JMLR: Workshop and Conference Proceedings 57:3–14, 2016

Simple K-star Categorial Dependency Grammars and their Inference

Denis Béchet DENIS.BECHET@UNIV-NANTES.FR
LINA UMR CNRS 6241
Université de Nantes, France

Annie Foret ANNIE.FORET@IRISA.FR
IRISA
Université de Rennes 1, France

Abstract

We propose a novel subclass in the family of Categorial Dependency Grammars (CDG), based on a syntactic criterion on categorial types associated to words in the lexicon and study its learnability. This proposal relies on a linguistic principle and relates to a former non-constructive condition on iterated dependencies. We show that the projective CDG in this subclass are incrementally learnable in the limit from dependency structures. In contrast to previous proposals, our criterion is both syntactic and does not impose a (rigidity) bound on the number of categorial types associated to a word.

Keywords: Grammatical inference, Categorial grammar, Dependency grammar, Incremental learning, Iterated dependencies, Computational linguistics, Dependency Treebanks, Context-free languages.

1. Introduction

The paper studies the problem of inference of a dependency grammar from positive examples of dependency trees or structures it generates. Categorial Dependency Grammars (*CDG*, Dekhtyar et al., 2015) the grammars considered in this paper, are a unique class of grammars directly generating unbounded dependency structures (DS), beyond context-freeness. They are well adapted to real NLP applications and are analysed in tractable polynomial time.[1] 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. The iterated dependencies are due to the basic principles of dependency syntax, which concern optional repeatable dependencies (cf. Mel'čuk, 1988): All modifiers of a noun n share n as their *governor* and, similarly, all modifiers of a verb v share v as their *governor*. At the same time, as we explain below, the iterated dependencies are a challenge for grammatical inference.

1. The dependency treebank CDGFr (Béchet and Lacroix, 2015), a large scale CDG of French and a general purpose deterministic parser have been implemented (Dikovsky, 2011; Lacroix and Béchet, 2014).

In Béchet et al. (2004) it was shown that, in contrast with the classical categorial grammars, the *rigid* CDG[2] are not learnable in the limit[3]. This negative effect is due to the use of iterated dependency types which express iterated dependencies. On the other hand, it was also shown that the k-valued CDG[4] 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 constraint, called below K-*star-revealing* has been introduced in Béchet et al. (2010), enabling learnability. Intuitively, under this constraint, the iterated dependencies and the dependencies repeatable at least K times for some fixed K are *indiscernible*. However this constraint relies on non-constructive condition on iterated dependencies. In contrast, below, we introduce a new constructive syntactic criterion on types called simple K-star, enabling learnability. We also compare the two notions.

The paper is organized as follows. Section 2 contains all background notions and facts, in particular, those concerning Categorial Dependency Grammars and learnability from positive examples. Section 3 introduces the new notion of simple K-star grammars. Section 4 presents the learning algorithm. Section 5 recalls the notion of K-star-revealing CDG that are learnable with a given inference algorithm. Section 6 establishes new results on simple K-star grammars: Their learnability and their comparison with K-star-revealing CDG.

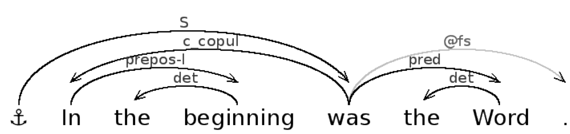
2. Background

2.1. Categorial Dependency Grammars

The lexicon of a *Categorial Dependency Grammar* may be seen as an assignment to words of first order dependency types of the form: $t = [l_m\backslash \ldots \backslash l_1 \backslash g / r_1 / \ldots / r_n]^P$. Intuitively, $w \mapsto [\alpha\backslash d\backslash\beta]^P$ means that the word w has a left subordinate through dependency d (similar for the right part $[\alpha/d/\beta]^P$). Similarly $w \mapsto [\alpha\backslash d^*\backslash\beta]^P$ means that w may have $0, 1$ or several left subordinates through dependency d. The *head type* g in $w \mapsto [\alpha\backslash g/\beta]^P$ means that w is governed through dependency g. The P exponent, called *potential*, is used for non-projective dependencies (beyond context-free) and will remain empty in this article.

