta_{n(i,j)}$.[9] The types $\Delta_1, \Delta_2, \ldots$ are based strictly on alternating forward- and backward slashes, with the main operator always the backward slash. The type $\Delta_1$ is $X\backslash t$. For any two $u, v$ such that $u \neq v$, $\Delta_u$ and $\Delta_v$ are not unifiable. Note that this allows any two $\Gamma_{n(i,x)}$ and $\Gamma_{n(i,y)}$, $x \neq y$, to be unifiable, since the $\Delta$-subtypes appear in different positions of these $\Gamma$ terms.

From GF($D$), the grammar shown in Fig. 2 is derived by unifying all types assigned to `x`. This simplifies the presentation without affecting the proof of W[2]-hardness in any way. Note that in the interest of clarity we omit type assignments to symbols $\mathtt{f}_{n(x,y)}$, $\mathtt{g}_{n(x,y)}$, $\mathtt{s}_{x,y}$, $\mathtt{t}_{x,y}$ and $\mathtt{u}_x$.

Note that *hg* runs in time polynomial in the size of the hypergraph. There are bounds on the parameters of the grammar: given hypergraph $HG = (V, E)$ and stipulated size of the cover $c$, then $k = \max(2, c)$, and

$$|\Sigma| = 5 + |V| + 2|E| + 2\sum_{i=1}^{|E|} d(i) + 2\sum_{i=1}^{|E|} d(i)^2.$$

The construction works just for $k \geq 2$, but this does not affect the result in any way. Note that for $k = 1$, the consistency problem is known to be solvable in polynomial time. Similarly, HITTING SET is solvable in polynomial time with parameter $k = 1$. So, it would be possible to slightly modify *hg* as

[9] Recall that function $n(i, j)$ yields the index of the $j$th vertex that edge $i$ is incident on. Also note that, otherwise, $G_{i,j}$ is a primitive type.

$$
\begin{array}{lll}
 & \mathtt{e}_1 & \mapsto \quad t/\Upsilon_1, \\
 & & \quad t/\Gamma_{n(1,1)}, \ldots, t/\Gamma_{n(1,d(1))}, \\
 & & \quad t/\Sigma_1, \\
 & & \quad \mathit{Padding} \\
 & \ldots & \\
 & \mathtt{e}_e & \mapsto \quad t/\Upsilon_e, \\
 & & \quad t/\Gamma_{n(e,1)}, \ldots, t/\Gamma_{n(e,d(e))}, \\
 & & \quad t/\Sigma_e, \\
 & & \quad \mathit{Padding} \\
 & \mathtt{v}_1 & \mapsto \quad \Delta_1, \mathit{Padding} \\
G' : & \ldots & \\
 & \mathtt{v}_v & \mapsto \quad \Delta_v, \mathit{Padding} \\
 & \mathtt{c}_1 & \mapsto \quad C_{1,1}/X, \ldots, C_{1,v}/X, C_1/X, \mathit{Padding} \\
 & \ldots & \\
 & \mathtt{c}_e & \mapsto \quad C_{e,1}/X, \ldots, C_{e,v}/X, C_e/X, \mathit{Padding} \\
 & \mathtt{c} & \mapsto \quad t/C_1, \ldots, t/C_e, \mathit{Padding} \\
 & \mathtt{x} & \mapsto \quad X \\
 & \mathtt{t} & \mapsto \quad t \\
 & \mathtt{tt} & \mapsto \quad t/t, \mathit{Padding} \\
 & \mathtt{ttt} & \mapsto \quad (t/t)/t, \mathit{Padding}
\end{array}
$$

Figure 2: The grammar obtained from the general form by unifying all types assigned to $\mathtt{x}$.

follows to be parameter-preserving for all $c \geq 1$. If $c \geq 2$, $hg(HG, c)$ is defined as before, while for $c = 1$, the transformation would first solve the problem $(HG, 1)$ and then, depending on the outcome of this first step, output either a fixed YES-instance of $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY or a fixed NO-instance of $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY (for $k = 1$).

**Thm. 19** *$\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is* W[2]*-hard.*

**Proof.** By modifying the mentioned NP-hardness proofs, we show how to transform an instance $(G, k)$ of HITTING SET to $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY, preserving the parameter. To be more precise, the HITTING SET problem can be reduced in polynomial time to finding a grammar consistent with structures $D$ and in the class $\mathcal{G}_{k\text{-max-val}}$. We achieve this using the algorithm $hg$ as given in Definition 18 and the following remark. We can assume that $k \geq 2$ in the following discussion.

Consider the hypergraph $HG = (V, E)$, $G = \mathrm{GF}(hg(HG, k))$, and $k$ such that a cover of size $k$ exists for $HG$. Hence, $(HG, k)$ is a YES-instance of HITTING SET.

For any symbol $\mathtt{e}_i$, unification of all types $t/\Gamma_{n(i,1)}, \ldots, t/\Gamma_{n(i,d(i))}$ to $t/\Sigma_i$ will lead to a substitution such that $S_{g(1,i,0)} = S_{g(1,i,1)} = \cdots = S_{g(v,i,0)} = S_{g(v,i,1)}$. This is not possible, since this would require unification of $t/t$ and $(t/t)/t$, so for each symbol $\mathtt{e}_i$, at most one of the types $t/\Gamma_{n(i,1)}, \ldots t/\Gamma_{n(i,d(i))}$ can be unified with $t/\Upsilon_i$ instead. For each $i$, only one of these $t/\Gamma$ types can be chosen for this, since it will block unification of $t/\Upsilon_i$ with any of the other $t/\Gamma$ types: for any given $i$, the $U$ types in $\Upsilon_i$ all have to be of the same type, and

such a unification step will result in a substitution such that some $\Delta$ type will be substituted for all these $U$ types.

For every $i$, the $T$ types in $\Gamma_{n(i,j)}$, $1 \leq j \leq d(i)$, overlap: $T_{j,j}$ occurs twice in every $\Gamma_{n(i,j)}$.

Thus, they cannot all be unified with $\Sigma_i$, since the pair of the first and last $S$ type in every $\Sigma$-type is not unifiable.

Hence, for each $i$, *exactly* one of the $t/\Gamma$ types has to be unified with the $t/\Upsilon$ type, and the rest with the $\Sigma$ type. This implies a substitution such that for each $C_i$, a $\Delta_\ell$ is substituted such that $\ell = n(i,j)$, $1 \leq j \leq d(i)$. This corresponds to choosing vertex $\ell$ in the original hypergraph to cover edge $i$. Since, for all $i$, $t/C_i$ is assigned to symbol $\mathtt{c}$, and since zero padding types are assigned to this symbol (because $k \geq 2$), the number of distinct $\Delta$ types that substitute for the $C$ types can be no more than $k$. This proves that the answer to $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is YES.

