On Disperse an Choice Iteration in Incrementally Learnable Depen ency Types

Denis Béchet[1], Alexandre Dikovsky[1], and Annie Foret[2]

[1] LINA UMR CNRS 6241, Université de Nantes, France
Denis.Bechet@univ-nantes.fr,
Alexandre.Dikovsky@univ-nantes.fr
[2] IRISA, Université de Rennes1, France
Annie.Foret@irisa.fr

Abstract. We study learnability of Categorial Dependency Grammars (CDG), a family of categorial grammars expressing all kinds of projective, discontinuous and repeatable dependencies. For these grammars, it is known that they are not learnable from dependency structures.
We propose two different ways of modelling the repeatable dependencies through iterated types and the two corresponding families of CDG which cannot distinguish between the dependencies repeatable at least K times and those repeatable `any` number of times. For both we show that they are incrementally learnable in the limit from dependency structures.

Keywords: Grammatical inference, Categorial grammar, Dependency grammar, Incremental learning, Iterated types.

1 Intro uction

Languages generated by grammars in a class $\mathcal{G}$ are `learnable` if there is an algorithm A which, for every target grammar $G_T \in \mathcal{G}$ and every finite set σ of generated words, computes a hypothetical grammar $A(\sigma) \in \mathcal{G}$ in a way that:
(i) the sequence of languages generated by the grammars $A(\sigma)$ converges to the target language $L(G_T)$ and
(ii) this is true for any increasing enumeration of sub-languages $\sigma \subset L(G_T)$.

This concept due to E.M. Gold [10] is also called `learning from strings`. More generally, the hypothetical grammars may be computed 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 the surveys in [1] and in [11]). In particular, it is known that 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. At the same time, some interesting positive results were also obtained. In particular, k-rule string and term generating grammars are learnable from strings for every k [14] and k-`rigid` (i.e. assigning no more than k types per word) classical categorial grammars (CG) are learnable from the so called "function-argument" structures and also from strings [4, 11].

In our recent paper [2], we adapt this concept of learning to surface dependency structures (DS), i.e. graphs of named binary relations on words, called dependencies (see Fig. 1,2,4). Dependencies are asymmetric. When two words w_1, w_2 are related through dependency d, $w_1 \xrightarrow{d} w_2$, w_1 is called governor and w_2 is called subordinate. Dependencies may be projective, i.e. non-crossing, as in Fig. 1,4, or discontinuous like $clit-a-obj$, $clit-3d-obj$ in Fig. 2. Very importantly, the linguistic intuition behind a dependency name d is that it identifies **all** syntactic and distributional properties of the subordinate in the context of its governor. In more detail, it identifies its syntactic role (e.g., "subject" "direct object", "copula", "attribute", "circumstantial" etc.), its position with respect to the governor and its part of speech (POS). In principle, the words dependent through the same dependency are substitutable (see the quasi-Kunze property in [13]). This might explain why the dependency structure cannot be completely defined through constituent structure with head selection.

Grammars defining dependency relations directly, in conformity with the basic dependency structure principles (see [13]) must face the problem of expressing the so called repeatable dependencies. These dependencies satisfy specific conditions most clearly formulated by I. Mel'čuk in the form of the following Principle of repeatable dependencies (see [13]).

> Every dependency is either *repeatable* or *not repeatable*. If a dependency d is not repeatable, then no word may have two subordinates through d. If d is repeatable, then **any word** g which governs a subordinate word s through d may have **any number** of subordinates through d.

E.g., the verbs may have any number of subordinate circumstantials (but no more than one direct or indirect complement), the nouns may have any number of attributes and of modifiers (but no more than one determiner), etc.

We choose the Categorial Dependency Grammars (CDG) [7, 5] as the grammars to be inferred from dependency structures because these grammars define DS directly, without any order restrictions and in particular, they express the repeatable dependencies through the so called "iterated" types in conformity with the Principle of repeatable dependencies. As it was shown in [3], the k-rigid CDG without iterated types are learnable from analogues of the function-argument structures (and from strings) as it is the case of the classical categorial grammars. At the same time, even rigid (i.e. 1-rigid) CDG with iterated types are not learnable from function-argument structures. Moreover, in [2] we show that they are not learnable from the DS themselves. This may be seen as a proof of unlearnability from dependency treebanks of dependency grammars which express dependency relations in accordance with the basic dependency structure principles (in particular with the Principle of repeatable dependencies). On the other hand, in strict conformity with this Principle, in [2] a subclass of CDG which cannot distinguish between the dependencies repeatable K (or more) times and those repeatable any number of times (the Principle sets $K = 2$) is defined. For these CDG, called in [2] K-star revealing, it is proved that they are incrementally learnable from dependency structures.