Example 1 *The assignment on the left yields the structure on the right :*

in	$\mapsto$	$[c\_copul/prepos{-}l]$
the	$\mapsto$	$[det]$
beginning	$\mapsto$	$[det\backslash prepos{-}l]$
was	$\mapsto$	$[c\_copul\backslash S/@fs/pred]$
word	$\mapsto$	$[det\backslash pred]$
.	$\mapsto$	$[@fs]$



Projective dependency structure.

Definition 1 (CDG dependency structures) *Let $W = a_1 \ldots a_n$ be a list of words and $\{d_1, \ldots, d_m\}$ be a set of* dependency names. *A directed graph $D = (W, E)$ whose nodes are the words of W and whose arcs (a, d, a') are labeled by a dependency name in $\{d_1, \ldots, d_m\}$ 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).*[5] *An*

2. A rigid grammar associates at most one type to each word.
3. Here, the learning mechanism means Gold style identification in the limit from positive examples
4. A k-valued grammar associates at most k types to each word.
5. Evidently, every DS is connected and has a unique root.

arc $(a, d, a') \in E$ *is called* dependency d *from* a *to* a'. a *is called a* governor *of* a' *and* a' *is called a* subordinate *of* a *through* d. *The* precedence order *on* D *is the linear order on* W.

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

An expression of the form d^* *where* $d \in \mathbf{C}$*, is called* iterated dependency type*. Dependency names and iterated dependency types are* primitive types.

An expression of the form $t = [l_m\backslash \ldots \backslash l_1 \backslash H / r_1 \ldots / r_n]$ *in which* $m, n \geq 0$*,* $l_1, \ldots, l_m$*,* $r_1, \ldots, r_n$ *are primitive types and* H *is a dependency name is called a* basic dependency type*.* $l_m, \ldots, l_1$ *and* $r_1, \ldots, r_n$ *are respectively successive left and successive right* argument types *of* t*.* H *is called the* head type *of* t*.*

In this paper we consider only *basic dependency type* (i.e. with an empty potential)[6]. In this context, DS are projective trees. CDG are defined using the following calculus of dependency types.[7] In these rules, types must be adjacent.

Definition 3 (Relativized calculus of dependency types)

$\mathbf{L^l}$. $(C, i_1)([C\backslash\beta], i_2) \vdash ([\beta], i_2)$ *(classical elimination rule)*

$\mathbf{I^l}$. $(C, i_1)([C^*\backslash\beta], i_2) \vdash ([C^*\backslash\beta], i_2)$

$\mathbf{\Omega^l}$. $([C^*\backslash\beta], i) \vdash ([\beta], i)$

These rules are relativized with respect to the word positions in the sentence, using state (B, i) for a type B assigned to word at position i, which allows to interpret them as rules of construction of DS. We may suppress word positions, when the context is clear.

DS. Eliminating the argument type C in $\mathbf{L^l}$ constructs a (*projective*) dependency C. For every proof ρ in this calculus, 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 $R = \mathbf{L^l}$ or $R = \mathbf{I^l}$, then $DS_x((\rho, R)) = (W, E \cup \{(a_{i_2}, C, a_{i_1})\})$.

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

Definition 4 (CDG) *A (projective)* categorial dependency grammar (CDG) *is a system* $G = (W, \mathbf{C}, S, \lambda)$*, where* W *is a finite set of words,* $\mathbf{C}$ *is a finite set of dependency names containing the selected name* S *(an axiom), and* λ*, called* lexicon*, is a finite substitution on* W *such that* $\lambda(a)$ *is a subset of dependency types over* $\mathbf{C}$ *for each word* $a \in W$*.* λ *is extended on sequences of words* W^* *in the usual way.*[8]

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

$$D = DS_x(\rho) \quad \begin{array}{l} \textit{where } \rho \textit{ is a proof of } (t_1, 1) \cdots (t_n, n) \vdash (S, j) \\ \textit{for some } n, j,\ 0 < j \leq n \textit{ and } t_1 \cdots t_n \in \lambda(x). \end{array}$$

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 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.*

6. The full calculus is presented in Dekhtyar et al. (2015).
7. We show left-oriented rules. The right-oriented rules are symmetrical.
8. $\lambda(a_1 \cdots a_n) = \{t_1 \cdots t_n \mid t_1 \in \lambda(a_1), \ldots, t_n \in \lambda(a_n)\}$.