Let $k$ and $HG$ be such that a minimum cover for $HG$ is of size $k' > k$, and let $G = \mathrm{GF}(hg(HG,k))$ as before. Hence, $(HG,k)$ is a NO-instance. Following the same line of reasoning as earlier in this proof, it is clear that for each $C_i$, a $\Delta_\nu$ must be substituted in order to obtain a consistent grammar that is in the class. Given the definition of $hg$, these $\Delta$ types correspond to one of the vertices that edge $i$ is incident on. Given that $k' > k$, there are at least $k'$ distinct such $\Delta$ types, and since for all $i$, $t/C_i$ is assigned to $\mathtt{c}$, at least $k'$ distinct types are assigned to $\mathtt{c}$ in a consistent grammar, which thus cannot be in $\mathcal{G}_{k\text{-max-val}}$. Thus the answer to $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is NO.

This proves that the answer to $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY for the sample $hg(HG,k)$ is the same as for HITTING SET for $HG$ with $k$ as size of the cover. Since the reduction $hg$ runs in polynomial time, this proves W[2]-hardness. □

As for the problem $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY, note that for $k \geq 2$, $\mathcal{L}_{k\text{-max-val}}$ contains $\Sigma^*$, which immediately trivializes the problem. Furthermore, we refer to [35] for a proof of NP-hardness for the case $k = 1$. It should be clear that these results render the parameterization discussed in this section meaningless from the viewpoint of parameterized complexity.

As an aside, let us mention that in the literature (see Corollary 5.13 from [2], for example) also consistency questions related to the *least-k-valued* grammars and languages were considered. This means we are looking for the smallest $k$ such that there is a grammar $G \in \mathcal{G}_{k\text{-max-val}}$ and $D \subseteq \mathrm{FL}(G)$. These problems are also known to be NP-hard, but it is an open question whether they belong to NP.

## 8. *k*-Sum-Value Problems

In parameterized complexity, the sum-variants of problems are often somewhat easier than the max-variants, see [39] for a nice overview. It is not clear whether this is also the case here, and we were not able to adapt the previous reduction (using HITTING SET). We were only able to prove the following, weaker result:

**Thm. 20** $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY *is* NP*-complete.*

**Proof.** The relationship between $\mathcal{G}_{k\text{-sum-val}}$ and $\mathcal{G}_{\text{least-card}}$ (recall Proposition 4) implies the following: when $k$ has the smallest value so that, for some given sample, the answer to the consistency problem is YES, $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY is equivalent to $\mathcal{FL}_{\text{LEAST-CARD}}$-CONSISTENCY.

Thus, $\mathcal{FL}_{\text{LEAST-CARD}}$-CONSISTENCY for some $D$ can be answered by considering $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY for $D$ with increasing values for $k$, starting at $k = |\Sigma|$. Since $\mathcal{FL}_{\text{LEAST-CARD}}$-CONSISTENCY is known to be NP-hard (see [2], Corollaries 5.14 and 5.18), it is thus NP-hard for $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY. Given a grammar $G \in \mathcal{G}_{k\text{-sum-val}}$ as a witness, both its consistency with the sample and membership in $\mathcal{G}_{k\text{-sum-val}}$ can be checked in polynomial time, so $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY is NP-complete. □

This leaves open the existence of FPT-algorithms for this problem. Such algorithms are usually based on some enumeration strategy, but a naive application of such an approach will not work in this case, since it is easy to come up with constructions such that the search space is of a size exponential in $\|D\|$.

Similar question can be asked for Hölder norms other than the max or sum norm, and the corresponding language families as introduced in Section 5.

## 9. Reparameterizations

As already seen with the example of VERTEX COVER versus INDEPENDENT SET, basically the same problem can be parameterized in different ways, possibly leading to positive (FPT) results or to negative (W[1]-, W[2]-hardness) results. So it might be that other choices of parameterization may lead to FPT-algorithms. This question of finding suitable parameterizations is both of theoretical and of practical importance, since it allows for a localization of the hard parts in the input and may also be helpful in finding practically useful algorithms. Here, not only single parameterizations are interesting, but also combinations of various parameters, as discussed in recent invited talks of Fellows and Niedermeier, see [40, 41].

One other natural choice of a parameter is the number of unification steps $u$ needed to transform the general form of $D$ into some $k$-max-valued (or $k$-sum-valued) grammar $G$ such that $D \subseteq \mathrm{L}(G)$. This leads to problems like $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY. The input to such a problem would be a triple $(D, u, k)$, where $D$ is the finite input sample. A variant could be UNIFORM $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY, where the input would be $(D, k)$, and the question would be whether there exists a $u$ such that $(D, u, k)$ is a YES-instance of $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY. Hence, the inputs to UNIFORM $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY and to $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY are the same.

**Thm. 21** $(D, k)$ *is a YES-instance to* $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY *(*$\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY*, resp.) if and only if* $(D, k)$ *is a YES-instance to* UNIFORM $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY *(*$\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY*, resp.).*

**Proof.** Given $G = \mathrm{GF}(D)$ and $c$, where $D$ is a sample of a structure language, a grammar $G' \in \mathcal{G}_{k\text{-sum-val}}$ consistent with $D$ has to be such that $G' \supseteq \sigma[G]$ for some substitution $\sigma$. This substitution corresponds to a finite number of unification steps over $G$.

In the case that $D$ is a sample of a string language, all possible structure language samples consistent with this sample can be computed. For each of these samples the argument for samples of structure languages holds. □

Along the same line of reasoning we can show:

**Thm. 22** *$(D, k)$ is a YES-instance to $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY if and only if $(D, k)$ is a YES-instance to* UNIFORM $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY*.*

In conclusion, the uniform problem variants do not offer new insights. However, they immediately provide:

**Cor. 23** UNIFORM $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY*,* UNIFORM $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY*, and* UNIFORM $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY *are* NP*-complete.*
UNIFORM $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY *is* W[2]*-hard.*

Instead of a single parameter, one could also consider two or more parameters. (Formally, this is captured by our definition by combining those multiple parameters into one single parameter.) So, the FPT-question of $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY would be whether an algorithm exists that runs in time $f(u, k)p(\|D\|)$ for some function $f$ and some polynomial $p$.

**Thm. 24** *$u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY is in* FPT*.*

**Proof.** Given $D$, $\|\mathrm{GF}(D)\|$ is exactly $\|D\|$. Hence, in order to obtain a grammar with exactly $k$ type assignments, at most $\|D\| - k$ unification steps need to be performed to obtain a grammar in the class. Given that $u$ bounds the number of these steps, for a sample larger than $u + k$ the answer to $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY is always NO.

