



Incremental learning of iterated dependencies

Denis Béchet¹ · Annie Foret²

Received: 17 September 2019 / Revised: 3 December 2020 / Accepted: 7 January 2021 /

Published online: 7 March 2021

© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2021

Abstract

We study some learnability problems in the family of Categorical Dependency Grammars (CDG), a class of categorical grammars defining dependency structures. CDG is a formal system, where types are attached to words, combining the classical categorical grammars' elimination rules with valency pairing rules defining non-projective (discontinuous) dependencies; very importantly, the elimination rules are naturally extended to the so called “iterated dependencies” expressed by a specific type constructor and related elimination rules. This paper first reviews key points on negative results: even the rigid (one type per word) CDG cannot be learned neither from function/argument structures, nor even from dependency structures themselves. Such negative results prove the impossibility to define a learning algorithm for these grammar classes. Nevertheless, we show that the CDG satisfying reasonable and linguistically valid conditions on the iterated dependencies are incrementally learnable in the limit from dependency structures. We provide algorithms and also discuss these aspects for recent variants of the formalism that allow the inference of CDG from linguistic treebanks.

Keywords Grammatical inference · Categorical grammar · Dependency grammar · Incremental learning · Iterated types · Treebanks

1 Introduction

The paper studies the problem of inference of dependency grammars presented in the CDG style from positive examples of dependency trees or structures they generate.

Editors: Olghierd Unold, François Coste, Colin de la Higuera.

✉ Denis Béchet
Denis.Bechet@univ-nantes.fr

Annie Foret
Annie.Foret@irisa.fr

¹ LS2N and Univ Nantes, Nantes, France

² IRISA and Univ Rennes, Rennes, France

1.1 Generalities on dependency grammars

There is no general agreement on the notion of dependency grammar. Generally speaking, it is a grammar which represents the syntactic structure as a graph of binary relations on words in the sentence, *dependencies*, and not as a hierarchy of *syntagmas* (i.e. of phrases referred to their grammatical categories), which is the case of all syntagmatic grammars. These syntactic structure graphs called *dependency structures* (DS) are sometimes linearly ordered by the precedence order of the words in the sentence, sometimes they are considered without this order. Very often it is claimed that the DS is a tree (called *dependency tree* (DT)) and, moreover, for the ordered DT, it is often claimed that its arcs do not cross (such DS are called *projective*). For example the DT in Fig. 1 is not projective (considering only the arcs above the sentence in the DS), whereas the DS in Fig. 2 is a projective DT. In many theoretical papers the dependency relations in DS are not named, whereas in the linguistic and in the NLP literature on dependency grammars, as a rule, they are named.

The DS we consider below are actually *directed acyclic graphs* (DAG) as they are in general neither trees, nor projective, but they are linearly ordered and use named dependencies (see examples in Figs. 1, 2, 3). The dependency linguistic models (DLM) themselves are conceptually and technically diverse. Basically, there are three main formalisms: dependency grammars (DG) as *formal systems generating DS*, *constraint-based DLM* and *structure conversion DLM*. The first formal DG were defined as formal systems generating projective DT (Hays (1960); Gaifman (1965), see also a survey in Dikovskij and Modina (2000)). Among such kind of formal DG there are also the *Link Grammars* (Sleator and Temperley (1995)) and the *Categorial Dependency Grammars* (Dikovsky (2004)) considered in this paper. The constraint-based DLM define DS as models of formulae sets (cf. Maruyama (1990); Kruijff (2001); Duchier and Debusmann (2001)) or using finite automata (or regular expressions) on trees (cf. Eisner (1991)). Numerous systems belong to the third class which defines DS using a standard transformation of syntactic structures of a different nature (constituent structures, feature structures, unordered DT, etc.) defined algebraically or through an automaton or a lexicalized grammar (CF, HPSG, LTAG, LFG, CCG, etc.).

1.2 Categorial dependency grammars

The Categorial Dependency Grammars (CDG) considered in this paper is a unique class of DG directly generating unbounded DS (without tree-constraints, projectivity constraints,

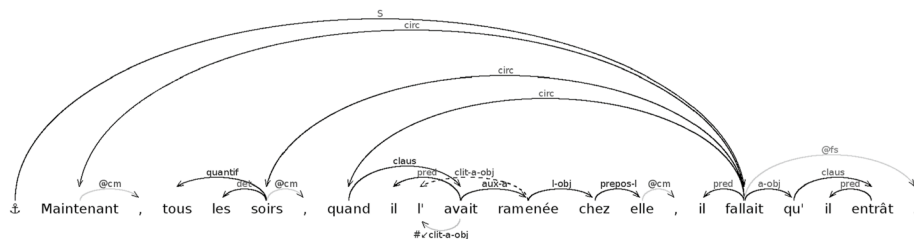


Fig. 1 Iterated circumstantial dependency.(fr. *now all the evenings when he took her home he had to enter [M: Proust])

etc.), which is well adapted to real NLP applications and is analysed in tractable polynomial in Dekhtyar et al. (2015) for grammars with bounded numbers of non-projective valency symbols.¹²

A CDG is a formal system combining the classical categorial grammars' elimination rules with valency pairing rules defining discontinuous dependencies. A special feature of CDG is that the elimination rules are interpreted as local dependency constructors. Very importantly, these rules are naturally extended to the so called “*iterated dependencies*”. This point needs explanation. A dependency d is *iterated* in a DS D if some word in D governs through dependency d several other words. For instance, in the DS in Fig. 1, the dependency *circ* is iterated because the main verb *fallait* (fr. *had to*) governs through this dependency three words: *maintenant* (fr. *now*), *soirs* (fr. *evenings*) and *quand* (fr. *when*). The iterated dependencies are due to the one of the basic principles of dependency syntax, which concerns optional repeatable dependencies (cf. Mel'čuk (1988)): All modifiers of a noun n share n as their *governor* and, similarly, all words circumstantial of a verb v share v as their *governor*. The DG incompatible with this principle, i.e., unfit for expression of iterated dependencies, cannot be considered as linguistically relevant. At the same time, as we explain below, the iterated dependencies are a challenge for grammatical inference, so as to infer a grammar that exactly generates the targeted language to which examples belong.

1.3 Grammatical inference

The idea of grammatical inference is as follows. A class of languages defined using a class of grammars \mathcal{G} is learnable if there exists a learning algorithm ϕ from finite sets of words generated by the target grammar $G_0 \in \mathcal{G}$ to hypothetical grammars in \mathcal{G} , such that for any increasing enumeration of finite sublanguages of $L(G_0)$, the sequence of output grammars in \mathcal{G} converges to a grammar in \mathcal{G} generating the target language $L(G_0)$.

This concept of *identification in the limit* is due to E.M. Gold (1967). *Learning from strings* refers to hypothetical grammars generated from finite sets of strings. More generally, the hypothetical grammars may be generated from finite sets of structures defined by the target grammar. This kind of learning is called *learning from structures*. Both concepts were intensively studied (see surveys in Angluin (1980), Kanazawa (1998), Bonato (2006) and de la Higuera (2010)). Most results are pessimistic. In particular, any family of grammars generating all finite languages and at least one infinite language (as it is the case of all classical grammars) is not learnable from strings. Other negative results have been shown in Costa Florêncio (2012, 2003); Béchet and Foret (2003, 2003). Nevertheless, due to several sufficient conditions of learnability, such as *finite elasticity* Wright (1989); Motoki et al. (1991) and *finite thickness* Shinohara (1991), some interesting positive results were obtained. In particular, for every k , k -rule string and term generating grammars are learnable from strings Shinohara (1991) and k -valued (i.e. assigning no more than k types per word) classical categorial grammars (CG) are learnable from so called “function-argument” structures and also from strings Buszkowski (1987); Buszkowski and Penn (1990); Kanazawa (1998). Other classes such as least cardinality grammars are studied in Costa Florêncio and Fernau (2012); Costa Florêncio and Fernau (2010).

¹ Theorem 8 in Dekhtyar et al. (2015) provides the time complexity of the CdgAnalyst parsing algorithm.

² A large scale wide coverage CDG of French and a general purpose deterministic parser have been implemented Dikovsky (2011); Béchet et al. (2014); Lacroix and Béchet (2014).

In Béchet et al. (2004) it was shown that, in contrast with the classical categorical grammars, the *rigid* (i.e. 1-valued) CDG are not learnable. This negative effect is due to the use of iterated types which express the iterated dependencies. On the other hand, it is also shown that the k -valued CDG with iteration-free types are learnable from the so called “*dependency nets*” (an analogue of the function-argument structures adapted to CDG) and also from strings. A similar positive result was also obtained for Link Grammars (which do not express iterated dependencies) Béchet (2003). But, as we explain above, iteration-free DG cannot be considered as an acceptable compromise. Still worse, as we prove in Sect. 3, the CDG with iterated types cannot be learned even from the DS themselves. This means that linguistically relevant CDG cannot be inferred from dependency treebanks (in which the iterated dependencies are of course not marked).

1.4 Main results of the paper

Firstly, in this paper, we follow the results presented in Béchet et al. (2010). We consider two main different approaches to CDG, in the presence of iterated types, with different underlying structured examples. The first one considers functor-argument structures that provide information in a proof-tree manner, but where no dependency name is known; in this case a bound on the number of types may be required. The second one relies on labelled dependency structures, but with a relaxed cardinality constraint. We propose a pragmatic solution of the learnability problem for CDG with iterated dependency types. It consists in limiting the family of CDG to the grammars satisfying a strong and at the same time a linguistically well-grounded condition. Intuitively, in the grammars satisfying this condition, the iterated dependencies and the dependencies repeatable at least K times for some fixed K are *indiscernible*. For example, in Fig. 1 the dependency *circ* is repeated three times: for $K = 2$ or for $K = 3$, the underlying grammar can have an iterated dependency allowing more than three repetitions. This constraint, called below *K-star revealing*, is more or less generally accepted in the traditional dependency syntax (cf. Mel’čuk (1988), where $K = 2$). For the class of K -star revealing CDG, we present an algorithm which *incrementally learns* the target CDG from dependency structures with iterated dependencies.

The K -star revealing condition is difficult to check on a grammar. On the contrary, we introduced in Béchet and Foret (2016) a syntactic criterion called *simple K-star* that can be easily checked on a grammar. In Béchet and Foret (2016) it is studied only on projective CDG. Here, an extension of this concept to the whole class of CDG (not only the projective CDG but also the CDG with non-projective dependencies) is presented and compared to the K -star revealing condition. The class of (projective or non-projective) CDG is proved also to be learnable from dependency structures.

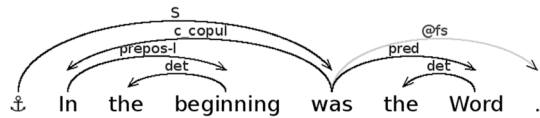
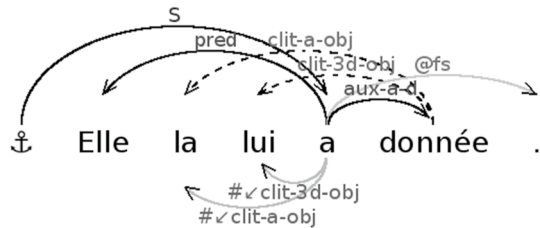
Finally, a new criterion called *global simple K-star* that introduces iterated types on a more global principle for each inferred type is studied. In fact, this variant corresponds to a different interpretation of the repetition principle. Because the learning algorithm for the class of simple K -star CDG does not produce a global simple K -star grammar, variants of this algorithm are discussed. Table 1 shows a synthetic view of the different positive results presented here.

1.5 Organization of the paper

The paper is organized as follows. Section 2 contains all background notions and facts, in particular, those concerning the Categorical Dependency Grammars and learnability from positive

Table 1 Approaches in the presence of iterated types

Structured example	Annotation	Number (k) of types per word	Repetition number (K) for indiscernibility
Functor-argument (FA, proof-tree)	Unlabelled (no dep. name)	Bound	No bound
Dependency structure (DS)	Labelled (dep. names)	No bound	Bound

Fig. 2 Projective dependency structure**Fig. 3** Non-projective dependency structure

examples. In Sect. 3, functor-argument structures for CDG are defined and several negative results concerning learnability of CDG from these structures are shown. In particular, it is shown that CDG cannot be learned even from dependency structures. In Sect. 4, the main notions of the paper are presented: those of incremental learning and of K -star-revealing CDG. In this section, the inference algorithm for the K -star revealing CDG is also presented and proved. In Sect. 5 we define and study “simple K -star” grammars with a syntactic condition. We also introduce the “global” variants. Section 6 concludes the paper.

2 Background

2.1 Categorical dependency grammars

Categorical dependency grammars may be seen as an assignment to words of first order (unnested) dependency types of the form: $t = [l_m \setminus \dots \setminus l_1 \setminus g/r_1 / \dots / r_n]^P$. Intuitively, $w \mapsto [\dots \setminus d \setminus \dots]^P$ means that the word w has a left subordinate through dependency d (similar for the right part $[\dots / d / \dots]^P$). Similarly $w \mapsto [\dots \setminus d^* \setminus \dots]^P$ means that w may have 0, 1 or several left subordinates through dependency d . The *head type* g in $w \mapsto [\dots \setminus g / \dots]^P$ means

that w is governed through dependency g . For instance, the assignment of Example 1 determines the projective dependency structure in Fig. 2.

<i>in</i>	$\mapsto [c_copul/prepos-l]$
<i>the</i>	$\mapsto [det]$
<i>beginning</i>	$\mapsto [det\backslash prepos-l]$
<i>was</i>	$\mapsto [c_copul\backslash S/@fs/pred]$
<i>word</i>	$\mapsto [det\backslash pred]$
.	$\mapsto [@fs]$

Example 1

In Fig. 2, a pseudo-dependency S links an anchor and the main word of the dependency structure that has the head type S . Not essential dependencies that end on punctuation marks start by the symbol $@$ like $@fs$ for the full stop. For the verb *was*, the type $[c_copul\backslash S/@fs/pred]$ means that the word has a left subordinate through dependency c_copul and two right subordinates through dependencies $@fs$ and $pred$. It implies the right subordinate through dependency $pred$ is on the left of the right subordinate through dependency $@fs$: In types, the lists of right and left dependency names appear in the reverse order as the order of the right and left subordinates.