CDG are very expressive. Projective CDG generate all CF-languages. CDG with potentials (Dekhtyar et al., 2015) can also generate non-CF languages.

2.2. Learnability and Limit Points

Let $\mathcal{C}$ be a class of grammars. An *observation set* $\Phi(G)$ is related with every $G \in \mathcal{C}$; this may be $L(G)$ or an image of the constituent or dependency structures generated by G.

Definition 5 (Inference algorithm) *Below we call an enumeration of $\Phi(G)$ a* training sequence *for G. An algorithm A is an* inference algorithm *for $\mathcal{C}$ if, for every $G \in \mathcal{C}$, A applies to any training sequence σ for G and, for every initial subsequence $\sigma[i] = \{s_1, \ldots, s_i\}$ of σ, it returns a* hypothesized grammar *$A(\sigma[i]) \in \mathcal{C}$. A* learns *a target grammar $G \in \mathcal{C}$ if on any training sequence σ for G A stabilizes on a grammar $\mathcal{A}(\sigma[T]) \equiv G$.*[9] *The grammar $\lim\limits_{i\to\infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ returned at the stabilization step is the* limit grammar. *A* learns *$\mathcal{C}$ if it learns every $G \in \mathcal{C}$. $\mathcal{C}$ 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, in Wright (1989); Motoki et al. (1991) finite elasticity, implying learnability, was introduced. We use here the related concept of limit points.

Definition 6 (Limit points) *A class $\mathcal{L}$ of languages has* a limit point *iff there exists an infinite sequence $(L_n)_{n\in N}$ of languages in $\mathcal{L}$ and a language $L \in \mathcal{L}$ such that: $L_0 \subsetneq L_1 \ldots \subsetneq \ldots \subsetneq L_n \subsetneq \ldots$ and $L = \bigcup_{n\in N} L_n$ (L is* a limit point *of $\mathcal{L}$).*

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

2.3. Limit Points Construction for CDG with Iterated Types

In Béchet 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 Béchet et al. (2004).

Theorem 1 *Let S, A, B be dependency names. Grammars G_n, G_∞ are defined as follows:*

$t_0 = S$	$\lambda_n = \{a \mapsto [A], b \mapsto [B], c \mapsto [t_n]\}$	$G_n = (\{a,b,c\}, \{A,B,S\}, S, \lambda_n)$
$t_{n+1} = t_n/A^*/B^*$	$\lambda_\infty = \{a, b \mapsto [A], c \mapsto [S/A^*]\}$	$G_\infty = (\{a,b,c\}, \{A,S\}, S, \lambda_\infty)$

The type assigned to c by G_n is $[S/A^*/B^*/\cdots/A^*/B^*]$ where the pattern $/A^*/B^*$ appears n times. These constructions yield a limit point $\bigcup_{k\le n} c(b^*a^*)^k \subseteq c\{b,a\}^*$ and show the non-learnability from strings for the classes of (rigid) grammars allowing iterative types (X^*).

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

3. Simple K-star Grammars

We introduce a new syntactic criterion on categorial grammar types, leading to the definition of simple K-star grammars.[10]

9. $\mathcal{A}$ *stabilizes* on σ on step T means that T is the minimal number t for which there is no $t_1 > t$ such that $\mathcal{A}(\sigma[t_1]) \neq \mathcal{A}(\sigma[t])$.
10. By "simple" we mean here "un-nested"

Definition 7 (Simple K-star) *Let $K > 1$ be an integer. Let t denote a categorial type and d denote a dependency name. t is said* simple left K-star on d *if for any successive occurrences $l_1\backslash l_2\backslash ...l_p\backslash$ on the left where each l_i is either d or some x^*, there are: (2.1) at most $K-1$ occurrences of d and (2.2) no occurrence of d if there exists at least one occurrence of d^*. t is said* 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.*
A type t is said simple K-star *if it is* simple left K-star *and* simple right K-star*.*
A CDG G is said simple K-star *whenever all types in its lexicon are simple K-star.*
The class of CDG that are simple K-star is noted $\mathcal{CDG}^{K\sim *}$.