If $\|D\| \leq u + k$, a grammar might be obtained by unifying types in $\mathrm{GF}(D)$. Given $\mathrm{GF}(D)$, the number of all possible unification steps is upper bounded by the number of pairs[10] of types in $\mathrm{GF}(D)$, i.e., $\frac{1}{2} \cdot \|\mathrm{GF}(D)\| \cdot (\|\mathrm{GF}(D)\| - 1)$, which is overestimated by $\frac{1}{2} \cdot (u + k)^2$. Out of all these pairs, the learning algorithm must choose at most $u$ pairs. This puts an upper bound on the size of the search-space of

$$f(u, k) := \frac{u!}{(\frac{1}{2} \cdot (u + k)^2)! \cdot (u - \frac{1}{2} \cdot (u + k)^2)!}$$

where $u$ and $k$ are fixed parameters. Since each unification step can be performed in time linear[11] in $\|D\|$, an answer can be obtained in time $f(u, k) \cdot p(\|D\|)$ for some polynomial $p$.

[10] In the sense of sets with cardinality 2.
[11] Provided the terms are represented as DAGs or systems of equations, see [42].

Thus, for a sample larger than $u + k$, the answer is always NO, and in the case that $\|D\| \leq u + k$, it takes FPT-time to settle the question. □

**Thm. 25** $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY *is in* FPT.

**Proof.** We can easily extend Theorem 24 to the case of string languages.

Just as is the case for structure languages, in the case that $\|D\| > u+k$, the answer to $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY is always NO.

In the case that $\|D\| \leq u + k$, a grammar may exist that is consistent with $D$ and is in the class. It can be obtained analogously to the procedure defined in the proof of Theorem 17: again, we use a nondeterministic procedure that consists of two parts; first, for each string in $D$, a derivation is chosen. Then, the union of the resulting structures, $D'$ (note that $\|D'\|$ is obviously polynomial in $\|D\|$), is used as a sample for learning from structures.

The Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$ specifies the number of pairwise distinct binary trees with $n+1$ leaves. Since these trees have $n$ internal nodes and, since in the context of CCG, every one of these can be labeled as either forward or backward application, the number of possible functor-argument structures for a string of length $l$ is $2^{l-1}C_l$. The length of the strings in $D$ is upper bounded by $\|D\|$ and thus by $u + k$, and the number of strings is $|D|$, which is upper bounded by $u + k$ as well.

Thus, given $D$, the number of possible structure samples $D'$ is bounded by

$$g(u,k) := \left(2^{u+k-1}\frac{1}{u+k}\binom{2(u+k-1)}{(u+k-1)}\right)^{u+k}$$

where $u$ and $k$ are fixed parameters.

For each of the possible $D'$, a problem from $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY can be considered, which is in FPT, as shown above. Thus $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY is in FPT. □

On first sight, it may seem that the similar $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY and $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY problems can be approached in the same way. However, by definition, $k$ bounds only the number of types assigned to single symbols, so the total size of the consistent grammar is not bounded with these parameters. This can be solved by including $|\Sigma|$ as a parameter. Thus we obtain the following two results:

**Thm. 26** FIXED $|\Sigma|$ $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY *is in* FPT.

**Proof.** It is easy to see that, given that $u$ bounds the number of unification steps, for a sample larger than $u + |\Sigma| \cdot k$, the answer to FIXED $|\Sigma|$ $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is always NO.

When the sample is smaller than this, we obtain the same bound for pairs of types as in the proof of Theorem 24, namely $\frac{1}{2} \cdot \|\mathrm{GF}(D)\| \cdot (\|\mathrm{GF}(D)\| - 1)$, and thus a bound on the size of the search-space of

$$\frac{u!}{(u + |\Sigma| \cdot k)!(|\Sigma| \cdot k)!}.$$

Thus, FIXED $|\Sigma|$ $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY is in FPT. □

**Thm. 27** FIXED $|\Sigma|$ $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY *is in* FPT.

**Proof.** As in the case for structure languages, for a sample larger than $u+|\Sigma|\cdot k$ the answer to FIXED $|\Sigma|$ $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY is NO.

For smaller samples a consistent grammar may exist, so analogous to the proof of Theorem 25, we can consider the number of derivations compatible with the strings in $D$. Again, the length of the strings in $D$ is upper bounded by $\|D\|$, and thus, in this case, by $u + |\Sigma| \cdot k$, and the number of strings is $|D|$, which is upper bounded by $u + |\Sigma| \cdot k$ as well.

Thus, given $D$, the number of possible structure samples $D'$ is bounded by

$$\left(2^{u+|\Sigma|\cdot k-1}\frac{1}{u+|\Sigma|\cdot k}\binom{2(u+|\Sigma|\cdot k-1)}{(u+|\Sigma|\cdot k-1)}\right)^{u+|\Sigma|\cdot k}.$$

For each of the possible $D'$, FIXED $|\Sigma|$ $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY can be considered, which is in FPT. Thus FIXED $|\Sigma|$ $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY is in FPT. □

One could also study the step number $u$ as a parameter in isolation. One problem formulation could be the following one: Given a finite sample $D$ and a categorial grammar $G$ (and the parameter $u$), does there exist a sequence of at most $u$ unification steps, starting from the general form $\mathrm{GF}(D)$ and leading to $G$? This might be an interesting subject for future study. However, note that the slightly more general problem of deciding the existence of such a sequence from some arbitrary given grammar (not necessarily in general form) is already at least as hard as the well-known GRAPH ISOMORPHISM problem for $u = 0$.[12] We can encode a graph into a grammar by assigning to one single symbol the types $T_i/T_j$ for every edge from vertex $i$ to vertex $j$ in the graph (and assigning $T_\ell$ and $t/T_\ell$ for every vertex $\ell$ to avoid useless types). Let $G_1$ and $G_2$ be two such grammars encoding graphs $\mathit{Graph}_1$ and $\mathit{Graph}_2$, then $u = 0$ (i.e., an empty sequence of unification steps) just if there exists a renaming such that, when it is applied to $G_1$, $G_2$ is obtained. It is easy to see that this is only the case if $\mathit{Graph}_1$ and $\mathit{Graph}_2$ are isomorphic.

Finally note that similar FPT results can be shown for language families whose definition is based on other Hölder norms, see Sec. 5, but this is not so interesting as long as it is still unsettled if the corresponding consistency problems are NP-hard, see Sec. 6.

## 10. Consequences for the Complexity of Learning

In [2], the hardness results were phrased in different terminology. To formulate these, we need some more notions.

[12] Though it is not known if GRAPH ISOMORPHISM is NP-complete, this problem is also believed not to be solvable in polynomial time, see [43, 44, 45].