The intuitive meaning of part P , called *potential*, is that it defines *discontinuous* dependencies of the word w . P is a string of *polarized valencies*, i.e. of symbols of four kinds: $\swarrow d$ (left negative valency d), $\searrow d$ (right negative valency d), $\nwarrow d$ (left positive valency d), $\nearrow d$ (right positive valency d). Intuitively, $v = \nwarrow d$ requires a subordinate through dependency d situated *somewhere* on the left, whereas the *dual* valency $\check{v} = \swarrow d$ requires a governor through the same dependency d situated *somewhere* on the right. So together they describe the discontinuous dependency d . Similarly for the other pairs of dual valencies. For negative valencies $\swarrow d$, $\searrow d$ a special kind of types $\#(\swarrow d)$, $\#(\searrow d)$ is provided. Intuitively, they serve to check the adjacency of a distant word subordinate through discontinuous dependency d to a *host word*. The dependencies of these types are called *anchor*. For instance, the assignment of Example 2 determines the non-projective DS in Fig. 3.

<i>elle</i>	$\mapsto [pred]$
<i>la</i>	$\mapsto [\#(\swarrow clit-a-obj)]^{\swarrow clit-a-obj}$
<i>lui</i>	$\mapsto [\#(\swarrow clit-3d-obj)]^{\swarrow clit-3d-obj}$
<i>a</i>	$\mapsto [\#(\swarrow clit-3d-obj)\nwarrow \#(\swarrow clit-a-obj)\backslash pred\backslash S/@fs/aux-a-d]$
<i>donnée</i>	$\mapsto [aux-a-d]^{\nwarrow clit-3d-obj}\nwarrow clit-a-obj$
.	$\mapsto [@fs]$

Example 2

In Fig. 3, there are two non-projective dependencies. The dependency $clit-a-obj$ links the past participle *donnée* to the (accusative) clitic *la* ($it_{g=fem}$). The dependency $clit-3d-obj$ links *donnée* to the (3rd case dative) clitic *lui* (*to him*). Each non-projective dependency is associated to an anchor that is shown below the words: the anchor $\#(\swarrow clit-a-obj)$ for the dependency $clit-a-obj$ and the anchor $\#(\swarrow clit-3d-obj)$ for the dependency $clit-3d-obj$. For the clitic *la*, the head type is $\#(\swarrow clit-a-obj)$ and the left negative valency is $\swarrow clit-a-obj$. It means that the word must be the (left or right) subordinate of a word through an anchor $\#(\swarrow clit-a-obj)$ and in the same time must be the left subordinate of another word through the non-projective dependency $clit-a-obj$. The auxiliary verb *a* (*has*) has two left subordinates through the anchors $\#(\swarrow clit-a-obj)$ and $\#(\swarrow clit-3d-obj)$. The order of subtypes in the type of the auxiliary verb means that the clitic *la* must be before the clitic *lui* in the linear order of the sentence. The left positive valencies $\nwarrow clit-3d-obj$ and $\nwarrow clit-a-obj$ mean that the past participle *donnée* has two

left subordinates through the non-projective dependencies *clit*–*3d*–*obj* and *clit*–*a*–*obj*. For a word, the order of polarized valencies is not important and does not imply an order of the subordinates. In fact, the order of subordinates is specified by anchors. Anchors are very important because they fix the places where the subordinates of a non-projective dependency can be inserted in the linear order. The polarized valencies are important to define the real non-projective dependency. A positive valency (on the governor) and negative valency (on the subordinate) create a non-projective dependency when the link corresponds to the first available principles for a valency (the name of the dependency) and orientation (from left to right or from right to left). For instance, the non-projective dependency *clit*–*3d*–*obj* between *donnée* and *lui* is a (right to) left non-projective dependency with the valency *clit*–*3d*–*obj*. The first available principle is applied here for left *clit*–*3d*–*obj* independently to any other right non-projective dependency or to any other left non-projective dependency with another valency. This is the case for the left non-projective dependency *clit*–*a*–*obj* between *donnée* and *la*. The kind of links in CDG is also reminiscent of polarized axiom links in proof nets as in linear logic and in Lambek grammars analysis, which by the way can also be used as structures for learning categorical grammars Bonato and Retoré (2014).

Definition 1 (CDG dependency structures) Let $W = a_1 \dots a_n$ be a list of symbols and $\{d_1, \dots, d_m\}$ be a set of *dependency names*, with their dependency nature that can be either local, discontinuous or anchor. A structure $D = (W, E)$, viewed as a graph of occurrences of the symbols in W , with the labeled arcs E is a *dependency structure* (DS) of W if it has a *root*, i.e. a node $a_i \in W$ such that (i) for any node $a \in W$, $a \neq a_i$, there is a path from a_i to a and (ii) there is no arc (a', d, a_i) .³ An arc $(a, d, a') \in E$ is called *dependency* d from a to a' . The symbol a is called a *governor* of a' and a' is called a *subordinate* of a through d . The linear order on W is the *precedence order* on D .

Definition 2 (CDG types) Let \mathbf{C} be a set of *local dependency names* and \mathbf{V} be a set of *valency names*.

The expressions of the form $\swarrow v$, $\searrow v$, $\backslash v$, $\nearrow v$, where $v \in \mathbf{V}$, are called *polarized valencies*. $\searrow v$ and $\nearrow v$ are *positive*, $\swarrow v$ and $\backslash v$ are *negative*; $\searrow v$ and $\swarrow v$ are *left*, $\nearrow v$ and $\backslash v$ are *right*. Two polarized valencies with the same valency name and orientation, but with opposite arrow directions are *dual*.

An expression of one of the forms $\#(\swarrow v)$, $\#(\searrow v)$, $v \in \mathbf{V}$, is called an *anchor type* or just an *anchor*. An expression of the form d^* where $d \in \mathbf{C}$, is called an *iterated dependency type*.

Local dependency names, iterated dependency types and anchor types are *primitive types*.

An expression of the form $t = [l_m \backslash \dots \backslash l_1 \backslash H / r_1 / \dots / r_n]$ in which $m, n \geq 0$, $l_1, \dots, l_m, r_1, \dots, r_n$ are primitive types and H is either a local dependency name or an anchor type, is called a *basic dependency type*. l_1, \dots, l_m and r_1, \dots, r_n are left and right *argument types* of t , H is called the *head type* of t . We may write $t = H$ instead of $[H]$, when t has no argument type.

A (possibly empty) string P of polarized valencies is called a *potential*.

³ Evidently, every DS is connected and has a unique root.

A *dependency type* is an expression B^P in which B is a basic dependency type and P is a potential. $\mathbf{CAT}(\mathbf{C}, \mathbf{V})$ will denote the set of all dependency types over \mathbf{C} and \mathbf{V} .

CDG are defined using the following calculus of dependency types.⁴ These rules are relativized with respect to the word positions in the sentence, which allows us to interpret them as rules of construction of DS. Namely, when a type $B^{v_1 \dots v_k}$ is assigned to the word in a position i , we encode it using the *state* $(B, i)^{(v_1, i) \dots (v_k, i)}$. In these rules, states must be adjacent.

Definition 3 (Relativized calculus of dependency types) In the following set of rules on lists of states, the symbol C stands for a local dependency name or an anchor type, but cannot be an anchor in rules \mathbf{I}^1 and $\mathbf{\Omega}^1$ (anchors are not iterated); The symbol β ranges over expressions of the form $l_m \setminus \dots \setminus l_1 \setminus H/r_1 / \dots / r_n$

$$\begin{aligned} \mathbf{L}^1. & ([C], i_1)^{P_1} ([C \setminus \beta], i_2)^{P_2} \vdash ([\beta], i_2)^{P_1 P_2} \\ \mathbf{I}^1. & ([C], i_1)^{P_1} ([C^* \setminus \beta], i_2)^{P_2} \vdash ([C^* \setminus \beta], i_2)^{P_1 P_2} \\ \mathbf{\Omega}^1. & ([C^* \setminus \beta], i)^P \vdash ([\beta], i)^P \\ \mathbf{D}^1. & ([\beta], i)^{P_1 (\swarrow v, i_1) P (\searrow v, i_2) P_2} \vdash ([\beta], i)^{P_1 P P_2}, \end{aligned}$$

if $i_1 < i_2$ (non-internal constraint)⁵ and
if the potential $(\swarrow v, i_1) P (\searrow v, i_2)$ satisfies
the following pairing rule **FA**:

FA (*first available*): P has no occurrences of $(\swarrow v, i)$ or $(\searrow v, i)$ for any i

The rule \mathbf{L}^1 is the classical elimination rule. Eliminating the argument type $C \neq \#(\alpha)$ constructs the (*projective*) dependency C and concatenates the potentials. The type $C = \#(\alpha)$ creates an *anchor dependency*. The rule \mathbf{I}^1 derives $k > 0$ instances of C . The rule $\mathbf{\Omega}^1$ serves in particular for the case $k = 0$. \mathbf{D}^1 creates *discontinuous dependencies*. It pairs and eliminates dual valencies with name C satisfying the rule **FA** to create the discontinuous dependency C .

Now, in this relativized calculus, for every proof ρ represented as a sequence of rule applications, we may define the DS $DS_x(\rho)$ constructed in this proof. Namely, let us consider the calculus relativized with respect to a sentence x with the set of word occurrences W . Then $DS_x(\varepsilon) = (W, \emptyset)$ is the DS constructed in the empty proof $\rho = \varepsilon$. Now, let (ρ, R) be a nonempty proof with respect to x and $(W, E) = DS_x(\rho)$. Then $DS_x((\rho, R))$ is defined as follows:

If $R = \mathbf{L}^1$ or $R = \mathbf{I}^1$, then $DS_x((\rho, R)) = (W, E \cup \{(a_{i_2}, C, a_{i_1})\})$. When C is a local dependency name, the new dependency is *local*. In the case where C is an anchor, this is an *anchor dependency*.

If $R = \mathbf{\Omega}^1$, then $DS_x((\rho, R)) = DS_x(\rho)$.

If $R = \mathbf{D}^1$, then $DS_x((\rho, R)) = (W, E \cup \{(a_{i_2}, C, a_{i_1})\})$ and the new dependency is *discontinuous*.

⁴ We show left-oriented rules. The right-oriented are symmetrical.

⁵ This constraint disallows internal primitive loops because a dependency must link two different words.

$$\begin{array}{c}
\frac{\frac{[\#(\swarrow \text{clit} - 3d - \text{obj})]^{\swarrow \text{clit} - 3d - \text{obj}} [\#(\swarrow \text{clit} - 3d - \text{obj}) \setminus \#(\swarrow \text{clit} - a - \text{obj}) \setminus \text{pred} \setminus S / aux - a - d]}{[\#(\swarrow \text{clit} - a - \text{obj})]^{\swarrow \text{clit} - a - \text{obj}}} (L^1)}{[\text{pred}]} \\
\frac{\frac{[\text{pred} \setminus S / aux - a - d]^{\swarrow \text{clit} - a - \text{obj} \swarrow \text{clit} - 3d - \text{obj}}} { [S / aux - a - d]^{\swarrow \text{clit} - a - \text{obj} \swarrow \text{clit} - 3d - \text{obj}}} (L^1)}{[S]^{\swarrow \text{clit} - a - \text{obj} \swarrow \text{clit} - 3d - \text{obj} \swarrow \text{clit} - 3d - \text{obj} \swarrow \text{clit} - a - \text{obj}}} (L^r) \\
\hline
S \quad (D' \times 2)
\end{array}$$

Fig. 4 Dependency structure correctness proof

We may consider the following simpler version, suppressing word positions, when the context is clear or in examples. Note that the non-internal constraint⁵ is not expressed but left implicit.

Definition 4 (Simplified calculus of dependency types) In this set of rules⁴ on lists of types, the symbol C stands for a local dependency name or an anchor type, but cannot be an anchor in rules **I**¹ and **Q**¹ (anchors are not iterated); The symbol β ranges over expressions of the form $l_m \setminus \dots \setminus l_1 \setminus H / r_1 / \dots / r_n$

$$\begin{array}{ll}
\mathbf{L}^1. & [C]^{P_1} [C \setminus \beta]^{P_2} \vdash [\beta]^{P_1 P_2} \\
\mathbf{I}^1. & [C]^{P_1} [C^* \setminus \beta]^{P_2} \vdash [C^* \setminus \beta]^{P_1 P_2} \\
\mathbf{Q}^1. & [C^* \setminus \beta]^P \vdash [\beta]^P \\
\mathbf{D}^1. & [\beta]^{P_1 (\swarrow v) P (\swarrow v) P_2} \vdash [\beta]^{P_1 P P_2},
\end{array}$$

if the potential $(\swarrow v)P(\swarrow v)$ satisfies the following pairing rule **FA**:

FA (*first available*): P has no occurrences of $\swarrow v$ or $\swarrow v$.

For a proof ρ as a sequence of rule applications, $DS(\rho)$ denotes the DS constructed in this proof.

This simplified calculus enjoys a subformula property adapted to CDG types:

- each type formula without bracket and potential that occurs on the right on a rule (β , α , $C^* \setminus \beta$) also occurs on the left of the same rule;
- each potential expression on the right of a rule also occurs on the left of the same rule.

Definition 5 (Categorical Dependency Grammar) A *Categorical Dependency Grammar* (CDG) is a system $G = (W, \mathbf{C}, \mathbf{V}, S, \lambda)$, where W is a finite set of words, \mathbf{C} is a finite set of local dependency names containing the selected name S (an axiom), \mathbf{V} is a finite set of discontinuous dependency names and λ , called *lexicon*, is a finite substitution on W such that $\lambda(a) \in \mathbf{CAT}(\mathbf{C}, \mathbf{V})$ for each word $a \in W$. λ is extended on sequences of words W^* in the usual way.⁶

For $G = (W, \mathbf{C}, \mathbf{V}, S, \lambda)$, a DS D and a sentence x , let $G[D, x]$ denote the relation:

“ $D = DS_x(\rho)$, where ρ is a proof of $\Gamma \vdash S$ for some $\Gamma \in \lambda(x)$ ”. Then the *language* generated by G is the set $L(G) =_{df} \{w \mid \exists D \ G[D, w]\}$ and the *DS-language* generated by G is

⁶ $\lambda(a_1 \dots a_n) = \{t_1 \dots t_n \mid t_1 \in \lambda(a_1), \dots, t_n \in \lambda(a_n)\}$.

the set $\Delta(G) =_{df} \{D \mid \exists w G[D, w]\}$. $\mathcal{D}(CDG)$ and $\mathcal{L}(CDG)$ will denote the families of DS-languages and languages generated by these grammars.