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

Limit point. The grammars in Theorem 1 are simple K-star ($\forall K{>}1$). The class of rigid simple 2-star CDG is thus unlearnable from strings (also for any $K{>}1$ or non rigid class).

4. Inference Algorithm

We show an algorithm *strongly* learning CDG from DS. This means that $\Delta(G)$ serves as the observation set $\Phi(G)$ and the limit grammar is *strongly* equivalent to the target grammar.

Definition 8 (Strong equivalence) *Let G_1, G_2 be CDG, $G_1 \equiv_s G_2$ iff $\Delta(G_1) = \Delta(G_2)$. G_1, G_2 are then said strongly equivalent.*

Note that in contrast with the constituent structure grammars and also with classical categorial grammars, the existence of such learning algorithm is not guaranteed because, due to the iterated types, the straightforward arguments of subformulas' 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*. Another advantage of this algorithm is that it can be applied on many linguistic treebanks as those in the CoNLL format[11], possibly with universal dependencies[12] developped cross-linguistically.

Definition 9 (Vicinity) *Let D be a DS in which an occurrence of a word w has the incoming dependency h (or the axiom S), the left dependencies $l_k, \dots, l_1$ (in this order), the right dependencies $r_1, \dots, r_m$ (in this order). Then the* vicinity *of w in D is the type*

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

Definition 10 (Algorithm) *We present an inference algorithm* $\mathbf{TGE}^{(K)}$ *(see Figure 1) which, for every next DS in a training sequence, transforms the observed dependencies of every word into a type with repeated dependencies by introducing iteration for each group of at least K consecutive dependencies with the same name.*

Algorithm $\mathbf{TGE}^{(K)}$ (type-generalize-expand):
Input: σ, a training sequence of length N.
Output: CDG $\mathbf{TGE}^{(K)}(\sigma)$.

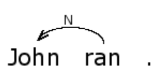
let $G_H = (W_H, \mathbf{C}_H, S, \lambda_H)$ where $W_H := \emptyset$; $\mathbf{C}_H := \{S\}$; $\lambda_H := \emptyset$;
(loop) **for** $i = 1$ **to** N // loop on σ
  let D such that $\sigma[i] = \sigma[i-1] \cdot D$; // the i-th DS of σ
  let $(X, E) = D$;
  (loop) **for every** $w \in X$ // the order of the loop is not important
    $W_H := W_H \cup \{w\}$;
    let $t_w = V(w, D)$ // the vicinity of w in D
    (loop) **while** $t_w = [\alpha\backslash l\backslash \mathbf{d}\backslash \cdots \backslash \mathbf{d}\backslash r\backslash \beta]$
      with at least K consecutive occurrences of d, $l \neq d$ (or not present) and $r \neq d$ (or not present)
        $t_w := [\alpha\backslash l\backslash d^*\backslash r\backslash \beta]$
    (loop) **while** $t_w = [\alpha/l/\mathbf{d}/\cdots/\mathbf{d}/r/\beta]$
      with at least K consecutive occurrences of d, $l \neq d$ (or not present) and $r \neq d$ (or not present)
        $t_w := [\alpha/l/d^*/r/\beta]$
    $\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // lexicon expansion
    end end
return G_H

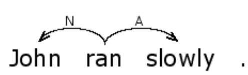
Figure 1: *Inference algorithm* $\mathbf{TGE}^{(K)}$.