Fix a grammar system $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$. A learning function $\varphi$ learns a grammar class $\mathcal{G} \subseteq \Omega$ *prudently* if it learns $\mathcal{G}$ and the range of $\varphi$ is a subset of $\mathcal{G}$. In other words, hypotheses are only drawn from $\mathcal{G}$. A learning function $\varphi$ is *responsive* on $\mathcal{G} \subseteq \Omega$ if it is defined on any finite sequence of words the range of which is a subset of some language $\mathrm{L}(G)$, where $G \in \mathcal{G}$. Hence, whenever possible, $\varphi$ should produce a hypothesis. A learning function $\varphi$ is *consistent* if, for any finite sequence $s$ the range of which is a subset of some language $\mathrm{L}(G)$, where $G \in \mathcal{G}$, either $\varphi(s)$ is undefined or the set of words enumerated by $s$ is a subset of $\mathrm{L}(\varphi(s))$.

Consider now more specifically the grammar system $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL} \rangle$, i.e., consider learning from structures. For instance, [2, Proposition 5.12] reads as follows:

**Prop. 28** *Let $\varphi$ be a learning function that, given $k \in \mathbb{N}^+$, can learn any of the classes $\mathcal{G}_{k\text{-}max\text{-}val}$ from structures, and that,* for each *$k$, is responsive and consistent on this class and learns this class prudently. Deciding whether $\varphi$ is defined for an arbitrary sample $D$ is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

More generally, we can phrase the problem $\mathcal{G}_k$-LEARNABILITY as follows: The input is a uniform learning function $\varphi$ that learns each family $\mathcal{G}_k$ prudently and that is, for each $k$, responsive and consistent on this class. The problem is to decide if $\varphi$ is defined for arbitrary inputs $(D, k)$. (A similar problem can be phrased for learning from strings.)

The complexity results obtained in this paper are summarized in Table 1. They can be easily interpreted as results on learnability. For instance, we could phrase:

**Prop. 29** *Let $\varphi$ be a learning function that, given $k \in \mathbb{N}^+$, can learn any of the classes $\mathcal{G}_{k\text{-}sum\text{-}val}$ from structures, and that,* for each *$k$, is responsive and consistent on this class and learns this class prudently. Deciding whether $\varphi$ is defined for an arbitrary sample $D$ is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

## 11. Conclusions and Future Research

The main contributions of this paper are the complexity results as summarized in the previous section, in particular, in Table 1. We have shown that for one natural parameterization, the consistency problem for one of the classes is (most likely) not in FPT. For a parameterization that includes a bound on the number of unification steps that the learner is allowed to perform, we have obtained positive results.

Furthermore, we have defined a new (natural) hierarchy of families that we have shown to be learnable from both strings and structures. We have also generalized this hierarchy to a family of hierarchies defined in terms of Hölder norm bounds on the descriptional complexity of categorial grammars. In general

| Problem | Complexity |
|---|---|
| $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY | W[2]-hard |
| $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY | NP-complete |
| UNIFORM $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY | W[2]-hard |
| UNIFORM $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY | NP-complete |
| UNIFORM $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY | NP-complete |
| UNIFORM $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY | NP-complete |
| $u$-STEP $\mathcal{FL}_{k\text{-SUM-VAL}}$-CONSISTENCY | FPT |
| $u$-STEP $\mathcal{L}_{k\text{-SUM-VAL}}$-CONSISTENCY | FPT |
| FIXED $\lvert\Sigma\rvert$ $u$-STEP $\mathcal{FL}_{k\text{-MAX-VAL}}$-CONSISTENCY | FPT |
| FIXED $\lvert\Sigma\rvert$ $u$-STEP $\mathcal{L}_{k\text{-MAX-VAL}}$-CONSISTENCY | FPT |

Table 1: Summary of the complexity results demonstrated in this paper.

terms, we believe that there are further quite interesting connections between the areas of Grammatical Inference and that of Descriptional Complexity that deserve further study. For example, we could associate to any nondeterministic $n$-state automaton $A$ an $n$-dimensional vector $\vec{v}(A)$, indicating in component $i$ the amount of lookahead needed to disambiguate any nondeterminism in state $s_i$. Then, a deterministic automaton $A$ is $k$-reversible [46] iff its reversal $A^r$ satisfies: $\| \vec{v}(A^r) \|_\infty \leq k$. From the viewpoint detailed in this paper, it appears natural to study the learnability of classes of regular languages defined by the restriction $\| \vec{v}(A^r) \|_p \leq k$ for any $p \geq 1$.

From a technical point of view, it would be nice to complement our W[2]-hardness result (Theorem 19) by demonstrating membership in W[2]. A natural idea would be to design a multi-tape Turing machine with one tape storing or counting the rule (applications) for each symbol. However, it is not clear if such a Turing machine would need only $f(k)$ many steps to decide consistency. Such a question might also be interpreted in the direction of parallelizability of derivations in categorial grammars. We are not aware of any such study for this type of mechanism, but we would like to point to the fact that studies in this direction were undertaken for the weakly equivalent mechanism of context-free grammars, see [47, 48] and references therein.

The obvious interpretation of our positive (FPT) results would be that, as long as the parameters $k$, $u$, and $|\Sigma|$ are kept low, the classes under consideration are efficiently learnable, in the sense of having polynomial update time. Of these parameters, the last one is the most problematic, since for typical (NLP) applications the lexicon is very large. Thus, our analysis suggests the approach of choosing as a parameter the total number of distinct types in the grammar, and keeping this number low. This may also have implications for theories of natural language acquisition.

It would be interesting to study the consistency problem for other language class hierarchies, where each class has finite elasticity. A natural candidate are the language families based on different Hölder norms as introduced in this

paper. One further example might be families based on elementary formal systems as examined by Moriyama and Sato [28].

*Acknowledgements.* We would like to thank the two anonymous reviewers for their comments. The research was partly funded by the University of Trier, which allowed the first author to visit Trier. This work was partly conducted while the first author was at K.U. Leuven, Department of Computer Science, Leuven, Belgium.

## References

[1] M. Kanazawa, Learnable Classes of Categorial Grammars, Phd thesis, CSLI, 1998.

[2] C. Costa Florêncio, Learning Categorial Grammars, Ph.D. thesis, Universiteit Utrecht, The Netherlands, 2003.

[3] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer, 1999.

[4] M. Holzer, M. Kutrib, Descriptional complexity — an introductory survey, in: C. Martín-Vide (Ed.), Scientific Applications of Language Methods, volume 2 of *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, Imperial College Press, 2010, pp. 1–58.

[5] R. G. Downey, P. A. Evans, M. R. Fellows, Parameterized learning complexity, in: L. Pitt (Ed.), COLT'93, ACM, 1993, pp. 51–57.

[6] F. Stephan, R. Yoshinaka, T. Zeugmann, On the parameterised complexity of learning patterns, in: E. Gelenbe, R. Lent, G. Sakellari (Eds.), Computer and Information Sciences II - 26th International Symposium on Computer and Information Sciences, ISCIS, Springer, 2011, pp. 277–281.