Example 3 For instance, the proof in Fig. 4 shows that the DS in Fig. 3 belongs to the DS-language generated by a grammar containing the type assignments shown above for the French sentence *Elle la lui a donnée*.

CDG are very expressive. Evidently, they generate all CF-languages. They can also generate non-CF languages. For instance, the CDG of Example 4 generates the language $\{a^n b^n c^n \mid n > 0\}$ Dikovsky (2004).⁷

$$\begin{aligned} a &\mapsto \#(\swarrow A)^{\swarrow A}, [\#(\swarrow A) \setminus \#(\swarrow A)]^{\swarrow A} \\ b &\mapsto [B/C]^{\swarrow A}, [\#(\swarrow A) \setminus S/C]^{\swarrow A} \\ c &\mapsto C, [B \setminus C] \end{aligned}$$

Example 4

Seemingly, the family $\mathcal{L}(CDG)$ of CDG-languages is different from that of the mildly context sensitive languages Joshi et al. (1991); Shanker and Weir (1994) generated by multi-component TAG, linear CF rewrite systems and some other grammars. $\mathcal{L}(CDG)$ contains non-TAG languages, e.g. $L^{(m)} = \{a_1^n a_2^n \dots a_m^n \mid n \geq 1\}$ for all $m > 0$. In particular, it contains the language $MIX = \{w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c\}$ B  chet et al. (2005), for which E. Bach had conjectured that it is not mildly CS, but later shown to be 2-MCFL Nederhof (2016); Salvati (2015). On the other hand, Dekhtyar and Dikovsky (2004) conjectures that this family does not contain the TAG language $L_{copy} = \{xx \mid x \in \{a, b\}^+\}$. This comparison shows a specific nature of the valencies' pairing rule **FA**. It can be expressed in terms of valencies' bracketing. For this, one should interpret $\swarrow d$ and $\nearrow d$ as *left brackets* and $\searrow d$ and $\nwarrow d$ as *right brackets*. A potential is *balanced* if it is well-bracketed for each couple of positive and negative brackets independently to the other couples of positive and negative brackets.

CDG have an important property formulated in terms of two images of sequences of types γ :

Definition 6 (Local and valency projections) For a CDG G with lexicon λ the *local projection* $\|\gamma\|_l$ and the *valency projection* $\|\gamma\|_v$ are defined as follows:

1. $\|\varepsilon\|_l = \|\varepsilon\|_v = \varepsilon$; $\|\alpha\gamma\|_l = \|\alpha\|_l \|\gamma\|_l$ and $\|\alpha\gamma\|_v = \|\alpha\|_v \|\gamma\|_v$ for a type α .
2. $\|C^P\|_l = C$ et $\|C^P\|_v = P$ for every type C^P .

Theorem 1 Dekhtyar and Dikovsky (2004, 2008) For a CDG G with lexicon λ and a string x , $x \in L(G)$ iff there is $\Gamma \in \lambda(x)$ such that $\|\Gamma\|_l$ is reduced to S without the rule **D** and $\|\Gamma\|_v$ is balanced.

On this property resides a polynomial time parsing algorithm for CDG Dekhtyar and Dikovsky (2004, 2008).

⁷ One can see that a DS is not always a tree.

2.2 Learnability, finite elasticity and limit points

An *observation set* $\Phi(G)$ of G is associated with every grammar $G \in \mathcal{C}$. This may be the generated language $L(G)$ or an image of the constituent or dependency structures generated by G .

Definition 7 (Inference algorithm) Below we call an enumeration of $\Phi(G)$ a *training sequence* for G . An algorithm \mathcal{A} is an *inference algorithm* for \mathcal{C} if, for every grammar $G \in \mathcal{C}$, the algorithm \mathcal{A} applies to its training sequences σ of $\Phi(G)$ and, for every initial subsequence $\sigma[i] = \{s_1, \dots, s_i\}$ of σ , it returns a *hypothesized grammar* $\mathcal{A}(\sigma[i]) \in \mathcal{C}$. The algorithm \mathcal{A} *learns* a *target grammar* $G \in \mathcal{C}$ if on any training sequence σ for G \mathcal{A} stabilizes on a grammar $\mathcal{A}(\sigma[T]) \equiv G$.⁸ The grammar $\lim_{i \rightarrow \infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ returned at the stabilization step is the *limit grammar*. The algorithm \mathcal{A} *learns* \mathcal{C} if it learns every grammar in \mathcal{C} . \mathcal{C} is *learnable* if there is an inference algorithm learning \mathcal{C} .

Learnability and unlearnability properties have been widely studied from a theoretical point of view. In particular Wright (1989); Motoki et al. (1991) introduced finite elasticity, a property of classes of languages implying their learnability. The following elegant presentation of this property is cited from Kanazawa (1998).

Definition 8 (Finite elasticity) A class \mathcal{L} of languages has *infinite elasticity* iff $\exists (e_i)_{i \in \mathbb{N}}$ an infinite sequence of sentences, $\exists (L_i)_{i \in \mathbb{N}}$ an infinite sequence of languages of \mathcal{L} such that $\forall i \in \mathbb{N} : e_i \notin L_i$ and $\{e_0, \dots, e_{i-1}\} \subseteq L_i$. A class has *finite elasticity* iff it doesn't have infinite elasticity.

Theorem 2 (Wright 1989) *A class that is not learnable has infinite elasticity.*

Corollary 1 *A class that has finite elasticity is learnable.*

The finite elasticity can be extended from a class to every class obtained by a *finite-valued relation*⁹. We use here a version of the theorem that was proved in Kanazawa (1998) and is useful for various kinds of languages (strings, structures, nets) that can be described by lists of elements over some alphabets.

Theorem 3 (Kanazawa 1998) *Let \mathcal{L} be a class of languages over Γ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Gamma^*$ be a finite-valued relation. Then the class of languages $\{R^{-1}[L] = \{s \in \Sigma^* \mid \exists u \in L \wedge (s, u) \in R\} \mid L \in \mathcal{L}\}$ has finite elasticity.*

Definition 9 (Limit points) A class \mathcal{L} of languages has a *limit point* iff there exists an infinite sequence $(L_n)_{n \in \mathbb{N}}$ of languages in \mathcal{L} and a language $L \in \mathcal{L}$ such that: $L_0 \subsetneq L_1 \subsetneq \dots \subsetneq L_n \subsetneq \dots$ and $L = \bigcup_{n \in \mathbb{N}} L_n$ (L is a *limit point* of \mathcal{L}).

⁸ \mathcal{A} *stabilizes* on σ on step T means that T is the minimal number t for which there is no $t_1 > t$ such that $\mathcal{A}(\sigma[t_1]) \neq \mathcal{A}(\sigma[t])$.

⁹ A relation $R \subseteq \Sigma^* \times \Gamma^*$ is finite-valued iff for every $s \in \Sigma^*$ there are at most finitely many $u \in \Gamma^*$ such that $(s, u) \in R$.

Limit Points Imply non-effective Unlearnability. If the languages of the grammars in a class \mathcal{G} have a limit point then the class \mathcal{G} is *unlearnable*.¹⁰

2.3 Limit points for CDGs with iterated types

In B  chet et al. (2004) it is shown that, in contrast with the classical categorial grammars, the *rigid* (i.e. 1-valued) CDG are not learnable. This negative result is due to the use of iterated types. We recall the limit point construction of B  chet et al. (2004) concerning iterative types and discuss it later.

Limit Point Construction.

Lemma 10 *Let S, A, B be three local dependency names. Grammars G'_n, G'_* are defined as follows¹¹:*

$$\begin{aligned} C'_0 &= S \\ C'_{n+1} &= C'_n / A^* / B^* \\ G'_0 &= \{a \mapsto A, b \mapsto B, c \mapsto C'_0\} \\ G'_n &= \{a \mapsto A, b \mapsto B, c \mapsto [C'_n]\} \\ G'_* &= \{a \mapsto A, b \mapsto A, c \mapsto [S/A^*]\} \end{aligned}$$

These constructions yield a limit point as follows (B  chet et al. 2004).

$$L(G'_n) = \{c(b^*a^*)^k \mid k \leq n\} \text{ and } L(G'_*) = c\{b, a\}^*$$

Proof Only three rules apply to G'_n, G'_* : **L**. (local dependency rule), **I**. and **Ω** . (ω -dependency rules), all of them enjoying the subformula property.

- $L(G'_0)$: (1) it clearly contains c (assigned to S) and (2) only c since no rule applies to $\{A, B, S\}$.
- $L(G'_n)$ ($n > 0$). We have $D'_n = C'_{n-1} / A^*$ and $C'_n = D'_n / B^*$.
 - (1) For $w \in \{c(b^*a^*)^k \mid k \leq n\}$, we have $w \in L(G'_n)$ by :
 - $[C'_n]B\Delta \vdash [C'_n]\Delta$ and $[D'_n]A\Delta \vdash [D'_n]\Delta$ (by **I**. rule) and $[C'_n] \vdash [D'_n] \vdash [C'_{n-1}]$ (by **Ω** . rule)
 - (2) Let $w' \in L(G'_n)$. We observe that w' cannot start with an a or a b (an A or B on the left part of a type could not disappear, due to the use of right constructors only); and w' cannot contain several c (no S under a constructor) ; thus $w' = cw''$, where $w'' \in \{b, a\}^*$. We get $w' \in \{c(b^*a^*)^k \mid k \leq n\}$, from the following assertion proved by induction on n and on the length of types Γ for words $w \in \{b, a\}^*$: (i) if $[C'_n]\Gamma \vdash S$, then $w \in \{(b^*a^*)^k \mid k \leq n\}$ and (ii) if $[D'_{n+1}]\Gamma \vdash S$ then $w \in \{a^*(b^*a^*)^k \mid k \leq n\}$.
 - $n = 0$, (i) is clear from $L(G'_0) = \{c\}$.
 - (ii), if $\Gamma = B\Gamma'$, we get the first step (i) with the only possibility of $[D'_{n+1}]B\Gamma' \vdash [C'_n]B\Gamma'$.
 - (ii), if $\Gamma = A\Gamma'$, we have two possibilities $[D'_{n+1}]A\Gamma' \vdash [C'_n]A\Gamma'$ or $[D'_{n+1}]A\Gamma' \vdash [D'_{n+1}]\Gamma'$.
 - (i), if $\Gamma = A\Gamma'$, we get the first step with the only possibility of $[C'_n]A\Gamma' \vdash [D'_n]A\Gamma'$.

¹⁰ This implies that the class has infinite elasticity.

¹¹ We may write $t = H$ instead of $[H]$, when t has no argument type (see Definition 2).

- (i), if $\Gamma = B\Gamma'$, we have two possibilities $[C'_n]B\Gamma' \vdash [D'_n]B\Gamma'$ or $[C'_n]B\Gamma' \vdash [C'_n]\Gamma'$.

This implies (i), (ii) by induction on n or a shorter type.

- $L(G'_*)$: (1) it clearly contains $c\{b, a\}^*$ using $[S/A^*]A\Delta \vdash S\Delta$ (I. rule) and $[S/A^*] \vdash S$ (Ω. rule)
 (2) $w' \in L(G'_*)$ has exactly one c (at least one to provide S , and no more, as explained above for G'_n); it cannot start with a (otherwise a type part would remain before S). Therefore, $w' \in c\{b, a\}^*$.

□

Theorem 11 *The constructions show the non-learnability from strings for the classes of (rigid) grammars allowing iterative types (A^*).*

We observe that in these constructions, the number of iterative types (A^*) is unbounded.

3 Learnability from positive examples

Below we study the problem of learning CDG from positive examples of structures analogous to the FA-structures used for learning the classical categorial grammars.

3.1 Original algorithm on functor-argument data

FA Structures. Let Σ be an alphabet. An *FA structure* over Σ is a binary tree where each leaf is labelled by an element of Σ and each internal node is labelled by the name of the binary rule.

Background - RG Algorithm. We recall Buszkowski's Algorithm Buszkowski and Penn (1990), called RG in Kanazawa (1998) (see also Moot and Retoré (2012)), where it is defined for the classical categorial grammars with the rules $/_e$ and \backslash_e (binary elimination rules, like the local rules of CDG \mathbf{L}^r and \mathbf{L}^l , without potentials):

$$/_e : [A/B] [B] \Rightarrow [A] \quad \text{and} \quad \backslash_e : [B] [B \backslash A] \Rightarrow [A]$$

The RG algorithm takes a set D of functor-argument structures as positive examples and returns a rigid grammar $RG(D)$ compatible with the input if there is one (compatible means that D is in the set of functor-argument structures generated by the grammar).

Sketch of RG-algorithm, computing $RG(D)$:

1. assign S to the root of each structure
2. assign distinct variables to argument nodes
3. compute the other types on functor nodes according to $/_e$ and \backslash_e
4. collect the types assigned to each symbol, this provides $GF(D)$
5. unify (classical unification) the types assigned to the same symbol in $GF(D)$, and compute the most general unifier σ_{mgu} of this family of types.
6. The algorithm fails if unification fails, otherwise the result is the application of σ_{mgu} to the types of $GF(D)$: $RG(D) = \sigma_{mgu}(GF(D))$.

3.2 Functor-argument structures for CDG with iterated types

The **functor-argument structure** and **labelled functor-argument structure** associated with a dependency structure proof, are obtained as below.

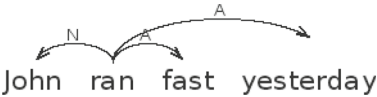
Definition 12 Let ρ be a dependency structure proof, ending in a type t . The **functor-argument structure** associated to ρ , denoted $fa_{iter}(\rho)$, is defined by induction on the length of the dependency proof ρ considering the last rule in ρ .