Example 3 *We illustrate* $\mathbf{TGE}^{(2)}$ *with the following CDG* G_{target} *as target grammar:*

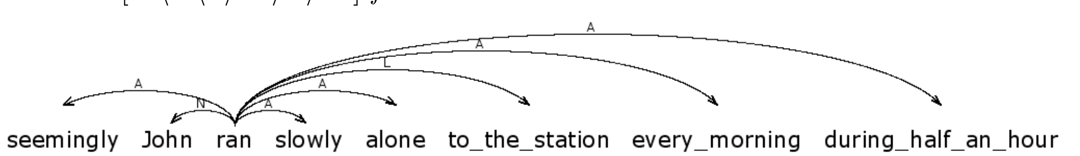
$$John \mapsto [N] \qquad to\_the\_station \mapsto [L]$$
$$ran \mapsto [N\backslash A^*\backslash S/A^*/L/A^*],\ [N\backslash A^*\backslash S/A^*]$$
$$seemingly,\ slowly,\ alone,\ during\_half\_an\_hour,\ every\_morning \mapsto [A]$$

Algorithm $\mathbf{TGE}^{(2)}$ *on* $(\sigma[i])$ *will add for ran:*

$ran \mapsto [N\backslash S]$ *for* $(i=1)$*:* 

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

etc...
$ran \mapsto [N\backslash A\backslash S/A^*/L/A^*]$ *for:*



The algorithm also assigns from this training sequence :
$John \mapsto [N] \qquad to\_the\_station \mapsto [L]$
$seemingly,\ slowly,\ alone,\ during\_half\_an\_hour,\ every\_morning \mapsto [A]$

11. http://ilk.uvt.nl/conll/#dataformat
12. http://universaldependencies.org/introduction.html

5. Learning K-star Revealing Grammars

As we explain in Section 3, the unlearnability of rigid CDG is due to the use of iterated types. In natural languages, the optional dependencies that are repeated successively several times are exactly the iterated dependencies. We use and formalize these properties to resolve the learnability problem ; a main definition concerns a restriction on the class of grammars that is learned, 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 Béchet et al. (2010).

Definition 11 (K-star generalization) *Let $K > 1$ be an integer and G be a CDG. For a word w having a type assignment $w \mapsto t$ in G and for a dependency name d, we suppose that t can be written as $[l_1\backslash \cdots \backslash l_a\backslash t_1\backslash \cdots \backslash t_p\backslash m_1\backslash \cdots \backslash m_b\backslash h/r_1/\cdots/r_c]$ where every $t_1, \ldots, t_p$ is either d or some iterated dependency type x^* and among $t_1, \ldots, t_p$ there are* **at least** K **occurrences of** d **or at least one occurrence of** d^*.
$\mathcal{C}^K(G)$, the K-star-generalization CDG of G, is defined by recursively adding, for every assignment $w \mapsto t$ of G and every dependency name d as above, the types
$[l_1\backslash \cdots \backslash l_a\backslash d^*\backslash m_1\backslash \cdots \backslash m_b\backslash h/r_1/\cdots/r_c]$ *and* $[l_1\backslash \cdots \backslash l_a\backslash m_1\backslash \cdots \backslash m_b\backslash h/r_1/\cdots/r_c]$
Symmetrically, corresponding types are added if $t_1, \ldots, t_p$ appear in the right part of t.

Example 4 *For instance, with $K = 2$, for the type $[a\backslash b^*\backslash a\backslash S/a^*]$, it adds $[a\backslash a\backslash S/a^*]$ and $[a\backslash b^*\backslash a\backslash S]$ but also $[a^*\backslash S/a^*]$ and $[S/a^*]$. Recursively, it also adds $[a\backslash a\backslash S]$, $[a^*\backslash S]$ and $[S]$. The size of $\mathcal{C}^K(G)$ can be exponential with respect to the size of G.*

Definition 12 (K-star revealing) *Let $K > 1$ be an integer. A CDG G is K-star revealing if $\mathcal{C}^K(G) \equiv_s G$. The class of CDG that are K-star revealing is noted $\mathcal{CDG}^{K\to*}$.*

Example 5 *The grammars $G(t)$ of Example 2 are 2-star revealing for $t \in \{t_1, t_2, t_3\}$ but not 2-star revealing for $t \in \{t_4, t_5, t_6\}$*

Theorem 13 *The class $\mathcal{CDG}^{K\to*}$ of K-star revealing CDG is learned from DS by the inference algorithm* $\mathbf{TGE}^{(K)}$ *(see Figure 1 and Appendix A).*

Theorem 13 results from Lemma 20 and Lemma 21 and further definitions. See Appendix A.