[7] C. Costa Florêncio, H. Fernau, Finding consistent categorial grammars of bounded value: a parameterized approach, in: A.-H. Dediu, H. Fernau, C. Martín-Vide (Eds.), Language and Automata Theory and Applications LATA, volume 6031 of *LNCS*, Springer, 2010, pp. 202–213.

[8] C. Costa Florêncio, H. Fernau, Hölder norms and a hierarchy theorem for parameterized classes of CCG, in: J. M. Sempere, P. García (Eds.), International Colloquium on Grammatical Inference ICGI, volume 6339 of *LNCS*, Springer, 2010, pp. 280–283.

[9] W. Buszkowski, Discovery procedures for categorial grammars, in: E. Klein, J. van Benthem (Eds.), Categories, Polymorphism and Unification, University of Amsterdam, 1987, pp. 35–64.

[10] W. Buszkowski, G. Penn, Categorial grammars determined from linguistic data by unification, Studia Logica 49 (1990) 431–454.

[11] K. Adjukiewicz, Die syntaktische Konnexität, Studia Philosophica 1 (1935) 1–27.

[12] J. Lambek, The mathematics of sentence structure, Am. Math. Mon. 65 (1958) 154–170.

[13] M. Steedman, Categorial grammar. Tutorial overview, Lingua 90 (1993) 221–258.

[14] M. Steedman, The Syntactic Process, MIT Press/Bradford Books, 2000.

[15] M. Moortgat, Categorial type logics, in: J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, Elsevier Science B.V., 1997, pp. 93–177. Chapter 2.

[16] D. Béchet, A. Foret, $k$-valued non-associative Lambek grammars are learnable from function-argument structures, in: Proceedings. of the 10th Workshop on Logic, Language, Information and Computation (WoLLIC'2003), Ouro Preto, Brazil, volume 84, pp. 60–72.

[17] C. Costa Florêncio, A limit point for rigid associative-commutative Lambek grammars, in: A. Copestake, J. Hajič (Eds.), Proceedings of EACL2003, Tenth Conference of the European Chapter of the Association for Computational Linguistics, Budapest, April 12–17, pp. 75–82.

[18] J. Dassow, H. Fernau, Comparison of some descriptional complexities of 0L systems obtained by a unifying approach, Inf. and Comput. 206 (2008) 1095–1103.

[19] E. M. Gold, Language identification in the limit, Inf. and Control 10 (1967) 447–474.

[20] S. Jain, D. N. Osherson, J. Royer, A. Sharma, Systems that Learn: An Introduction to Learning Theory, The MIT Press, Cambridge, MA., second edition, 1999.

[21] D. Angluin, Inductive inference of formal languages from positive data, Inf. and Control 45 (1980) 117–135.

[22] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun, Grammar Systems: A Grammatical Approach to Distribution and Cooperation, London: Gordon and Breach, 1994.

[23] D. Dudau-Sofronie, I. Tellier, M. Tommasi, A learnable class of classical categorial grammars from typed examples, in: 8th Conference on Formal Grammar, pp. 77–88.

[24] S. A. Fulop, On The Logic And Learning Of Language, Trafford on Demand Publishing, 2004.

[25] T. Yokomori, Polynomial-time learning of very simple grammars from positive data, in: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, ACM Press, University of California, Santa Cruz, 1991, pp. 213–227.

[26] R. Wiehagen, T. Zeugmann, Too much information can be too much for learning efficiently, in: K. P. Jantke (Ed.), Analogical and Inductive Inference: Proceedings of the International Workshop AII'92, Springer, Berlin, Heidelberg, 1992, pp. 72–86.

[27] R. Wiehagen, T. Zeugmann, Ignoring data may be the only way to learn efficiently, J. of Exp. and Theor. Artif. Intelligence 6 (1994) 131–144.

[28] T. Moriyama, M. Sato, Properties of language classes with finite elasticity, in: K. P. Jantke, et al. (Eds.), 4th Workshop on Algorithmic Learning Theory ALT'93, volume 744 of *LNCS/LNAI*, pp. 187–196.

[29] T. Motoki, T. Shinohara, K. Wright, The correct definition of finite elasticity: Corrigendum to identification of unions, in: COLT'91, Morgan Kaufmann, 1991, p. 375.

[30] K. Wright, Identification of unions and languages drawn from an identifiable class, in: COLT'89, Morgan Kaufmann, 1989, pp. 328–333.

[31] K. Mehlhorn, Graph algorithms and NP-completeness, Heidelberg: Springer, 1984.

[32] J. Flum, M. Grohe, Parameterized Complexity Theory, Text in Theoretical Computer Science, Springer, 2006.

[33] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.

[34] J. Chen, I. A. Kanj, G. Xia, Improved upper bounds for vertex cover, Theoretical Computer Science 411 (2010) 3736–3756.

[35] C. Costa Florêncio, Consistent identification in the limit of rigid grammars from strings is NP-hard, in: P. Adriaans, H. Fernau, M. van Zaanen (Eds.), Grammatical Inference: Algorithms and Applications; 6th International Colloquium, ICGI, volume 2484 of *LNCS/LNAI*, Springer, 2002, pp. 49–62.

[36] T. Gallai, Über extreme Punkt- und Kantenmengen, Ann. Univ. Sci. Budapest, Eötvös Sect. Math. 2 (1959) 133–138.

[37] I. v. Rooij, Tractable Cognition: Complexity Theory in Cognitive Psychology, Ph.D. thesis, University of Victoria, Canada, 2003.

[38] I. v. Rooij, The tractable cognition thesis, Cognitive Science 32 (2008) 939–984.

[39] M. Serna, D. M. Thilikos, Parameterized complexity for graph layout problems, EATCS Bulletin 86 (2005) 41–65.

[40] M. R. Fellows, Towards fully multivariate algorithmics: Some new results and directions in parameter ecology, in: J. Fiala, J. Kratochvíl, M. Miller (Eds.), Combinatorial Algorithms, 20th International Workshop, IWOCA, volume 5874 of *LNCS*, Springer, 2009, pp. 2–10.

[41] R. Niedermeier, Reflections on multivariate algorithmics and problem parameterization, in: J.-Y. Marion, T. Schwentick (Eds.), 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2010, pp. 17–32.

[42] F. Baader, J. H. Siekmann, Unification theory, in: [49], Oxford University Press, 1994, pp. 41–125.

[43] D. S. Johnson, The NP-completeness column, ACM Transactions on Algorithms 1 (2005) 160–176.

[44] J. Köbler, U. Schöning, J. Torán, Graph Isomorphism Problem: The Structural Complexity, Birkhäuser Verlag, 1993.

[45] R. C. Read, D. G. Corneil, The graph isomorphism disease, J. of Graph Theory 1 (1977) 339–363.

[46] D. Angluin, Inference of reversible languages, J. of the ACM 29 (1982) 741–765.