- if ρ has no rule, then it is reduced to a type t assigned to a word w , let $fa_{iter}(\rho) = w$
- if the last rule is: $c^{P_1} [c \setminus \beta]^{P_2} \vdash [\beta]^{P_1 P_2}$, by induction let ρ_1 be a dependency structure proof for c^{P_1} and $\mathcal{T}_1 = fa_{iter}(\rho_1)$ and let ρ_2 be a dependency structure proof for $[c \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = fa_{iter}(\rho_2)$: then $fa_{iter}(\rho)$ is the tree with root labelled by \mathbf{L}^1 and subtrees $\mathcal{T}_1, \mathcal{T}_2$.
- if the last rule is: $[c^* \setminus \beta]^{P_2} \vdash [\beta]^{P_2}$, by induction let ρ_2 be a dependency structure proof for $[c^* \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = fa_{iter}(\rho_2)$: then $faI(\rho)$ is \mathcal{T}_2 .
- if the last rule is: $c^{P_1} [c^* \setminus \beta]^{P_2} \vdash [c^* \setminus \beta]^{P_1 P_2}$, by induction let ρ_1 be a dependency structure proof for c^{P_1} and $\mathcal{T}_1 = fa_{iter}(\rho_1)$ and let ρ_2 be a dependency structure proof for $[c^* \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = fa_{iter}(\rho_2)$: $fa_{iter}(\rho)$ is the tree with root labelled by \mathbf{L}^1 and subtrees $\mathcal{T}_1, \mathcal{T}_2$.
- we define similarly the function fa_{iter} when the last rule is on the right, using $/$ and \mathbf{L}^r instead of \setminus and \mathbf{L}^1 .
- if the last rule is the one with potentials, $faI(\rho)$ is taken as the image of the proof above.

The functor-argument structure $fa_{iter}(\rho)$ is obtained from $\mathcal{J}fa_{iter}(\rho)$ (the labelled one) by erasing the labels $[c]$.

Example 5 Let $\lambda(John) = N$, $\lambda(ran) = [N \setminus S / A^*]$, $\lambda(yesterday) = \lambda(fast) = A$, then $s_3 = \mathbf{L}^1(John, \mathbf{L}^r(\mathbf{L}^r(ran, fast), yesterday))$ are associated to ρ_1 below :

$$\rho_1 : \frac{\frac{\frac{N}{S} \quad \frac{[N \setminus S / A^*]}{[N \setminus S]} \mathbf{L}^1}{[N \setminus S / A^*]} \mathbf{L}^r}{[N \setminus S / A^*]} \mathbf{L}^r \quad \frac{[N \setminus S / A^*] \quad A}{[N \setminus S / A^*]} \mathbf{I}^r \quad \frac{A}{[N \setminus S / A^*]} \mathbf{I}^r$$



(dependency structure)

3.3 On RG-like algorithms and iteration

In this subsection, we discuss that an RG-like algorithm cannot converge when applied on rigid grammars where the positive examples are functor-argument structures (without dependency names).

By *flat types*, we mean types without embedded operators. A grammar is said flat if it assigns only flat types as in next example. A *flat structure* denotes a structure associated with a proof that has only flat types.

Example 6 We consider the following functor-argument structures :

$$\begin{aligned}s_1 &= \mathbf{L}^1(\text{John}, \text{ran}) \\ s_2 &= \mathbf{L}^1(\text{John}, \mathbf{L}^r(\text{ran}, \text{fast})) \\ s_3 &= \mathbf{L}^1(\text{John}, \mathbf{L}^r(\mathbf{L}^r(\text{ran}, \text{fast}), \text{yesterday})) \\ s_4 &= \mathbf{L}^1(\text{John}, \mathbf{L}^r(\mathbf{L}^r(\mathbf{L}^r(\text{ran}, \text{fast}), \text{yesterday}), \text{nearby}))\end{aligned}$$

An RG-like algorithm could compute the following assignments and grammar from $\{s_1, s_2, s_3\}$:

$$\begin{aligned}\mathbf{L}^1(\text{John} : X_1, \text{ran} : X_1 \setminus S) : S \\ \mathbf{L}^1(\text{John} : X'_1, \mathbf{L}^r(\text{ran} : X'_1 \setminus S / X_2, \text{fast} : X_2) : X'_1 \setminus S) : S \\ \mathbf{L}^1(\text{John} : X''_1, \mathbf{L}^r(\mathbf{L}^r(\text{ran} : X''_1 \setminus S / X'_2 / X'_2, \text{fast} : X'_2) : X''_1 \setminus S / X'_2, \\ \text{yesterday} : X'_2) : X''_1 \setminus S) : S\end{aligned}$$

	general form	unification	flat rigid grammar for 2-iteration
<i>John</i>	X_1, X'_1, X''_1	$X_1 = X'_1 = X''_1$	X_1
<i>ran</i>	$X_1 \setminus S$ $X'_1 \setminus S / X_2$ $X''_1 \setminus S / X'_2 / X'_2$	<i>fails</i>	$X_1 \setminus S / X_2^*$ with $X_2 = X'_2 = X''_2$
<i>fast</i>	X_2, X'_2	X_2	X_2
<i>yesterday</i>	X'_2	X'_2	X_2

In the third column unification (without iterated types) fails for *ran* ; in the last column, we sketch a possible type (with iterated types) if a generalized unification is used instead.

Notice that the next annotated sentence s_4 would not change the type of *ran*.

In fact, when applied to rigid grammars in the case where the positive examples are functor-argument structures (without dependency names), such an RG-like algorithm **can-not converge** (in the sense of Gold).

This can be seen, as explained below, using the same grammars as in the limit point construction for string languages in Béchet et al. (2004) and as in Definition 10, involving iterated dependency types. In fact, the functor-argument structures are all flat structures, with only / operators.

$$\begin{aligned}C'_0 &= S & G'_0 &= \{a \mapsto A, b \mapsto B, c \mapsto C'_0\} \\ C'_{n+1} &= C'_n / A^* / B^* & G'_n &= \{a \mapsto A, b \mapsto B, c \mapsto [C'_n]\} \\ & & G'_* &= \{a \mapsto A, b \mapsto A, c \mapsto [S / A^*]\}\end{aligned}$$

Positive structured examples are then of the form :

$$c, \mathbf{L}^r(c, b), \mathbf{L}^r(\mathbf{L}^r(c, b), b), \mathbf{L}^r(c, a), \mathbf{L}^r(\mathbf{L}^r(c, a), a), \mathbf{L}^r(\mathbf{L}^r(c, b), a), \dots$$

Definition 13 We define $\text{flat}_{\mathbf{L}^r}$ and $\text{flat}_{\mathbf{L}^r[A]}$ on words by :

$$\text{flat}_{\mathbf{L}^r}(x1) = x1 = \text{flat}_{\mathbf{L}^r[A]}(x1) \text{ for words of length 1, and}$$

$flat_{\mathbf{L}^r}(x1.w1) = \mathbf{L}^r(x, flat_{\mathbf{L}^r}(w1))$;
 $flat_{\mathbf{L}^r[A]}(x1.w1) = \mathbf{L}^r_{[A]}(x, flat_{\mathbf{L}^r[A]}(w1))$;
 we extend the notation $flat_{\mathbf{L}^r}$ and $flat_{\mathbf{L}^r[A]}$ to sets of words (as the set of word images).

Let $FL(G)$ denote the language of functor-arguments structures of G . We use the same grammars as in the limit point construction for string languages in Sect. 2.3:

Lemma 14 $FL(G'_n) = flat_{\mathbf{L}^r}(\{c(b^*a^*)^k \mid k \leq n\})$ and $FL(G'_*) = flat_{\mathbf{L}^r}(c\{b, a\}^*)$

This property follows from the similar property on string language in Sect. 2.3, and the above remark that the functor-argument structures are all flat structures, with only / operators.

Theorem 15 $FL(G'_n)$ and $FL(G'_*)$ define a limit point that establishes the non-learnability from functor-argument structures for the underlying classes of grammars (rigid or k -valued) allowing iterated dependency types (A^*) with at least three words.

These impossibility results are to be contrasted with the case without iterated dependency types (A^*).

3.4 A limit point for labelled functor-arguments structures

If we drop restrictions such as k -valued, and consider learnability from labelled functor-arguments structures, we have a limit point as follows :

$$\begin{aligned}
 C_0 &= S \\
 C_{n+1} &= C_n/A \\
 G_0 &= \{a \mapsto A, c \mapsto C_0\} \\
 G_n &= \{a \mapsto A, c \mapsto [C_n], c \mapsto [C_{n-1}], \dots, c \mapsto C_0\} \\
 G_* &= \{a \mapsto A, c \mapsto [S/A^*]\}
 \end{aligned}$$

In fact, the functor-argument structures are all flat structures, with only / operators and always the same label A .

Let $IFL(G)$ denote the language of labelled functor-argument structures of G , we have :

Lemma 16 $IFL(G_n) = flat_{\mathbf{L}^r[A]}(\{c a^k \mid k \leq n\})$ and $IFL(G_*) = flat_{\mathbf{L}^r[A]}(c a^*)$

Proof Only three rules apply to G_n, G_* : **L**. (local dependency rule), **I**. and **Ω** . (ω -dependency rules), all of them enjoying the subformula property. We consider the string languages and show that $L(G_n) = \{c a^k \mid k \leq n\}$ and $L(G_*) = (c a^*)$

- $L(G_0)$: (1) it clearly contains c (assigned to S) and (2) only c since no rule applies to $\{A, S\}$.
- $L(G_n)$ ($n > 0$). We have $C_n = C_{n-1}/A$
 - (1) By induction $L(G_n)$ contains $\{c a^k \mid k < n\}$; it also contains $w = c a^n$, because : $[C_n]A \vdash [C_{n-1}]$ (by **L**. rule)
 - (2) Let $w' \in L(G_n)$. We observe that w cannot start with an a (an A on the left part of a type could not disappear, due to the use of right constructors only); and w' cannot

- contain several c (no S under a constructor) ; thus $w' = cw''$, where $w'' \in \{a\}^*$. Clearly if Γ has more than n occurrences of A , $C_n\Gamma \not\vdash S$, hence $w \in \{c a^k \mid k \leq n\}$
- $L(G_*)$: (1) it clearly contains $c\{a\}^*$ using $[S/A^*]A\Delta \vdash S\Delta$ (**I**. rule) and $[S/A^*] \vdash S$ (**Ω** . rule)
 - (2) $w' \in L(G_*)$ has exactly one c (at least one to provide S , and no more, as explained above for G_n); it cannot start with a (otherwise a type part would remain before S). Therefore, $w' \in c\{a\}^*$.

The language characterization extends to the (labelled) functor-argument structures, because types have only / operator, providing flat structures, and they have the same label A . \square

Theorem 17 *$IFL(G_n)$ and $IFL(G_*)$ define a limit point that establishes the non-learnability from labelled functor-argument structures for the classes of grammars allowing iterated dependency types (A^*) and at least 3 words.*

The similar question for rigid or k -valued CDG with iteration is left open.

3.5 Limit points for labelled and unlabelled dependency structures

We show that the former limit point constructions also give limit points for the dependency structure languages. Let ρ_k and $\rho_{*,k}$ denote the following proof trees (with word positions):

$$\begin{array}{c}
 \rho_k : \\
 \frac{\frac{([S/A \dots /A], 1) \quad (A, 2)}{k \text{ occ.}}}{([S/A \dots /A], 1)} \mathbf{L}^r \quad (A, 3) \\
 \frac{\vdots}{k=1 \text{ occ.}} \mathbf{L}^r \\
 \frac{([S/A], 1) \quad \dots \quad (A, k)}{([S], 1)} \mathbf{L}^r
 \end{array}
 \qquad
 \begin{array}{c}
 \rho_{*,k} : \\
 \frac{([S/A^*], 1) \quad (A, 2)}{([S/A^*], 1)} \mathbf{I}^r \quad (A, 3) \mathbf{I}^r \\
 \vdots \\
 \frac{([S/A^*], 1) \quad \dots \quad (A, k)}{([S/A^*], 1)} \mathbf{I}^r \\
 \frac{([S/A^*], 1)}{([S], 1)} \mathbf{\Omega}^r
 \end{array}$$

In fact, the dependency structures corresponding to ρ_k and $\rho_{*,k}$ are the same, let DS_k denote this dependency structure.

We recall that $\Delta(G)$ denotes the DS -language of G (the generated structures, see definition 5).

Lemma 18 $\Delta(G_n) = \{DS_k \mid k \leq n\}$ and $\Delta(G_*) = \{DS_k \mid k \geq 0\}$

Proof These proof trees are generated by grammars G_n, G_* respectively (this can be shown using the former string language characterization and the discussion concerning these grammars). \square

Note that $\cup_{i \geq 1} \Delta(G_i) = \Delta(G_*)$, and that these $\Delta(G_i)$ form an infinite ascending chain.

Theorem 19 *The limit point $\Delta(G_n), \Delta(G_*)$ establishes the non-learnability from (labelled) dependency structures for the underlying classes of grammars: those allowing iterated dependency types (A^*).*

Remark if we drop the dependency names in dependency structures, and consider the languages generated in this way, then we obtain a limit point in the rigid case, based on the previous limit point construction G'_n, G'_* in Definition 10 : these grammars generate the unlabelled version of $\{DS_k \mid k \leq n\}$ and of $\{DS_k \mid k \geq 0\}$ respectively.

The similar learnability question from labelled dependency structures for rigid or k -rigid CDG with iteration is left open. It would be interesting to know the amount of information needed to have the possibility to design a grammar learning algorithm in this paradigm, and to understand better the frontier between learnability and un-learnability.

3.6 Bounds and string learnability

A List-like Simulation. In order to simulate an iterated type such that :

$$[\beta/a^*]^{P_0} a^{P_1} \dots a^{P_n} \vdash [\beta]^{P_0 P_1 \dots P_n}$$

we can distinguish two types, one type a for a first occurrence in a sequence and one type $a \setminus a$ for following occurrences in a sequence of elements of type a (this encodes in fact one or more iterations of a). As in :

$$\begin{array}{ccccccc} \text{John} & \text{ran} & \text{fast} & \text{yesterday} & \text{nearby} & & \\ n & [n \setminus s/a] & a & [a \setminus a] & [a \setminus a] & & \end{array}$$

We have two assignments for *ran* in “John ran”, $\text{ran} \mapsto n \setminus s$ but in “John ran fast, yesterday”, $\text{ran} \mapsto n \setminus s/a$. Unfortunately, this approach increases the number of types in the lexicon: if a type includes N subtypes of the form A^* , the simulation associates 2^N types. For instance, $x/a^*/b^*$ is transformed into $x, x/a, x/b$ and $x/a/b$. A similar encoding is given for an extension of pregroups in Béchet et al. (2008).