6. Simple K-star Grammars and K-star Revealing Grammars

A K-star revealing grammar G is a CDG such that $\mathcal{C}^K(G) \equiv_s G$. 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 potential).

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 14 *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 $\Delta(\mathcal{C}^K(G)) = \Delta(G)$. 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, they are:

$[l_1\backslash\cdots\backslash l_a\backslash d^*\backslash m_1\backslash\cdots\backslash m_b\backslash h/r_1/\cdots/r_c]$ and $[l_1\backslash\cdots\backslash l_a\backslash m_1\backslash\cdots\backslash m_b\backslash h/r_1/\cdots/r_c]$ when w has an assignment $w \mapsto t$ where $t = [l_1\backslash\cdots\backslash l_a\backslash t_1\backslash\cdots\backslash t_p\backslash m_1\backslash\cdots\backslash m_b\backslash h/r_1/\cdots/r_c]$, every $t_1, \ldots, t_p$ is either d or some iterated dependency type x^* and among $t_1, \ldots, t_p$ 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 $t_1, \ldots, t_p$ of t contain at most $K-1$ occurrences of d and contain no occurrence of d or no occurrence of d^*. It means that $t_1, \ldots, t_p$ 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 $t'_1\backslash\cdots\backslash t'_q$ 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 $t'_1\backslash\cdots\backslash t'_q$, 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 $t_1\backslash\cdots\backslash t_p$ is inserted in $t'_1\backslash\cdots\backslash t'_q$ or replaces d^* in $t'_1\backslash\cdots\backslash t'_q$. As a consequence, $t'_1\backslash\cdots\backslash t'_q$ 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 $\mathcal{C}^K(G)$ that is equivalent to G: G is K-star revealing. ■

Corollary 15 *The class $\mathcal{CDG}^{K\sim*}$ of simple K-star CDG is learned from DS 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 $[A\backslash A\backslash S]$): $x \mapsto [S]$; $a \mapsto [A\backslash A\backslash S]$.

A more interesting example is given by $x \mapsto [S], [A\backslash A^*\backslash S]$; $a \mapsto [A]$: 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 $[A\backslash A^*\backslash S]$ contains the two successive types A and A^* on the left. The completion mechanism gives the following grammar: $x \mapsto [S], [A\backslash A^*\backslash S], [A\backslash S], [A^*\backslash S]$; $a \mapsto [A]$; 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 16 *Let G_2 be the 2-star revealing grammar :*

$$x \mapsto [A\backslash B^*\backslash A\backslash S], [A^*\backslash S] \qquad a \mapsto [A] \qquad b \mapsto [B]$$

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 $[t_1\backslash \cdots \backslash t_p\backslash S]$ 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 $[A\backslash B^*\backslash \cdots \backslash B^*\backslash A\backslash S]$. But it is not possible because $A\backslash B^*\backslash \cdots \backslash B^*\backslash A$ contains 2 occurrences A separated by iterated types and this sequence is forbidden in a simple 2-star grammar. ∎

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: $x \mapsto [A^*\backslash B^*\backslash A^*\backslash S], [A^*\backslash S]$; $a \mapsto [A]$; $b \mapsto [B]$.

7. Conclusion

In this paper, we have replaced a non-constructive criterion on CDG grammars by a syntactic constructive one that is slightly more restrictive ; we have shown that the new class is learnable from dependency structures. This work has been developped in the computational linguistic domain. It would be interesting to reconsider these notions in a purely theoretical way (languages and automaton) or other application domains. In the lexicalized grammar setting, some possible variants could also be explored.