[47] F. J. Brandenburg, On the height of syntactical graphs, in: P. Deussen (Ed.), Proceedings of the 5th GI-Conference on Theoretical Computer Science, volume 104 of *LNCS*, Springer, Karlsruhe, FRG, 1981, pp. 13–21.

[48] K. Reinhardt, A parallel context-free derivation hierarchy, in: G. Ciobanu, G. Păun (Eds.), Fundamentals of Computation Theory, 12th International Symposium, FCT '99, volume 1684 of *LNCS*, Springer, 1999, pp. 441–450.

[49] D. M. Gabbay, C. J. Hogger, J. A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford: Oxford University Press, 1994.

## Appendix A. Example of the construction

**Ex. 30** *Consider the hypergraph $HG = (V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2, 3\}, \{2, 3\}, \{3, 4\}\}$.*
*Then $hg(HG)$ is defined as in Fig. A.3.*

edges:
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_1, \mathtt{fa}(\mathtt{c}_1, x)), \mathtt{fa}(\mathtt{c}_1, x)), \mathtt{fa}(\mathtt{c}_1, x)), \mathtt{u}_1), \mathtt{u}_1), \mathtt{u}_1)$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_2, \mathtt{fa}(\mathtt{c}_2, x)), \mathtt{fa}(\mathtt{c}_2, x)), \mathtt{u}_2), \mathtt{u}_2), \mathtt{u}_2)$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_3, \mathtt{fa}(\mathtt{c}_3, x)), \mathtt{fa}(\mathtt{c}_3, x)), \mathtt{u}_3), \mathtt{u}_3), \mathtt{u}_3)$

vertices each edge is incident on:
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_1, \mathit{vertex}1), \mathtt{g}_{n(1,2)}), \mathtt{g}_{n(1,3)}), \mathtt{fa}(\mathtt{t}_{1,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{1,1}, \mathtt{x})), \mathtt{t}_{1,3}), \mathtt{t}_{1,4})$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_1, \mathtt{g}_{n(2,1)}), \mathit{vertex}2, \mathtt{g}_{n(2,3)}), \mathtt{t}_{2,1}), \mathtt{fa}(\mathtt{t}_{2,2}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{2,2}, \mathtt{x})), \mathtt{t}_{2,4})$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_1, \mathtt{g}_{n(3,1)}), \mathtt{g}_{n(3,2)}), \mathit{vertex}3), \mathtt{t}_{3,1}), \mathtt{t}_{3,2}), \mathtt{fa}(\mathtt{t}_{3,3}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{3,3}, \mathtt{x}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_2, \mathit{vertex}2), \mathtt{g}_{n(4,2)}), \mathtt{fa}(\mathtt{t}_{4,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{4,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{4,3}, \mathtt{x}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_2, \mathtt{g}_{n(5,1)}), \mathit{vertex}3), \mathtt{fa}(\mathtt{t}_{5,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{5,2}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{5,2}, \mathtt{x}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_3, \mathit{vertex}3), \mathtt{g}_{n(6,2)}), \mathtt{fa}(\mathtt{t}_{6,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{6,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{6,3}, \mathtt{x}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_3, \mathtt{g}_{n(7,1)}), \mathit{vertex}4), \mathtt{fa}(\mathtt{t}_{7,1}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{7,2}, \mathtt{x})), \mathtt{fa}(\mathtt{t}_{7,2}, \mathtt{x}))$

non-assigned:
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_1, \mathtt{f}_{n(1,1)}), \mathtt{f}_{n(1,1)}), \mathtt{f}_{n(1,2)}), \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{s}_{1,1}), \mathtt{s}_{1,2}), \mathtt{fa}(\mathtt{ttt}, \mathtt{t}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_2, \mathtt{f}_{n(2,1)}), \mathtt{f}_{n(2,1)}), \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{s}_{2,1}), \mathtt{fa}(\mathtt{ttt}, \mathtt{t}))$
$\mathtt{fa}(\ldots \mathtt{fa}(\mathtt{e}_3, \mathtt{f}_{n(3,1)}), \mathtt{f}_{n(3,1)}), \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{s}_{3,1}), \mathtt{fa}(\mathtt{ttt}, \mathtt{t}))$

cover :
$\mathtt{fa}(\mathtt{c}, \mathtt{fa}(\mathtt{c}_1, \mathtt{x})), \ldots \mathtt{fa}(\mathtt{c}, \mathtt{fa}(\mathtt{c}_e, \mathtt{x}))$

incompatible types :
$\mathtt{fa}(\mathtt{tt}, \mathtt{fa}(\mathtt{tt}, \mathtt{t}))$
$\mathtt{fa}(\mathtt{fa}(\mathtt{ttt}, \mathtt{fa}(\mathtt{tt}, \mathtt{t})), \mathtt{fa}(\mathtt{tt}, \mathtt{t}))$

Padding :
$\mathtt{ba}(\mathtt{x}, \mathtt{c})$
$\mathtt{ba}(\mathtt{x}, \mathtt{c}_1), \mathtt{ba}(\mathtt{x}, \mathtt{c}_2), \mathtt{ba}(\mathtt{x}, \mathtt{c}_3)$
$\mathtt{ba}(\mathtt{x}, \mathtt{v}_1), \mathtt{ba}(\mathtt{x}, \mathtt{v}_2), \mathtt{ba}(\mathtt{x}, \mathtt{v}_3), \mathtt{ba}(\mathtt{x}, \mathtt{v}_4)$
$\mathtt{ba}(\mathtt{x}, \mathtt{t}_{1,1}), \mathtt{ba}(\mathtt{x}, \mathtt{t}_{2,2}), \mathtt{ba}(\mathtt{x}, \mathtt{t}_{3,3})$
$\mathtt{ba}(\mathtt{x}, \mathtt{t}_{4,1}), \mathtt{ba}(\mathtt{x}, \mathtt{t}_{5,2}), \mathtt{ba}(\mathtt{x}, \mathtt{t}_{6,1}), \mathtt{ba}(\mathtt{x}, \mathtt{t}_{7,2})$
$\mathtt{ba}(\mathtt{x}, \mathtt{tt}), \mathtt{ba}(\mathtt{x}, \mathtt{ttt})$

Vertices :
$\mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{v}_1), \mathtt{x})$,
$\mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{fa}(\mathtt{v}_2, \mathtt{x})), \mathtt{x})$,
$\mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{v}_3), \mathtt{x})), \mathtt{x})$,
$\mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{fa}(\mathtt{ba}(\mathtt{x}, \mathtt{fa}(\mathtt{v}_4, \mathtt{x})), \mathtt{x})), \mathtt{x})$

where $\mathit{vertex}1 = \mathtt{fa}(\mathtt{v}_1, \mathtt{x})$, $\mathit{vertex}2 = \mathtt{fa}(\mathtt{v}_2, \mathtt{x})$ etc.

Figure A.3: Type assignments in $hg(HG)$.