In the case of structures, note that, however, such a simulation induces a particular and rather unnatural dependency structure (in the example above, every adverb is subordinate to the next adverb rather than directly to the verb). It is more pertinent for theoretical issues on string languages.

Bounds. As a corollary, for a class of CDG *without discontinuous dependencies* for which the number of iterated types is bounded by a fixed N , the simulation leads to a class of grammars without iterated types, which is also k -valued: the number of assignments per word is bounded by a large but fixed number ($k = 2^N$). This means that the class of rigid CDG allowing at most N iterated types is learnable from strings. This fact also extends to k -valued CDG, not only to rigid (1-valued) CDG.

4 Incremental learning

Below we show an incremental algorithm *strongly* learning CDG from structures DS (rather than from strings). This means that $\Delta(G)$ serves as the observation set $\Phi(G)$ and that the limit grammar is *strongly* equivalent (in the sense of Definition 20) to the target grammar. From the very beginning, it should be clear that, in contrast with the constituent structure grammars and also with the classical CG, the existence of such learning algorithm is not guaranteed because, due to the iterated types, the straightforward arguments of sub-formulas’ set cardinality do not work. On the other hand, the learning algorithm \mathcal{A} below

is *incremental* in the sense that every next hypothetical CDG $\mathcal{A}(\sigma[i+1])$ “extends” the preceding grammar $\mathcal{A}(\sigma[i])$ and it is so *without any rigidity constraint* (the algorithm and results apply without bound on the number of types). Incremental learning algorithms are rare. Those that we know, are unification-based and they apply only to *rigid* grammars (cf. Buszkowski and Penn 1990; Béchet et al. 2004). They cannot be considered as practical (at least for NLP) because the real application grammars are never rigid. In the cases when k -valued learnability is a consequence of rigid learnability, it is more of theoretical interest because the existence of a learning algorithm is based on Kanazawa’s finite-valued-relation reduction Kanazawa (1998), that may be untractable.

4.1 Grammar order and incrementality

Our notion of incrementality is based on a partial “flexibility” order \leq on CDGs. Basically, the order corresponds to grammar expansion in the sense that $G_1 \leq G_2$ means that G_2 defines no less dependency structures than G_1 and defines dependency structures at least as precise as those of G_1 .

Definition 20 (Strong equivalence) Let G_1 and G_2 be two CDG, $G_1 \equiv_s G_2$ iff $\Delta(G_1) = \Delta(G_2)$.

Definition 21 Δ is said to be *monotonic* with respect to a partial order \leq on CDGs iff for any CDG G_1, G_2 :

if $G_1 \leq G_2$ then $\Delta(G_1) \subseteq \Delta(G_2)$.

When this holds \leq is said to induce *structure-monotonicity*.

Such a partial order \leq is an element of our definition of *incremental learning*.

Definition 22 (Incremental learning algorithm) Let \mathcal{A} be an inference algorithm for CDGs from DS and σ be a training sequence for a CDG G .

1. \mathcal{A} is *monotonic* on σ if $\mathcal{A}(\sigma[i]) \leq \mathcal{A}(\sigma[j])$ for all $i \leq j$.
2. \mathcal{A} is *faithful* on σ if $\Delta(\mathcal{A}(\sigma[i])) \subseteq \Delta(G)$ for all i .
3. \mathcal{A} is *expansive* on σ if $\sigma[i] \subseteq \Delta(\mathcal{A}(\sigma[i]))$ for all i .

\mathcal{A} is said *incremental* w.r.t. \leq when it satisfies properties 1, 2 and 3.¹²

Theorem 4 Let \leq denote a partial order that induces structure-monotonicity. Let σ be a training sequence for a CDG G . If an inference algorithm \mathcal{A} is monotonic, faithful, and expansive on σ , and if \mathcal{A} stabilizes on σ then $\lim_{i \rightarrow \infty} \mathcal{A}(\sigma[i]) \equiv_s G$.

Proof Indeed, stabilization implies that $\lim_{i \rightarrow \infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ for some T . Then $\Delta(\mathcal{A}(\sigma[T])) \subseteq \Delta(G)$ because of faithfulness. At the same time, by expansiveness and monotonicity, $\Delta(G) = \sigma = \bigcup_{i=1}^{\infty} \sigma[i] \subseteq \bigcup_{i=1}^{\infty} \Delta(\mathcal{A}(\sigma[i])) \subseteq \bigcup_{i=1}^T \Delta(\mathcal{A}(\sigma[i])) \subseteq \Delta(\mathcal{A}(\sigma[T]))$. \square

¹² The notions of faithful and of expansive are close to those of prudent and of consistent in Kanazawa (1998).

We now define a particular partial order \leq_{cr} called “flexibility PO” in the case of CDG flexible types, where cr stands for “consecutive repetitions”. This PO is the reflexive-transitive closure of the following relation $<_{cr}$ (basic flexibility relation).

Definition 23 (Basic flexibility relation and flexibility PO)

1. for all $i \geq 0$, $0 \leq j \leq m$, $n \geq 0$: $[l_m \setminus \dots \setminus l_j \setminus \mathbf{c} \dots \setminus \mathbf{c} \setminus l_{j-1} \setminus \dots \setminus l_1 \setminus g/r_1 \dots /r_n]^p$
 $<_{cr} [l_m \setminus \dots \setminus l_j \setminus \mathbf{c}^* \setminus l_{j-1} \setminus \dots \setminus l_1 \setminus g/r_1 \dots /r_n]^p$ and for all $i \geq 0$, $0 \leq k \leq n$, $m \geq 0$:
 $[l_m \setminus \dots \setminus l_1 \setminus g/r_1 \dots /r_{k-1} / \mathbf{c} \dots / \mathbf{c} / r_k / \dots / r_n]^p <_{cr} [l_m \setminus \dots \setminus l_1 \setminus g/r_1 \dots /r_{k-1} / \mathbf{c}^* / r_k / \dots / r_n]^p$
 where c is repeated successively i times in $\mathbf{c} \setminus \dots \setminus \mathbf{c} \setminus$ or in $\mathbf{c} / \dots / \mathbf{c} /$ accordingly (i may be 0).
2. $\tau <_{cr} \tau'$ for sets of types τ, τ' , if either:
 - (i) $\tau' = \tau \cup \{t\}$ for a type $t \notin \tau$ or
 - (ii) $\tau = \tau_0 \cup \{t'\}$ and $\tau' = \tau_0 \cup \{t''\}$
 for a set of types τ_0 and some types t', t'' such that $t' <_{cr} t''$.
3. $\lambda <_{cr} \lambda'$ for two type assignments λ and λ' , if $\lambda(w') <_{cr} \lambda'(w')$ for a word w' and $\lambda(w) = \lambda'(w)$ for all words $w \neq w'$.
4. \leq_{cr} is the PO which is the reflexive-transitive closure of

It is not difficult to prove that the expressive power of CDG grows monotonically with respect to this PO:

Lemma 24 Let G_1 and G_2 be two CDG such that $G_1 \leq_{cr} G_2$. Then $\Delta(G_1) \subseteq \Delta(G_2)$ and $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$.

In other words $<_{cr}$ induces structure-monotonicity, and Theorem 4 will apply.

4.2 K-star revealing grammars

As we explain it in Sect. 3, the unlearnability of rigid or k -valued CDG is due to the use of iterated types. Such types are unavoidable in real grammars (cf. the iterated dependency *circ*, as illustrated in Fig. 1 by three successive circumstantial arguments of one verb). But in particular in the real application grammars, the iterated types have very special properties. Firstly, the discontinuous dependencies are never iterated. Secondly, in natural languages, the optional constructions repeated successively several times (two or more) are exactly those iterated (for example a circumstantial is an optional argument of a verb, but the same verb can occur with successively repeated circumstantial arguments, like verb “fallait” in Figure 1). This is the resource we use to resolve the learnability problem. To formalize these properties we need some notations and definitions. The main definition concerns a restriction on the class of grammars that is learned. This class corresponds to grammars where an argument that is used at least K times in a DS must be an iterated argument. Such grammars are called *K-star revealing grammars*.

Definition 25 (K -star-generalization) Let $K > 1$ be an integer. We define a CDG $\mathcal{C}^K(G)$, the *K-star-generalization* of G , by recursively adding for every word w and every local dependency name d the types

$$[l_1 \setminus \dots \setminus l_a \setminus d^* \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c]^P$$

and

$$[l_1 \setminus \dots \setminus l_a \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c]^P$$

when w has a type assignment $w \mapsto t$, where

$$t = [l_1 \setminus \dots \setminus l_a \setminus t_1 \setminus \dots \setminus t_k \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c]^P,$$

every t_1, \dots, t_k is either d or an iterated dependency type x^* and among t_1, \dots, t_k there are **at least K occurrences of d or at least one occurrence of d^*** . Symmetrically, we also add the corresponding types if t_1, \dots, t_k appear in the right part of t .

Example 7 For instance, with $K = 2$, for the type $[a \setminus b^* \setminus a \setminus S/a^*]$, we add the types $[a \setminus a \setminus S/a^*]$ and $[a \setminus b^* \setminus a \setminus S]$ but also $[a^* \setminus S/a^*]$ and $[S/a^*]$. Recursively, we also add $[a \setminus a \setminus S]$, $[a^* \setminus S]$ and $[S]$. The size of $C^K(G)$ can be exponential with respect to the size of G .

Definition 26 (K -star revealing CDG) Let $K > 1$ be an integer. CDG G is **K -star revealing** if $C^K(G) \equiv_s G$. The class of CDGs that are K -star revealing is noted $\mathcal{CDG}^{K \rightarrow *}$.

Example 8 For instance, if we define the grammar $G(t)$ by $A \mapsto [a]$, $B \mapsto [b]$, $C \mapsto t$, where t is a type, then we can prove that:

- $G([a^* \setminus S/a^*])$, $G([a^* \setminus b^* \setminus a^* \setminus S])$ and $G([a^* \setminus b \setminus a^* \setminus S])$ are all 2-star revealing,
- $G([a^* \setminus a \setminus S])$, $G([a^* \setminus b^* \setminus a \setminus S])$ and $G([a \setminus b^* \setminus a \setminus S])$ are not 2-star revealing.

We see that in a K -star revealing grammar, one and the same iterated type d^* may be used in a type several times. Usually, each occurrence is not in the same block as the local dependency name d . Besides this, there should be less than K occurrences of d in a block if there is no occurrence of d^* and this block is separated from other blocks by types that are not iterated.

4.3 Inference algorithm

A vicinity corresponds to the part of a type that is used in a DS.

Definition 27 (Vicinity) Given a DS D , the incoming and outgoing dependencies of a word w can be either local, anchor or discontinuous. For a discontinuous dependency d on a word w , we define its polarity p (\setminus , \searrow , \swarrow , $/$), according to its direction (left, right) and as negative if it is incoming to w , positive otherwise.

Let D be a DS in which an occurrence of a word w has : the incoming projective dependency or anchor h (or the axiom S), the left projective dependencies or anchors l_k, \dots, l_1 (in this order), the right projective dependencies or anchors r_1, \dots, r_m (in this order), and the discontinuous dependencies $d_1, \dots, d_n \in \mathbf{V}$ with their respective polarities p_1, \dots, p_n .

Then the *vicinity* of w in D is the type

$$V(w, D) = [l_1 \setminus \dots \setminus l_k \setminus h/r_m / \dots / r_1]^P,$$

Fig. 5 Inference algorithm $\mathbf{TGE}^{(K)}$

Algorithm $\mathbf{TGE}^{(K)}$ (type-generalize-expand):
Input: $\sigma[i]$ (σ being a training sequence).
Output: CDG $\mathbf{TGE}^{(K)}(\sigma[i])$.
let $G_H = (W_H, \mathbf{C}_H, S, \lambda_H)$ **where**
 $W_H := \emptyset$; $\mathbf{C}_H := \{S\}$; $\lambda_H := \emptyset$; $k := 0$
 (loop) **for** $i \geq 0$ // Infinite loop on σ
 let $\sigma[i+1] = \sigma[i] \cdot D$;
 let $(x, E) = D$;
 (loop) **for every** $w \in x$
 $W_H := W_H \cup \{w\}$;
 let $t_w = V(w, D)$
 (loop) **while** $t_w = [\alpha \backslash l \backslash \mathbf{d} \backslash \dots \backslash \mathbf{d} \backslash r \backslash \beta]^P$
 with at least K consecutive occurrences of d ,
 $l \neq d$ (or not present) and $r \neq d$ (or not present)
 $t_w := [\alpha \backslash l \backslash d^* \backslash r \backslash \beta]^P$
 (loop) **while** $t_w = [\alpha / l / \mathbf{d} / \dots / \mathbf{d} / r / \beta]^P$
 with at least K consecutive occurrences of d ,
 $l \neq d$ (or not present) and $r \neq d$ (or not present)
 $t_w := [\alpha / l / d^* / r / \beta]^P$
 $\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // Expansion
 end end

in which P is a permutation of $p_1 d_1, \dots, p_n d_n$ in a standard lexicographical order $<_{lex}$, for instance, compatible with the polarity order $\backslash < \backslash < / < /$.

For instance, the verb *fallait* in the DS in Figure 1 has the vicinity

$$[pred \backslash circ \backslash circ \backslash circ \backslash S / a - obj].$$

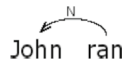
The vicinity corresponding to *donnée* (given) in Figure 3 is

$$[aux] \backslash_{clit-3d-obj} \backslash_{clit-a-obj}$$

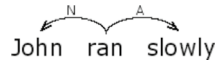
Definition 28 (Algorithm $\mathbf{TGE}^{(K)}$) Figure 5 shows the inference algorithm $\mathbf{TGE}^{(K)}$ which, for every next DS in a training sequence, transforms the observed local, anchor and discontinuous dependencies of every word into a type with repeated local dependencies by introducing iteration for each group of at least K consecutive local dependencies with the same name.