Appendix A. Proof Details

Definition 17 (Monotonic, faithful, expansive and incremental) *Let $\preceq$ be a partial order on CDG which denotes a generalization relation on the lexicons of CDG. Let $\mathcal{A}$ be an inference algorithm for CDG from DS and σ be a training sequence for a CDG G.*
1. $\mathcal{A}$ is monotonic *on σ (w.r.t. $\preceq$) if $\mathcal{A}(\sigma[i]) \preceq \mathcal{A}(\sigma[j])$ 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. $\preceq$) when it satisfies properties 1, 2 and 3.*[13]

Definition 18 ($\preceq_{cr}$ (Consecutive repetitions)) *1. for all $i \geq 0$, $0 \leq j \leq m$, $n \geq 0$:*

$$[l_m\backslash \cdots \backslash l_j\backslash \mathbf{c} \cdots \backslash \mathbf{c}\backslash l_{j-1}\backslash \cdots l_1\backslash g/r_1 \cdots /r_n] <_{cr} [l_m\backslash \cdots \backslash l_j\backslash \mathbf{c}^*\backslash l_{j-1}\backslash \cdots l_1\backslash g/r_1 \cdots /r_n]$$

and for all $i \geq 0$, $0 \leq k \leq n$, $m \geq 0$:

$$[l_m\backslash \cdots l_1\backslash g/r_1 \cdots /r_{k-1}/\mathbf{c} \cdots /\mathbf{c}/r_k/ \cdots /r_n] <_{cr} [l_m\backslash \cdots l_1\backslash g/r_1 \cdots /r_{k-1}/\mathbf{c}^*/r_k/ \cdots /r_n]$$

where c is repeated successively i times in $\mathbf{c}\backslash \cdots \backslash \mathbf{c}\backslash$ or in $\mathbf{c}/ \cdots /\mathbf{c}/$ accordingly.
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. $\preceq_{cr}$ is the PO (partial order) which is the reflexive-transitive closure of the preorder $<_{cr}$

Lemma 1 *Let G_1, G_2 be CDG. If $G_1 \preceq_{cr} G_2$ Then $\Delta(G_1) \subseteq \Delta(G_2)$ and $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$.*

Definition 19 (Repetition blocks, patterns and superposition)
1. Repetition blocks *(R-blocks) : for $d \in \mathbf{C}$,*
$LB_d = \{t_1\backslash \cdots \backslash t_i \mid i>0, t_1, \ldots, t_i \in \{d, d^*\}\}$ and $RB_d = \{t_1/ \cdots /t_i \mid i>0, t_1, \ldots, t_i \in \{d, d^*\}\}$
Elements of LB_d and of RB_d are called d R-blocks or R-blocks of label d.
2. Patterns *are defined as types, but in the place of $\mathbf{C}$, we use $\mathbf{G}$, where $\mathbf{G}$ is the set of* gaps *$\mathbf{G} = \{\langle d \rangle \mid d \in \mathbf{C}\}$, d is called the label of gap $\langle d \rangle$. Two consecutive gaps cannot have the same label. The head 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, $\pi(\langle d_1 \rangle \leftarrow \beta_1, \ldots, \langle d_m \rangle \leftarrow \beta_m)$ is the expression resulting from π by the parallel substitution of the R-blocks β_i for the corresponding gaps $\langle d_i \rangle$.
(ii) Let E be a type or a vicinity, π is superposable *on E if: $E = \pi(\langle d_1 \rangle \leftarrow \beta_1, \ldots, \langle d_m \rangle \leftarrow \beta_m)$ for some $\langle d_1 \rangle, \ldots, \langle d_m \rangle, \beta_1, \ldots, \beta_m$, such that all β_i are R-blocks of label d_i.*

Lemma 2 *For every vicinity V there is a single pattern π superposable on V and a single decomposition (called* R-decomposition*): $V = \pi(\langle d_1 \rangle \leftarrow \beta_1, \ldots, \langle d_m \rangle \leftarrow \beta_m)$*

Proof This 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. ■

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

Lemma 3 *For $D \in \Delta(G)$ and a word occurrence w, let π denote the pattern superposable on the vicinity of w in D: $V(w, D)$. There exists a type t that is assigned to w in $\mathcal{C}^K(G)$ 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 $V(w, D) = \pi(\langle d_1\rangle \leftarrow \beta_1, \ldots, \langle d_m\rangle \leftarrow \beta_m)$. The vicinity of w in D must match t. 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 $x \in \mathbf{C}$ 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. ■