*We define the substitution $\sigma$ to be such that for every $i$, all types assigned to $\mathtt{c}_i$ are unified, and all types assigned to $\mathtt{x}$ are unified to $X$. This yields the grammar shown in Fig. A.4.*

$$
\begin{array}{lll}
 & \mathtt{e}_1 & \mapsto (((((t/C_1)/C_1)/C_1)/U_1)/U_1)/U_1)/U_1, \\
 & & ((((((t/V_1)/G_{n(1,2)})/G_{n(1,3)})/T_{1,1,1})/T_{1,1,2})/T_{1,3})/T_{1,4}, \\
 & & ((((((t/G_{n(2,1)})/V_2)/G_{n(2,3)})/T_{2,1})/T_{2,2,1})/T_{2,2,2})/T_{2,4}, \\
 & & ((((((t/G_{n(3,1)})/G_{n(3,2)})/V_3)/T_{3,1})/T_{3,2})/T_{3,3,1})/T_{3,3,2}, \\
 & & ((((((t/F_{n(1,1)})/F_{n(1,2)})/F_{n(1,3)})/T_7)/S_{1,1})/S_{1,2})/T_9 \\
 & \mathtt{e}_2 & \mapsto ((((t/C_2)/C_2)/U_2)/U_2)/U_2, \\
 & & ((((t/V_{2,1})/G_{n(4,2)})/T_{4,1,1})/T_{4,1,2})/T_{4,3}, \\
 & & ((((t/G_{n(5,1)})/V_{3,1})/T_{5,1})/T_{5,2,1})/T_{5,2,2}, \\
 & & ((((t/F_{n(2,1)})/F_{n(2,2)})/T_{11})/S_{2,1})/T_{13} \\
 & \mathtt{e}_3 & \mapsto ((((t/C_3)/C_3)/U_3)/U_3)/U_3, \\
 & & ((((t/V_{3,2})/G_{n(6,2)})/T_{6,1,1})/T_{6,1,2})/T_{6,3}, \\
 & & ((((t/G_{n(7,1)})/V_4)/T_{7,1})/T_{7,2,1})/T_{7,2,2}, \\
 & & ((((t/F_{n(3,1)})/F_{n(3,2)})/T_{15})/S_{3,1})/T_{17} \\
 & \mathtt{c} & \mapsto t/C_1, t/C_2, t/C_3, X\backslash t \\
 & \mathtt{t} & \mapsto T_2, T_5, T_6, T_8, T_{10}, T_{12}, T_{14}, T_{16}, T_{18} \\
 & \mathtt{tt} & \mapsto t/T_1, T_1/T_2, T_3/T_5, T_4/T_6, T_7/T_8, T_{11}/T_{12}, T_{15}/T_{16}, X\backslash t \\
 & \mathtt{ttt} & \mapsto (t/T_3)/T_4, T_9/T_{10}, T_{13}/T_{14}, T_{17}/T_{18}, X\backslash t \\
 & \mathtt{c}_1 & \mapsto C_1/X, X\backslash t \\
\sigma[\mathrm{GF}(D)] = & \mathtt{c}_2 & \mapsto C_2/X, X\backslash t \\
 & \mathtt{c}_3 & \mapsto C_3/X, X\backslash t \\
 & \mathtt{x} & \mapsto X \\
 & \mathtt{t}_{1,1} & \mapsto T_{1,1,1}/X, T_{1,1,2}/X, X\backslash t \\
 & \mathtt{t}_{2,2} & \mapsto T_{2,2,1}/X, T_{2,2,2}/X, X\backslash t \\
 & \mathtt{t}_{3,3} & \mapsto T_{3,3,1}/X, T_{3,3,2}/X, X\backslash t \\
 & \mathtt{t}_{4,1} & \mapsto T_{4,1,1}/X, T_{4,1,2}/X, X\backslash t \\
 & \mathtt{t}_{5,2} & \mapsto T_{5,2,1}/X, T_{5,2,2}/X, X\backslash t \\
 & \mathtt{t}_{6,1} & \mapsto T_{6,1,1}/X, T_{6,1,2}/X, X\backslash t \\
 & \mathtt{t}_{7,2} & \mapsto T_{7,2,1}/X, T_{7,2,2}/X, X\backslash t \\
 & \mathtt{v}_1 & \mapsto V_1/X, (X\backslash t)/X, X\backslash t \\
 & \mathtt{v}_2 & \mapsto V_2/X, V_{2,1}/X, (X\backslash(t/X))/X, X\backslash t \\
 & \mathtt{v}_3 & \mapsto V_3/X, V_{3,1}/X, V_{3,2}/X, (X\backslash((X\backslash t)/X))/X, X\backslash t \\
 & \mathtt{v}_4 & \mapsto V_4/X, (X\backslash((X\backslash(t/X))/X))/X, X\backslash t \\
 & \mathtt{g}_{n(1,2)} & \mapsto G_{n(1,2)} \\
 & \mathtt{f}_{n(1,1)} & \mapsto F_{n(1,1)} \\
 & \mathtt{s}_{1,2} \mapsto S_{1,2} & \mathtt{u}_1 \mapsto U_1 \\
 & \mathtt{s}_{1,3} \mapsto S_{1,3} & \mathtt{u}_2 \mapsto U_2 \\
 & \ldots & \mathtt{u}_3 \mapsto U_3
\end{array}
$$

Figure A.4: Unification of type assignments.

*Let $G'$ be obtained from $\sigma[\mathrm{GF}(D)]$ by unifying the types assigned to* `c`, `tt`, *and all other symbols that have padding types assigned to them, resp., so that just two type assignments remain for all these symbols (we suppress some irrelevant assignments here for clarity), see Fig. A.5.*