Example 9 We illustrate the learning algorithm, for $K = 2$, with the following 2-star-revealing CDG G_{target} as target grammar:

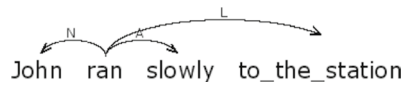
$John \mapsto [N]$ $to_the_station \mapsto [L]$
 $ran \mapsto [N \backslash A^* \backslash S / A^* / L / A^*]$, $[N \backslash A^* \backslash S / A^*]$
 $seemingly, slowly, alone, during_half_an_hour, every_morning \mapsto [A]$
 Algorithm $\mathbf{TGE}^{(2)}$ on $(\sigma[i])$ will add for *ran*:
 $ran \mapsto [N \backslash S]$ for $(i = 1)$:



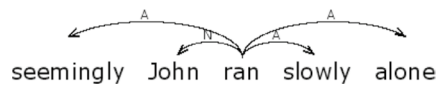
$ran \mapsto [N \setminus S/A]$ for $(i = 2)$:



$ran \mapsto [N \setminus S/L/A]$ for $(i = 3)$:

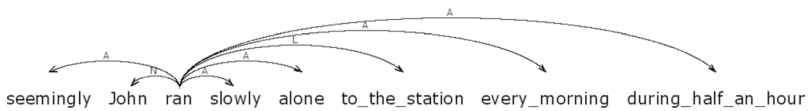


$ran \mapsto [N \setminus A \setminus S/A^*]$ for $(i = 4)$:



etc...

$ran \mapsto [N \setminus A \setminus S/A^*/L/A^*]$ for:



The algorithm also assigns from this training sequence :

$John \mapsto [N]$

$to_the_station \mapsto [L]$

$seemingly, slowly, alone, during_half_an_hour, every_morning \mapsto [A]$

Notice that, given a word, each type assigned at some stage of the algorithm is subsumed by a type in the target grammar:

$[N \setminus S] \leq_{cr} [N \setminus A^* \setminus S/A^*]$ etc.

$[N \setminus S/L/A] \leq_{cr} [N \setminus A^* \setminus S/A^*/L/A^*]$

We will show in the next section that $\mathbf{TGE}^{(K)}$ learns $\mathcal{CDG}^{K \rightarrow *}$.

4.4 Learnability properties

The section shows that the class of K -star revealing CDG is (incrementally) learnable from DS ; to prove Theorem 5, we consider the inference algorithm $\mathbf{TGE}^{(K)}$ (see Figure 5) and introduce further definitions.

Lemma 29 *For G a CDG, there exists a grammar G' such that $G \equiv_s G'$ and the potentials that appear in the type of G verify the standard lexicographical order $<_{lex}$ used to define vicinities. The potentials of $C^K(G')$ are also compatible with the order and $C^K(G) \equiv_s C^K(G')$.*

Proof The types in the lexicon of G with a potential of the form $P_1 \not\prec CP \searrow CP_2$ can never be used in a proof of a DS because it would create an *internal dependency* which are forbidden (see non-internal constraint in Definition 3). Thus, those types can be deleted in G without changing the resulting language. For the other types, the order of the valencies in potentials is not important as long as we do not create internal dependency. For $C^K(G)$ and $C^K(G')$, their potentials are the same as the potentials of G and G' . \square

For the rest of the section, we fix the order for potentials as the one defined by vicinity.

Definition 30

1. *Repetition blocks (R-blocks)* : For $d \in \mathbf{C}$,

$$LB_d = \{t_1 \setminus \cdots \setminus t_i \mid i > 0, t_1, \dots, t_i \in \{d, d^*\}\}$$

$$RB_d = \{t_1 / \cdots / t_i \mid i > 0, t_1, \dots, t_i \in \{d, d^*\}\}$$

Elements of LB_d and of RB_d are called d R-blocks.

2. *Patterns*: Patterns are defined exactly as types, but in the place of \mathbf{C} , we use \mathbf{G} , where \mathbf{G} is the set of *gaps* $\mathbf{G} = \{<d> \mid d \in \mathbf{C}\}$. Moreover, for any α, β, P and d , $[\alpha / <d> \setminus <d> \setminus \beta]^P$ and $[\alpha / <d> / <d> / \beta]^P$ are not patterns. The head, the anchors and the valencies of a type cannot be replaced by gaps. Gaps cannot be iterated.
3. *Superposition and indexed occurrences* of R-blocks :
 - (i) Let π be a pattern, β_1, \dots, β_m be R-blocks and $<d_1>, \dots, <d_m>$ be gaps. Then $\pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$ is the expression resulting from π by the parallel substitution of the R-blocks for the corresponding gaps.
 - (ii) Let E be a type or a vicinity. Then π is *superposable* on E if:

$$E = \pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$$

for some $<d_1>, \dots, <d_m>, \beta_1, \dots, \beta_m$, such that all β_i are d_i R-blocks (β_i are not empty).

A repetition block in LB_d is the part of a type that can correspond to a list of d dependencies on the left and a repetition block in RB_d is the part of a type that can correspond to a list of d dependencies on the right. For instance, for the type

$[a \backslash b^* \backslash d \backslash d^* \backslash h / a]$, the repetition block $d \backslash d$ corresponds to a list of 2 d dependencies on the left and the repetition block $d \backslash d \backslash d^*$ corresponds to a list of at least 2 d dependencies on the left. A pattern is a type with gaps that may be filled by a repetition block of a particular dependency name. The main characteristic of a pattern is that two consecutive gaps cannot correspond to the same dependency name. For instance, $[<a> \backslash \backslash <d> \backslash h / <a>]$ is a pattern but $[<a> \backslash \backslash <d> \backslash <d> \backslash h / <a>]$ is not a pattern. A type is obtained from a pattern by replacing the gaps by a repetition block of the same dependency name. For instance the type $[a \backslash b^* \backslash d \backslash d \backslash d^* \backslash h / a]$ is obtained from $[<a> \backslash \backslash <d> \backslash h / <a>]$ by replacing the left gap $<a>$ by a , the gap $$ by b^* , the gap $<d>$ by $d \backslash d \backslash d^*$ and the right gap $<a>$ by a . This substitution is noted : $[<a> \backslash \backslash <d> \backslash h / <a>](<a> \leftarrow a, \leftarrow b^*, <d> \leftarrow d \backslash d \backslash d^*, <a> \leftarrow a)$. Gaps are introduced only for types that can be iterated. Thus, there is no gap for the heads, the anchors and the valencies of types.

Lemma 31 *For every vicinity V there is a single pattern π superposable on V and a single decomposition (called R -decomposition)*

$$V = \pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$$

Proof The proposition comes from the fact that a vicinity contains no iterated type, a pattern cannot have 2 consecutive gaps for the same dependency and a repetition block is not empty. \square

The verb *fallait* in Figure 1 has the vicinity $[pred \backslash circ \backslash circ \backslash circ \backslash S / @fs / a - obj]$. The only pattern superposable on this vicinity is:

$$\pi = [<pred> \backslash <circ> \backslash S / <@fs> / <a - obj>]$$

and the corresponding type is obtained through the following substitution:

$$\pi(<pred> \leftarrow pred, <circ> \leftarrow circ \backslash circ \backslash circ, <@fs> \leftarrow @fs, <a - obj> \leftarrow a - obj)$$

The vicinity of the participle *ramenée* is:

$$[aux - a / l - obj]^{\text{clit-a-obj}} = [aux - a / <l - obj>]^{\text{clit-a-obj}}(<l - obj> \leftarrow l - obj)$$

Lemma 32 *For $D \in \Delta(G)$ and an occurrence w of a word in D , let $V(w, D) = \pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$ be the R -decomposition of the vicinity of w . There exists a type t that is assigned to w in the lexicon of $\mathcal{C}^K(G)$ and can be used in a proof of D for w such that π is superposable on t .*

Proof For $D \in \Delta(G)$ and w a word of D . There exists a type that is associated to w in the lexicon of G and is used in a proof of D . Thus, there exists at least one type associated to w in the lexicon of $\mathcal{C}^K(G)$ that can be used for w in a proof of Δ . Let t be the minimum length type associated to w in the lexicon of $\mathcal{C}^K(G)$ that can be used for w in a proof of Δ . The R -decomposition of the vicinity of w in D is $V(w, D) = \pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$. The vicinity of w in D must match t because we suppose that the valencies of the potential are ordered (Proposition 29). If π is not superposable on t , it means that some part of t does not correspond to $V(w, D)$: It must be an iterative type x^* , with $x \in \mathbb{C}$ that corresponds to no dependency in the match. Because the types assigned to w in the lexicon of $\mathcal{C}^K(G)$ are

closed when an iterated type is removed, we could find a smaller type for w in $\mathcal{C}^K(G)$ that can be used in a proof of D that is not possible. Thus π is superposable on t . \square

Lemma 1 *The inference algorithm $\mathbf{TGE}^{(K)}$ is monotonic, faithful and expansive on every training sequence σ of a K -star revealing CDG.*

Proof By definition, the algorithm $\mathbf{TGE}^{(K)}$ is *monotonic* (the lexicon is always extended). It is *expansive* because for $\sigma[i]$, we add types to the grammar that are based on the vicinities of the words of $\sigma[i]$. Thus, $\sigma[i] \subseteq \Delta(\mathbf{TGE}^{(K)}(\sigma[i]))$.

To prove that $\mathbf{TGE}^{(K)}$ is *faithful* for $\sigma[i]$ of $\Delta(G) = \Delta(\mathcal{C}^K(G))$, we prove that $\mathbf{TGE}^{(K)}(\sigma[i]) \leq_{cr} \mathcal{C}^K(G)$. In fact, we prove that for any type t in the lexicon of $\mathbf{TGE}^{(K)}(\sigma[i])$, there exists a type t_G in the lexicon of $\mathcal{C}^K(G)$ such that $t = t_G$ or $t = t_1 <_{cr} \dots <_{cr} t_n = t_G$ with $n > 0$, and t_1, \dots, t_n types.

Let t be a type of the word w in the lexicon of $\mathbf{TGE}^{(K)}(\sigma[i])$. The algorithm \mathbf{TGE} produces t for the analysis D of a DS. The analysis D is a positive example thus $D \in \Delta(G) = \Delta(\mathcal{C}^K(G))$. By Proposition 32, if π is the pattern superimposable on the vicinity $V(w, D)$, there exists a minimum length type t' in the lexicon of $\mathcal{C}^K(G)$ assigned to w which can be used in a proof of D . The two superimposings of π for t and t' are : $t = \pi(<d_1> \leftarrow \alpha_1, \dots, <d_m> \leftarrow \alpha_m)$ and $t' = \pi(<d_1> \leftarrow \beta_1, \dots, <d_m> \leftarrow \beta_m)$. For $1 \leq i \leq m$, α_i contains either a list of at most $K - 1$ d_i or d_i^* and β_i can be any d_i R-block (a list of d_i and d_i^*).

The type t' is not more general than or not equal to t if $\exists i, 1 \leq i \leq m$, such that $\alpha_i = d_i^*$ and $\beta_i = d_i \dots d_i$ (d_i l times and no d_i^*). It means that the vicinity has exactly l dependencies labelled by d_i for the position i of the pattern and l must be greater than or equal to K ($\alpha_i = d_i^*$). The type $t'' = \pi(<d_1> \leftarrow \beta_1, \dots, <d_i> \leftarrow \alpha_i, \dots, <d_m> \leftarrow \beta_m)$ must be also assigned to w in $\mathcal{C}^K(G)$, associated is more general than t' (it can be used in a proof of D) but is strictly smaller than t' which is not possible (t' has the minimum type length). Thus t' more general or equal to t . \square

Lemma 2 *The inference algorithm $\mathbf{TGE}^{(K)}$ stabilizes on every training sequence σ of a K -star revealing CDG.*

Proof Because $\mathcal{C}^K(G)$ has a finite number of types, the number of corresponding patterns is also finite. Thus the number of patterns that correspond to the DS in $\Delta(\mathcal{C}^K(G))$ (and of course in σ) is also finite. Because the R-blocks are generalized using $*$ by $\mathbf{TGE}^{(K)}$ when their length is greater or equal to K , the number of R-blocks used by $\mathbf{TGE}^{(K)}$ is finite. Thus the number of generated types is finite and the algorithm certainly stabilizes. \square

Theorem 5 *The class $CDG^{K \rightarrow *}$ of K -star revealing CDG is (incrementally) learnable from DS.*

This theorem results from the two preceding lemmas, Lemma 1 and Lemma 2.

5 Simple K-star grammars

A K -star revealing grammar G verifies a complex property that is difficult or even impossible to check: $C^K(G) \equiv_s G$. We need to define classes of CDG using a more constructive property. This section reconsiders a recent syntactic criterion on categorial grammar types that is easy to check, leading to the definition of simple K -star grammars.¹³

5.1 Simple K-star types and grammar classes

The original definition of “simple K -star” Béchet and Foret (2016) was restricted to CDG with empty potentials and related to consecutive repetitions. We consider it below in the full general case with potentials. Note that considering string languages, the class of CDG with empty potentials generates the context-free languages, but with potentials it goes beyond this class. We also introduce a variant called “global simple K -star” corresponding to a different reading of the repetition principle, when repetitions need not be consecutive in a type.

Definition 33 (Simple K -star) Let $K > 1$ be an integer. Let t denote a categorial type and d denote a dependency name. The type t is said to be *simple left K -star on d* if for any successive occurrences $l_1 \setminus l_2 \setminus \dots \setminus l_p \setminus$ on the left where each l_i is either d or some x^* , there are: (1) at most $K - 1$ occurrences of d and (2) no occurrence of d if there exists at least one occurrence of d^* . The type t is said to be *simple left K -star* if it is simple left K -star on d , for all d . These two notions are defined similarly on the right.

The type t is said to be *simple K -star* if it is *simple left K -star* and *simple right K -star*.

The CDG G is said to be *simple K -star* whenever all types in its lexicon are simple K -star.

The class of CDG that are simple K -star is noted $CDG^{K\sim*}$.