Lemma 20 *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 $\Delta(G)$, we show $\mathbf{TGE}^{(K)}(\sigma[i]) \preceq_{cr} \mathcal{C}^K(G)$. 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 $t = t_1 <_{cr} \cdots <_{cr} t_n = t_G$ with $n > 0$, and $t_1, \ldots, t_n$ 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 of a DS D. D is a positive example thus $D \in \Delta(G) = \Delta(\mathcal{C}^K(G))$. By Proposition 3, if π is the pattern superposable 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 superpositions of π for t and t' are : $t = \pi(\langle d_1\rangle \leftarrow \alpha_1, \ldots, \langle d_m\rangle \leftarrow \alpha_m)$ and $t' = \pi(\langle d_1\rangle \leftarrow \beta_1, \ldots, \langle d_m\rangle \leftarrow \beta_m)$. 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 R-block of label d_i. 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 $\beta_i = d_i \cdots d_i$ (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 we must have $l \geq K$ ($\alpha_i = d_i^*$). The type $t'' = \pi(\langle d_1\rangle \leftarrow \beta_1, \ldots, \langle d_i\rangle \leftarrow \alpha_i, \ldots, \langle d_m\rangle \leftarrow \beta_m)$ must be also assigned to w in $\mathcal{C}^K(G)$, 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. ■

Lemma 21 *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. ■

References

Denis Béchet and Ophélie Lacroix. CDGFr, un corpus en dépendances non-projectives pour le français. In *Actes de la 22e conférence sur le Traitement Automatique des Langues Naturelles, June 2015, Caen, France*, pages 522–528. Association pour le Traitement Automatique des Langues, 2015. Short paper in French.

Denis Béchet, Alexander Dikovsky, Annie Foret, and Erwan Moreau. On learning discontinuous dependencies from positive data. In Paola Monachesi, editor, *Proceedings of the 9th International Conference on Formal Grammar August 2004*, pages 1–16. CSLI Publications, 2004. URL http://cslipublications.stanford.edu/FG/2004.

Denis Béchet, Alexander Ja. Dikovsky, and Annie Foret. Two models of learning iterated dependencies. In Philippe de Groote and Mark-Jan Nederhof, editors, *Formal Grammar, 15th and 16th International Conferences, FG 2010, FG 2011, Revised Selected Papers*, volume 7395 of *LNCS*, pages 17–32. Springer, 2010. doi: 10.1007/978-3-642-32024-8_2.

Michael Dekhtyar, Alexander Dikovsky, and Boris Karlov. Categorial dependency grammars. *Theoretical Computer Science*, 579:33 – 63, 2015. ISSN 0304-3975.

Alexander Dikovsky. Categorial Dependency Grammars: from Theory to Large Scale Grammars. In K. Gerdes, E. Hajicova, and L. Wanner, editors, *Conference on Dependency Linguistics 2011*, pages 262–271, Barcelona, Spain, 2011.

Makoto Kanazawa. *Learnable classes of categorial grammars.* Studies in Logic, Language and Information. FoLLI & CSLI, 1998.

Ophélie Lacroix and Denis Béchet. A three-step transition-based system for non-projective dependency parsing. In Jan Hajic and Junichi Tsujii, editors, *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, August 23-29, 2014, Dublin, Ireland*, pages 224–232. ACL, 2014.

Igor Mel'čuk. *Dependency Syntax*. SUNY Press, Albany, NY, 1988.

Tatsuya Motoki, Takeshi Shinohara, and Keith Wright. The correct definition of finite elasticity: Corrigendum to identification of unions. In *The fourth Annual Workshop on Computational Learning Theory*, page 375, San Mateo, Calif., 1991.

Keith Wright. Identification of unions of languages drawn from an identifiable class. In Ronald L. Rivest, David Haussler, and Manfred K. Warmuth, editors, *Proceedings of the Second Annual Workshop on Computational Learning Theory, COLT 1989, Santa Cruz, CA, USA, July 31 - August 2, 1989.*, pages 328–333. Morgan Kaufmann, 1989.