$$
\begin{array}{rll}
 & \mathtt{e}_1 & \mapsto (((((((t/C_1)/C_1)/C_1)/U_1)/U_1)/U_1)/U_1, \\
 & & (((((((t/(X\backslash t))/G_{n(1,2)})/G_{n(1,3)})/T_{1,1})/T_{1,1})/T_{1,3})/T_{1,4}, \\
 & & (((((((t/G_{n(2,1)})/(X\backslash(t/X)))/G_{n(2,3)})/T_{2,1})/T_{2,2})/T_{2,2})/T_{2,4}, \\
 & & (((((((t/G_{n(3,1)})/G_{n(3,2)})/(X\backslash((X\backslash t)/X)))/T_{3,1})/T_{3,2})/T_{3,3})/T_{3,3}, \\
 & & (((((((t/F_{n(1,1)})/F_{n(1,2)})/F_{n(1,3)})/t)/S_{1,1})/S_{1,2})/(t/t) \\
 & \mathtt{e}_2 & \mapsto ((((t/C_1)/C_1)/U_2)/U_2)/U_2, \\
 & & ((((t/(X\backslash(t/X)))/G_{n(4,2)})/T_{4,1})/T_{4,1})/T_{4,3}, \\
 & & ((((t/G_{n(5,1)})/(X\backslash((X\backslash t)/X)))/T_{5,1})/T_{5,2})/T_{5,2}, \\
 & & ((((t/F_{n(2,1)})/F_{n(2,2)})/t)/S_{2,1})/(t/t) \\
 & \mathtt{e}_3 & \mapsto ((((t/C_1)/C_1)/U_3)/U_3)/U_3, \\
 & & ((((t/(X\backslash((X\backslash t)/X)))/G_{n(6,2)})/T_{6,1})/T_{6,1})/T_{6,3}, \\
 & & ((((t/G_{n(7,1)})/(X\backslash((X\backslash(t/X))/X)))/T_{7,1})/T_{7,2})/T_{7,2}, \\
 & & ((((t/F_{n(3,1)})/F_{n(3,2)})/t)/S_{3,1})/(t/t) \\
 & \mathtt{c} & \mapsto t/C_1, X\backslash t \\
 & \mathtt{t} & \mapsto t \\
G' = & \mathtt{tt} & \mapsto t/t, X\backslash t \\
 & \mathtt{ttt} & \mapsto (t/t)/t, X\backslash t \\
 & \mathtt{c}_1 & \mapsto C_1/X, X\backslash t \\
 & \mathtt{c}_2 & \mapsto C_1/X, X\backslash t \\
 & \mathtt{c}_3 & \mapsto C_1/X, X\backslash t \\
 & \mathtt{t}_{1,1} & \mapsto T_{1,1}/X, X\backslash t \\
 & \mathtt{t}_{2,2} & \mapsto T_{2,2}/X, X\backslash t \\
 & \mathtt{t}_{3,3} & \mapsto T_{3,3}/X, X\backslash t \\
 & \mathtt{t}_{4,1} & \mapsto T_{4,1}/X, X\backslash t \\
 & \mathtt{t}_{5,2} & \mapsto T_{5,2}/X, X\backslash t \\
 & \mathtt{t}_{6,1} & \mapsto T_{6,1}/X, X\backslash t \\
 & \mathtt{t}_{7,2} & \mapsto T_{7,2}/X, X\backslash t \\
 & \mathtt{v}_1 & \mapsto (X\backslash t)/X, X\backslash t \\
 & \mathtt{v}_2 & \mapsto (X\backslash(t/X))/X, X\backslash t \\
 & \mathtt{v}_3 & \mapsto (X\backslash((X\backslash t)/X))/X, X\backslash t \\
 & \mathtt{v}_4 & \mapsto (X\backslash((X\backslash(t/X))/X))/X, X\backslash t
\end{array}
$$

Figure A.5: More unification steps.

*All that remains now to obtain a grammar that is in $\mathcal{G}_{k\text{-}max\text{-}val}$, $k = 2$, is to unify some types assigned to $\mathtt{e}_1$, $\mathtt{e}_2$, $\mathtt{e}_3$, resp. Given the construction, for every one of these symbols, one of the types that represent vertices has to be unified with the type that represents a hyperedge, and the rest to the single remaining type.*

*For $\mathtt{e}_1$ this means that the fourth assigned type has to be unified with the first one, for $\mathtt{e}_2$ the third type needs to be unified with the first one, and for $\mathtt{e}_3$, the second with the first one. For each of these symbols, the remaining types should be unified with the last type assigned, yielding $G''$, cf. Fig. A.6. This grammar is in $\mathcal{G}_{k\text{-}max\text{-}val}$, $k = 2$.*

$$
G'' = \begin{array}{lcl}
\mathtt{e}_1 & \mapsto & ((((((t/(X\backslash((X\backslash t)/X)))/(X\backslash((X\backslash t)/X)))/ \\
 & & \quad (X\backslash((X\backslash t)/X)))/U_1)/U_1)/U_1, \\
 & & ((((((t/(X\backslash t))/(X\backslash(t/X)))/F_{n(1,3)})/t)/t)/t)/(t/t) \\
\mathtt{e}_2 & \mapsto & ((((t/(X\backslash((X\backslash t)/X)))/(X\backslash((X\backslash t)/X)))/U_2)/U_2)/U_2, \\
 & & ((((t/(X\backslash(t/X)))/F_{n(2,2)})/t)/t)/(t/t) \\
\mathtt{e}_3 & \mapsto & ((((t/(X\backslash((X\backslash t)/X)))/(X\backslash((X\backslash t)/X)))/U_3)/U_3)/U_3, \\
 & & ((((t/F_{n(3,1)})/(X\backslash((X\backslash(t/X))/X)))/t)/(t/t))/(t/t) \\
\mathtt{c} & \mapsto & t/(X\backslash((X\backslash t)/X)),\ X\backslash t \\
\mathtt{t} & \mapsto & t \\
\mathtt{tt} & \mapsto & t/t,\ X\backslash t \\
\mathtt{ttt} & \mapsto & (t/t)/t,\ X\backslash t \\
\mathtt{c}_1 & \mapsto & (X\backslash((X\backslash t)/X))/X,\ X\backslash t \\
\mathtt{c}_2 & \mapsto & (X\backslash((X\backslash t)/X))/X,\ X\backslash t \\
\mathtt{c}_3 & \mapsto & (X\backslash((X\backslash t)/X))/X,\ X\backslash t \\
\mathtt{v}_1 & \mapsto & (X\backslash t)/X,\ X\backslash t \\
\mathtt{v}_2 & \mapsto & (X\backslash(t/X))/X,\ X\backslash t \\
\mathtt{v}_3 & \mapsto & (X\backslash((X\backslash t)/X))/X,\ X\backslash t \\
\mathtt{v}_4 & \mapsto & (X\backslash((X\backslash(t/X))/X))/X,\ X\backslash t
\end{array}
$$

Figure A.6: The final grammar.

*Apart from the padding, there is now just one single type assigned to* `c`*. This is the only grammar with* $k = 2$ *obtainable from* $D$*. This type,* $t/(X\backslash((X\backslash t)/X))$*, corresponds to vertex 3 in* $HG$ *and, in the general form of* $D$*, is assigned to* $\mathtt{v}_3$ *in the form of* $(X\backslash((X\backslash t)/X))/X$*. This is to be interpreted as a cover consisting of just the vertex 3. As can be verified, this is indeed the unique solution for the hypergraph* $HG$ *given at the beginning of this example. Hence,* any *grammar* $G$ *in* $\mathcal{G}_{k\text{-}max\text{-}val}$*,* $k = 2$*, consistent with* $D$*, is such that there exists a substitution* $\tau$*,* $G''' \supseteq \tau[G'']$*, and a substitution* $\tau'$ *such that* $G''' \supseteq \tau'[G]$*, where* $\tau'$ *is either empty or contains only substitutions that unify (primitive) types assigned to* `x`*.*