Example 10 For a type t , we define the grammar $G(t)$ by the lexicon $\{a \mapsto [A], b \mapsto [B], c \mapsto t\}$. Then for $t_1 = [A^* \setminus S / A^*]$, $t_2 = [A^* \setminus B^* \setminus A^* \setminus S]$, $t_3 = [A^* \setminus B \setminus A^* \setminus S]$: $G(t_1)$, $G(t_2)$, $G(t_3)$ are simple 2-star and for $t_4 = [A^* \setminus A \setminus S]$, $t_5 = [A^* \setminus B^* \setminus A \setminus S]$, $t_6 = [A \setminus B^* \setminus A \setminus S]$: $G(t_4)$, $G(t_5)$, $G(t_6)$ are not simple 2-star. In fact, for $G(t_4)$, the type assigned to c contains A^* and A in $A^* \setminus A \setminus$ on the left, for $G(t_5)$, A^* and A are separated by B^* and for $G(t_6)$, there are 2 occurrences of A (separated by B^*).

Definition 34 (Global Simple K -star) Let $K > 1$ be an integer. Let t denote a categorial type and d denote a dependency name. The type t is said to be *global simple left K -star on d* if for any successive occurrences $l_1 \setminus l_2 \setminus \dots \setminus l_p \setminus$ on the left there are: (1) at most $K - 1$ occurrences of d and (2) no occurrence of d if there exists at least one occurrence of d^* . The type t is said to be *global simple left K -star* if it is global simple left K -star on d , for all d . These two notions are defined similarly on the right.

The type t is said to be *global simple K -star* if it is *global simple left K -star* and *global simple right K -star*.

¹³ By “simple” we mean here “un-nested”

The CDG G is said to be *global simple K -star* whenever all types in its lexicon are global simple K -star.

The class of CDG that are global simple K -star is noted $\mathcal{CDG}^{K \sim_g^*}$.

Notice that given K , the set of global simple K -star is still infinite (the set of grammars $\mathcal{CDG}^{K \sim_g^*}$ is infinite): consider for example $S \setminus A^* \setminus B^* \setminus A^* \setminus B^* \dots$

Obviously “global simple K -star” entails “simple K -star”, at each stage of the two above definitions, and a global simple K -star grammar is simple K -star.

Example 11 In Example 10, grammars $G(t_1), G(t_2), G(t_3)$ are global simple 2-star and $G(t_4), G(t_5), G(t_6)$ are not global simple 2-star.

Limit point The grammars in Sect. 3.3 are simple K -star ($\forall K > 1$). The class of rigid simple K -star CDG $\forall K > 1$ is thus unlearnable from strings. (also for k -valued classes).

These grammars are moreover global simple K -star. A similar unlearnability result thus holds for the class of rigid global simple 2-star CDG (also for any $K > 1$ or non rigid class).

5.2 Simple K -star grammars and K -star revealing grammars

A K -star revealing grammar G is a CDG such that its K -star generalisation $\mathcal{C}^K(G)$ is equivalent to itself: $\mathcal{C}^K(G) \equiv_s G$. This definition is not constructive because one must prove that two grammars generate the same set of dependency structures. For instance, the following CDG G_1 corresponds to the string language xa^* : $x \mapsto [S/A^*]$; $a \mapsto [A]$. The K -star generalisation of this grammar $\mathcal{C}^K(G_1)$ is: $x \mapsto [S/A^*], [S]$; $a \mapsto [A]$. G_1 and $\mathcal{C}^K(G_1)$ are equivalent because a dependency structure of G_1 can be obtained from a dependency structure of $\mathcal{C}^K(G_1)$ by replacing the type $[S]$ assigned to x by $[S/A^*]$. For complex grammars the problem is more difficult and may be even not decidable for the full class of CDG (CDG with potentials are beyond context-free grammars); In one direction, equivalence problems are shown decidable for large grammar classes in Sénizergues (2002).

Conversely, the notion of being a simple K -star grammar can be easily checked: each type in the lexicon must be checked independently to the other types. Thus it is simpler to use the class of simple K -star grammars rather than the class of K -star revealing grammars.

Theorem 6 *A simple K -star grammar is a K -star revealing grammar.*

Proof Let G be a simple K -star grammar. We have to prove that $\mathcal{C}^K(G) \equiv_s G$ or equivalently that the sets of DS are equal: $\Delta(\mathcal{C}^K(G)) = \Delta(G)$. Because $\mathcal{C}^K(G)$ has the same lexicon as G except that some types are added to some words, $\Delta(G) \subseteq \Delta(\mathcal{C}^K(G))$. For the reverse inclusion, we have to look at the types that are added to the lexicon of G in the K -star generalization $\mathcal{C}^K(G)$. Potentially, for a word w and a dependency name d , the following types have to be added:

$$[l_1 \setminus \dots \setminus l_a \setminus d^* \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c]$$

$$\text{and } [l_1 \setminus \dots \setminus l_a \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c]$$

when w has an assignment $w \mapsto t$ where

$$t = [l_1 \setminus \dots \setminus l_a \setminus t_1 \setminus \dots \setminus t_p \setminus m_1 \setminus \dots \setminus m_b \setminus h/r_1 / \dots / r_c],$$

every t_1, \dots, t_p is either d or an iterated dependency type x^* and among t_1, \dots, t_p there are at least K occurrences of d or at least one occurrence of d^* (and symmetrically).

Because G is a simple K -star grammar, the p successive occurrences t_1, \dots, t_p of t contain at most $K - 1$ occurrences of d and contain no occurrence of d or no occurrence of d^* . It means that t_1, \dots, t_p contain at least one occurrence of d^* and no occurrence of d : Each t_i is an iterated dependency type x^* and from them at least one is d^* . As a consequence, a vicinity of a DS that matches one of the added types also matches t and the DS is also generated by G . G and the grammar obtained by adding the two types are equivalent.

Moreover, the grammar with the two new types is also a simple K -star grammar. The new types verify the condition of the types of a simple K -star grammar. Let $t'_1 \setminus \dots \setminus t'_q$ be q successive occurrences on the left of one of the new types. If the added d^* type or l_a and m_1 aren't in $t'_1 \setminus \dots \setminus t'_q$, the q occurrences verify the condition for simple K -star grammars. Otherwise, the condition on t for simple K -star grammars holds for a segment of successive occurrences where $t_1 \setminus \dots \setminus t_p$ is inserted in $t'_1 \setminus \dots \setminus t'_q$ or replaces d^* in $t'_1 \setminus \dots \setminus t'_q$. As a consequence, $t'_1 \setminus \dots \setminus t'_q$ must also verify the condition for simple K -star grammars.

Thus, the added types don't change the DS-language and define a simple K -star grammar. Recursively, the completion algorithm, that starts with a simple K -star grammar G , ends with a simple K -star grammar $C^K(G)$ that is equivalent to G : G is K -star revealing. \square

Corollary 2 *The class $CDG^{K\sim*}$ of simple K -star CDG is (incrementally) learnable from DS. The class is learnt by the inference algorithm $\mathbf{TGE}^{(K)}$.*

In fact, the class of simple K -star grammars and the class of K -star revealing grammars are not identical. Some K -star revealing grammars are not simple K -star grammar. A very simple reason for this fact comes from the syntactical definition of the simple K -star grammars versus the language equivalence definition of the K -star revealing grammars. It is easy to define a grammar where some part of the lexicon is not used. This part does not create a problem for the definition of a K -star revealing grammar but is a problem for the definition of a simple K -star grammar. For instance, the following grammar is a 2-star revealing grammar (a can never be used) but is not a simple 2-star grammar (2 successive A on the left of $[A \setminus A \setminus S]$): $x \mapsto [S]$; $a \mapsto [A \setminus A \setminus S]$.

A more interesting example is given by $x \mapsto [S]$, $[A \setminus A^* \setminus S]$; $a \mapsto [A]$: It is a 2-star revealing grammar that has only useful types but is not simple 2-star. It is not simple 2-star because the type $[A \setminus A^* \setminus S]$ contains the two successive types A and A^* on the left. The completion mechanism gives the following grammar: $x \mapsto [S]$, $[A \setminus A^* \setminus S]$, $[A \setminus S]$, $[A^* \setminus S]$; $a \mapsto [A]$; this grammar is equivalent to the initial one and thus it is 2-star revealing.

Moreover, there exist DS-languages that are generated by K -star revealing grammars but are not generated by any simple K -star grammar.

Theorem 7 *Let G_2 be the 2-star revealing grammar:*

$$x \mapsto [A \setminus B^* \setminus A \setminus S], [A^* \setminus S] \quad a \mapsto [A] \quad b \mapsto [B]$$

There is no simple 2-star grammar that generates $\Delta(G_2)$.

Useless types For a CDG, some parts of the lexicon may be useless. It can be all the types associated to a word (a word that doesn't appear in the language generated by the grammar), one or several types of a word (the word appears in the language but the derivations cannot use these types). It can also be some iterated type of useful types when it is impossible to define a derivation ending in this type. For instance, for the grammar $x \mapsto [Z^* \backslash S]$, there is only one DS, $[Z^* \backslash S]$ is useful but the left iterated type Z^* is useless.

We suppose below that we have a simple 2-star grammar that generates $\Delta(G_2)$ and that has no useless part (in the previous example, a simplified grammar would be $x \mapsto [S]$).

Proof The DS-language $\Delta(G_2)$ is the set of dependency structures that have one main head x and a set of dependent on the left that can be either one a , none, one or several b and one a or that can be none, one or several a . For this grammar, the types associated to a and to b are respectively $[A]$ and $[B]$ (a DS contains the local dependency names A and B for dependencies ending in a and b). The types associated to x are of the form $[t_1 \backslash \dots \backslash t_p \backslash S]$ where each t_i is A , B , A^* or B^* . Because the number of b is not bound in the DS-language, there exists at least one type associated to x that contains at least one B^* . The type cannot contain A^* and it must have exactly two local dependency names A that must be the left and the right ends of the left part of the type ($t_1 = A$ and $t_p = A$). The part between t_1 and t_p can only be occurrences of B or B^* . Because the grammar is simple 2-star and because one of them is B^* , the other cannot be B . Thus the type is $[A \backslash B^* \backslash \dots \backslash B^* \backslash A \backslash S]$. But it is not possible because $A \backslash B^* \backslash \dots \backslash B^* \backslash A$ contains 2 occurrences A separated by iterated types and this sequence is forbidden in a simple 2-star grammar. \square

The class of simple K -star grammars defines a smaller set of DS-languages than the class of K -star revealing grammars. This is generally not a problem because from a K -star revealing grammar it is always possible to define a simple K -star grammar that is a generalization of the former grammar: some local dependency names are transformed into iterated types. For instance, G_2 can be transformed into the following grammar which is a simple 2-star grammar: $x \mapsto [A^* \backslash B^* \backslash A^* \backslash S]$, $[A^* \backslash S]$; $a \mapsto [A]$; $b \mapsto [B]$. This example is in fact global simple 2-star. We may consider for any grammar its global simple K -star generalization as follows:

Definition 35 (Global Simple K -star Generalization)

For any type t , its global simple K -star generalization, written $gs^{(K)}(t)$ is obtained by applying these rules:

- for each d on the left, where $d \backslash$ occurs at least K times or if $d^* \backslash$ is present, then replace each $d \backslash$ with its starred version $d^* \backslash$;
- for each d on the right, proceed similarly.

The definition is extended to grammars, by replacing each assigned type t by $gs^{(K)}(t)$.

The grammar G is said to be global simple K -star when $G = gs^{(K)}(G)$.

We get the following structure language inclusions, for any CDG G :

$$\Delta(G) \subseteq \Delta(gs^{(K+1)}(G)) \subseteq \Delta(gs^{(K)}(G)), \text{ for each } K > 1.$$

If G is global simple K -star, then $gs^{(K)}(G) = G$ and the above inclusions are equalities.

Using directly algorithm $TGE^{(K)}$ on a global simple K -star grammar will provide a simple K -star grammar but not necessarily a global simple K -star grammar (for example, if

a vicinity such as $S/A/B/A$ is used in a DS and if $K = 2$). A possible adaptation of this approach to the global variant is to use algorithm $TGE^{(K)}$ and then apply $gs^{(K)}$ to the result; an alternative is to adapt the learning algorithm using a different generalization step, applying $gs^{(K)}$ at each stage.

These adaptations of the algorithm $TGE^{(K)}$ preserve the global simple K -star property as sketched below.

Lemma 3 (Global Simple Generalization Properties)

1. If G is global simple K -star, then $gs^{(K)}(C^K(G)) \equiv_s G$
2. For any types t_1, t_2 : if $t_1 \leq_{cr} t_2$ then $gs^{(K)}(t_1) \leq_{cr} gs^{(K)}(t_2)$
3. For any grammars G_1, G_2 : if $G_1 \leq_{cr} G_2$ then $gs^{(K)}(G_1) \leq_{cr} gs^{(K)}(G_2)$

Proof (1) comes from the fact that a global simple K -star grammar is also a simple K -star grammar and thus a K -star revealing grammar. (2) and (3) are straightforward. \square

Theorem 8 If G is global simple K -star, then for any step i

$$\Delta(gs^{(K)}(TGE^{(K)}(\sigma[i]))) \subseteq \Delta(G)$$

and the application of $gs^{(K)}$ on the output of $TGE^{(K)}$ at stabilization is equivalent to G (with the same set of DS).

Proof As a corollary of the above lemma, from $TGE^{(K)}(\sigma[i]) \leq_{cr} C^K(G)$, we get $gs^{(K)}(TGE^{(K)}(\sigma[i])) \leq_{cr} gs^{(K)}(C^K(G)) \equiv_s G$ which implies:

$$\Delta(gs^{(K)}(TGE^{(K)}(\sigma[i]))) \subseteq \Delta(gs^{(K)}(C^K(G))) = \Delta(G)$$

We then have $\Delta(TGE^{(K)}(\sigma[i])) \subseteq \Delta(gs^{(K)}(TGE^{(K)}(\sigma[i]))) \subseteq \Delta(G)$. We thus get a global simple K -star grammar equivalent to G if we apply $gs^{(K)}$ on the result of the $TGE^{(K)}$ algorithm at stabilization. \square

6 Conclusion

In this paper, we propose a new model of *incremental* learning of categorial dependency grammars with unlimited iterated types from input dependency structures without marked iteration. The model reflects the real situation of deterministic inference of a dependency grammar from a dependency treebank. The definition of K -star revealing grammars is a sufficient condition to insure learnability from dependency structures. It is widely accepted in traditional linguistics for small K , which makes this model interesting for practical purposes. As our study shows, the more traditional unification-based learning from function-argument structures fails even for rigid categorial dependency grammars, in the presence of iterated types with unlimited iteration.

The K -star revealing condition was defined in “semantic” terms. The question was raised whether one can find a syntactic formulation. In this paper, we replace this non-constructive criterion on CDG grammars by a syntactic constructive one called simple K -star

that is slightly more restrictive. We show that the new class is learnable from dependency structures.

We also consider several interpretations of repeatable dependency (local with simple K -star condition or global to a type with global simple K -star condition). Another source of variation for the global simple K -star grammars lies in the learning algorithm and the way types are generalized: instead of the local replacements of $TGE^{(K)}$, these could be dealt globally (on each side), which would lead to somewhat different definitions, orders and algorithm. For example, instead of $[N \setminus A^* \setminus S/A/L/A^*]$ (which A and A^* at the same time on the right side) we would get $[N \setminus A^* \setminus S/A^*/L/A^*]$ directly (A and A^* cannot be used at the same time on each side). This variant can be seen as an intermediate between the strict reading of repeatable optional dependencies as consecutive repetitions of $TGE^{(K)}$ and more flexible interpretations like “dispersed iterations” Béchet et al. (2011).

Other variants are still possible: such as a more global principle on types on the whole grammar (having “super-global” dependencies) where a dependency A is either repeatable every where (A^*) or never repeatable (A) in the whole grammar.

As further remarks on the inference algorithm: the algorithm presented in Sect. 4 may be improved by deleting types that are subsumed by another one. The resulting grammar would then be strongly equivalent but in a compressed form. We can also think of marking some dependency names as not following the repetition principle, the definition and the algorithm should then be adapted with respect to a given subset.

This work has been developed in the computational linguistic domain. It would be interesting to reconsider these notions in a purely theoretical way (other languages and automaton) or other application domains (such as bioinformatics).

This article contributes on symbolic learning possibilities and impossibilities for CDG. This could be extended to dependency grammars in general or other lexicalized grammar formalisms. As concern the CDG formalism itself, important questions for CDG and their languages were solved in Dekhtyar et al. (2015) but some theoretical questions listed in their conclusion remain open for CDG. One of them is the positioning of CDG in the variety of grammatical frameworks for NLP, going beyond context-free grammars.

On the practical side, works aiming at developing large-scale CDG grammars could integrate Algorithm $TGE^{(K)}$ presented in this paper. Such developments would also help to validate the hypotheses introduced (K -star revealing, simple K -star, global simple K -star) for defining restricted classes of CDG.

References

- Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control*, 45, 117–135.
- Béchet, D. (2003). k -valued link grammars are learnable from strings. In G. Penn (Ed.), *Proceedings of the 8th conference on formal grammar (FG-2003 or FGVienna)*, Vienna, Austria, August 16–17, 2003 (pp. 3–12). CSLI Publications. <http://cslipublications.stanford.edu/FG/2003>.
- Béchet, D., Dikovsky, A., & Foret, A. (2005). Dependency structure grammar. In P. Blache, E. Stabler, J. Busquets, R. Moot (Eds.), *Logical aspects of computational linguistics, 5th international conference, LACL 2005, Bordeaux, France, April 28–30, 2005. Proceedings, Lecture notes in artificial intelligence (LNAI)* (Vol. 3492, pp. 18–34). Springer. <https://doi.org/10.1007/b136076>.
- Béchet, D., Dikovsky, A., & Foret, A. (2010). Two models of learning iterated dependencies. In P. de Groote, M. Nederhof (Eds.), *Formal grammar, 15th and 16th international conferences, FG 2010, Copenhagen, Denmark, August 2010, FG 2011, Ljubljana, Slovenia, August 2011, Revised*

- selected papers, Lecture notes in computer science (LNCS)* (Vol. 7395, pp. 17–32). Springer. https://doi.org/10.1007/978-3-642-32024-8_2.
- Béchet, D., Dikovsky, A., & Foret, A. (2011). On dispersed and choice iteration in incrementally learnable dependency types. In: S. Pogodalla, J. Prost (Eds.), *Logical aspects of computational linguistics, 6th international conference, LACL 2011, Montpellier, France, June 29–July 1, 2011. Proceedings, Lecture notes in computer science (LNCS)* (Vol. 6736, pp. 80–95). Springer. https://doi.org/10.1007/978-3-642-22221-4_6.
- Béchet, D., Dikovsky, A., Foret, A., & Garel, E. (2008). Optional and iterated types for pregroup grammars. In: C. Martín-Vide, F. Otto, H. Fernau (Eds.), *Language and automata theory and applications, 2nd international conference, LATA 2008, Tarragona, Spain, March 13–19, 2008, Revised Papers, Lecture notes in computer science (LNCS)* (Vol. 5196, pp. 88–100). Springer. <https://doi.org/10.1007/978-3-540-88282-4>.
- Béchet, D., Dikovsky, A., Foret, A., Moreau, E. (2004). On learning discontinuous dependencies from positive data. In P. Monachesi (Ed.), *Proceedings of the 9th international conference on formal grammar*.
- Béchet, D., Dikovsky, A., & Lacroix, O. (2014). “CDG Lab”: An integrated environment for categorial dependency grammar and dependency treebank development. In K. Gerdes, E. Hajičová, L. Wanner (Eds.), *Computational dependency theory, Frontiers in artificial intelligence and applications* (Vol. 258, pp. 153–169). IOS Press. <https://doi.org/10.3233/978-1-61499-352-0-153>.
- Béchet, D., & Foret, A. (2003). k -valued non-associative Lambek categorial grammars are not learnable from strings. In *Proceedings of the 41st annual meeting of the association for computational linguistics (ACL 2003), Sapporo, Japan, July 2003* (pp. 351–358). ACL.
- Béchet, D., & Foret, A. (2003). 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* (Vol. 84).
- Béchet, D., & Foret, A. (2016). Simple k -star categorial dependency grammars and their inference. In S. Verwer, M. van Zaanen, R. Smetsers (Eds.), *Proceedings of the 13th international conference on grammatical inference, ICGI 2016, Delft, The Netherlands, October 5–7, 2016, JMLR workshop and conference proceedings* (Vol. 57, pp. 3–14). JMLR.org. <http://proceedings.mlr.press/v57/beche16.html>.
- Bonato, R. (2006). A Study on Learnability for Rigid Lambek Grammars. Research Report RR-5964, INRIA. <https://hal.inria.fr/inria-00088818>.
- Bonato, R., & Retoré, C. (2014). Learning lambek grammars from proof frames. In C. Casadio, B. Coecke, M. Moortgat, P. Scott (Eds.), *Categories and types in logic, language, and physics—essays dedicated to Jim Lambek on the Occasion of His 90th Birthday, Lecture notes in computer science* (Vol. 8222, pp. 108–135). Springer. https://doi.org/10.1007/978-3-642-54789-8_7.
- Buszkowski, W. (1987). Discovery procedures for categorial grammars. In E. Klein & J. van Benthem (Eds.), *Categories, polymorphism and unification*. Amsterdam: University of Amsterdam.
- Buszkowski, W., & Penn, G. (1990). Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49, 431–454.
- Costa Florêncio, C. (2003). Rigid grammars in the associative-commutative Lambek calculus are not learnable. In A. Copestake, J. Hajič (Eds.), *Proceedings of EACL2003, 10th conference of the European chapter of the association for computational linguistics, Agro Hotel, Budapest, April 12–17, 2003* (pp. 75–82). ACL.
- Costa Florêncio, C. (2012). Learning tree adjoining grammars from structures and strings. In *Proceedings of ICGI 2012, Washington D.C., USA, September 5–8, 2012, JMLR: Workshop and conference proceedings* (Vol. 21, pp. 129–132).
- Costa Florêncio, C., & Fernau, H. (2010). Hölder norms and a hierarchy theorem for parameterized classes of CCG. In J.M. Sempere, P. García (Eds.), *Proceedings of the international colloquium on grammatical inference, ICGI'10, Lecture notes in computer science (LNCS)* (Vol. 6339, pp. 280–283). Springer. <https://doi.org/10.1007/978-3-642-15488-1>.
- Costa Florêncio, C., & Fernau, H. (2012). On families of categorial grammars of bounded value, their learnability and related complexity questions. *Theoretical Computer Science*, 452, 21–38. <https://doi.org/10.1016/j.tcs.2012.05.016>.
- de la Higuera, C. (2010). *Grammatical inference: Learning automata and grammars*. New York: Cambridge University Press.
- Dekhtyar, M., & Dikovsky, A. (2004). Categorial dependency grammars. In M. Moortgat, V. Prince (Eds.), *Proceedings of the international conference on categorial grammars (CG2004), Montpellier, France, June 2004* (pp. 76–91).

- Dekhtyar, M., & Dikovskiy, A. (2008). Generalized categorial dependency grammars. In Trakhtenbrot/ Festschrift, *Lecture notes in artificial intelligence (LNCS)* (Vol. 4800, pp. 230–255). Springer.
- Dekhtyar, M. I., Dikovskiy, A., & Karlov, B. (2015). Categorial dependency grammars. *Theoretical Computer Science*, 579, 33–63. <https://doi.org/10.1016/j.tcs.2015.01.043>.
- Dikovskij, A., & Modina, L. (2000). Dependencies on the other side of the Curtain. *Traitement Automatique des Langues (TAL)*, 41(1), 79–111.
- Dikovskiy, A. (2004). Dependencies as categories. In *Recent advances in dependency grammars. COLING'04 workshop* (pp. 90–97).
- Dikovskiy, A. (2011). Categorial dependency grammars: From theory to large scale grammars. In K. Gerdess, E. Hajicova, L. Wanner (Eds.), *Conference on dependency linguistics 2011* (pp. 262–271). Barcelona, Spain.
- Duchier, D., & Debusmann, R. (2001). Topological dependency trees: A constraint-based account of linear precedence. In *Association for computational linguistic, 39th annual meeting and 10th conference of the European Chapter, Proceedings of the Conference*, July 9–11, 2001, Toulouse, France (pp. 180–187). ACL & Morgan Kaufmann Publishers.
- Eisner, J. M. (1991). Bilexical grammars and their cubic-time parsing algorithms. In H. Bunt, A. Nijholt (Eds.), *Advances in probabilistic and other parsing technologies* (pp. 29–62). Kluwer.
- Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control*, 8(3), 304–337. [https://doi.org/10.1016/S0019-9958\(65\)90232-9](https://doi.org/10.1016/S0019-9958(65)90232-9).
- Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10, 447–474.
- Hays, D. (1960). Grouping and dependency theories. Research memorandum RM-2646, The RAND Corporation. In *Proc. of the National Symp. on Machine Translation, Englewood Cliffs (N.Y.), 1961* (pp. 258–266).
- Joshi, A. K., Shanker, V. K., & Weir, D. J. (1991). The convergence of mildly context-sensitive grammar formalisms. In *Foundational issues in natural language processing* (pp. 31–81). Cambridge, MA.
- Kanazawa, M. (1998). Learnable classes of categorial grammars. *Studies in logic, language and information*. FoLLI & CSLI.
- Kruijff, G.J. (2001). Dependency grammar logic and information structure. Ph.D. thesis, Charles University, Prague.
- Lacroix, O., & Béchet, D. (2014). A three-step transition-based system for non-projective dependency parsing. In J. Hajic, J. Tsujii (Eds.), *COLING 2014, 25th international conference on computational linguistics, Proceedings of the conference: Technical papers*, August 23–29, 2014, Dublin, Ireland (pp. 224–232). ACL.
- Maruyama, H. (1990). Structural disambiguation with constraint propagation. In: R.C. Berwick (Ed.), *28th annual meeting of the association for computational linguistics, June 6–9, 1990, University of Pittsburgh, Pittsburgh, Pennsylvania, USA, Proceedings* (pp. 31–38). ACL.
- Meľčuk, I. (1988). *Dependency syntax*. Albany, NY: SUNY Press.
- Moot, R., & Retoré, C. (2012). *The logic of categorial grammars—A deductive account of natural language syntax and semantics, Lecture notes in computer science* (Vol. 6850). Springer. <https://doi.org/10.1007/978-3-642-31555-8>.
- Motoki, T., Shinohara, T., & Wright, K. (1991). The correct definition of finite elasticity: Corrigendum to identification of unions. In *The 4th annual workshop on computational learning theory* (p. 375). San Mateo, CA.
- Nederhof, M. J. (2016). A short proof that o_2 is an MCFL. In *Proceedings of the 54th annual meeting of the association for computational linguistics (Volume 1: Long Papers)* (pp. 1117–1126). Association for Computational Linguistics, Berlin, Germany. <https://doi.org/10.18653/v1/P16-1106>. <https://www.aclweb.org/anthology/P16-1106>.
- Salvati, S. (2015). MIX is a 2-MCFL and the word problem in \mathbb{Z}^2 is captured by the IO and the OI hierarchies. *Journal of Computer and System Sciences*, 81(7), 1252–1277. <https://doi.org/10.1016/j.jcss.2015.03.004>.
- Sénizergues, G. (2002). $L(A) = L(B)$? Decidability results from complete formal systems. In: P. Widmayer, F.T. Ruiz, R.M. Bueno, M. Hennessy, S.J. Eidenbenz, R. Conejo (Eds.), *Automata, languages and programming, 29th international colloquium, ICALP 2002, Malaga, Spain, July 8–13, 2002, Proceedings, Lecture notes in computer science* (Vol. 2380, p. 37). Springer. https://doi.org/10.1007/3-540-45465-9_4.
- Shanker, V. K., & Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27, 511–545.
- Shinohara, T. (1991). Inductive inference of monotonic formal systems from positive data. *New Generation Computing*, 8(4), 371–384.

- Sleator, D. D., & Temperley, D. (1995). Parsing english with a link grammar. CoRR [arXiv:abs/cmp-lg/9508004](https://arxiv.org/abs/cmp-lg/9508004).
- Wright, K. (1989). Identification of unions of languages drawn from an identifiable class. In: R.L. Rivest, D. Haussler, M.K. Warmuth (Eds.), *Proceedings of the 2nd annual workshop on computational learning theory, COLT 1989, Santa Cruz, CA, USA, July 31–August 2, 1989* (pp. 328–333). Morgan Kaufmann.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.