It is significant that the Principle of repeatable dependencies is uncertain as it concerns the precedence order of the repeatable subordinates. Let us consider the fragment in Fig. 1 of a DS of the French sentence *Ils cherchaient pendant une semaine surtout dans les quartiers nord un des deux évadés en bloquant systématiquement les entrées - sorties* (*fr.* **They tracked for a week especially in the nord quarters one of the two fugitives systematically blocking entries and exits*). For instance, for $K = 3$, the dependency *circ* is repeatable or not depending on how are counted its occurrences: all together or separately on the left and on the right of the direct complement *évadés*.

Fig. 1. Repeatable dependencies

In [2] is considered the simplest interpretation of `repeatable` dependencies as `consecutively repeatable`. This reading cannot be linguistically founded (even if the consecutively repeatable dependencies are the most frequent). In this paper two other readings of the repeatability are considered. One reading is maximally liberal and says that a subordinate through a repeatable dependency may be found anywhere on the left (or on the right) of the governor. We call such iteration `dispersed`. The other reading is closer to the consecutive one, but extends it with the disjunctive choice of repeatable dependencies which may occur in the same argument position. Respectively, we consider two extensions of the CDG: one with the dispersed iteration types (called `dispersed iteration` CDG) and the other with the choice iteration types (called `choice iteration` CDG). For both we consider the corresponding notion of K-star revealing: the `dispersed` K`-star revealing` and the `choice` K`-star revealing`. We show that both classes are incrementally learnable in the limit from dependency structures.

The plan of this paper is as follows. Section 2 introduces the background notions: Categorial Dependency Grammars, dispersed and choice iterations. Section 3, presents the notion of incremental learning in the limit. In Section 4, the condition of K-star revealing is adapted to dispersed iteration CDG and their incremental learnability from dependency structures is proved. Section 5 presents a similar result for choice iteration CDG.

2 Categorial Depen ency Grammars with Exten e Iteration Types

2.1 Categorial Dependency Grammars

`Categorial Dependency Grammars` (`CDG`) define projective dependency structures assigning to every word a set of first order types in which the argument subtypes determine the outgoing dependencies of the word and the head subtype determines its incoming dependency. They also define discontinuous dependencies through the so called potentials of the types, i.e. strings of `polarized valencies`. Every positive valency in the potential of a word's type determines the name and the direction of an outgoing dependency of the word, and every negative valency determines the name and the direction of the word's incoming dependency. The correspondence between the `dual` valencies (i.e. those having the same name and direction and the opposite signs) is established using general valency pairing principles such as **FA**: *Two dual valencies which are first available in the indicated direction may be paired.* In this way, the CDG define the dependency structures in the most direct and natural way and without any restrictions to the word order. Definitions, motivation, illustrations and properties of various classes of CDG may be found in [7, 8, 5, 6].

Definition 1. *Let* $\mathbf{C}$ *be a set of* `dependency names` *and* $\mathbf{V}$ *be a set of* `valency names`. *The expressions of the form* $\swarrow v$, $\nwarrow v$, $\searrow v$, $\nearrow v$, *where* $v \in \mathbf{V}$, *are called* `polarized valencies`. $\nwarrow v$ *and* $\nearrow v$ *are* `positive`, $\swarrow v$ *and* $\searrow v$ *are* `negative`; $\nwarrow v$ *and* $\swarrow v$ *are* `left`, $\nearrow v$ *and* $\searrow v$ *are* `right`. *Two polarized valencies with the same valency name and orientation, but with the opposite signs are* `dual`.

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

An expression of the form $t = [l_m \backslash \ldots \backslash l_1 \backslash H / \ldots / r_1 \ldots / r_n]$ *in which* $m, n \geq 0$, $l_1, \ldots, l_m, r_1, \ldots, r_n$ *are primitive types and* H *is either a dependency name or an anchor type, is called* `basic dependency type`. $l_1, \ldots, l_m$ *and* $r_1, \ldots, r_n$ *are respectively left and right* `argument subtypes` *of* t. H *is called* `head subtype` *of* t *(or* `head type` *for short).*

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

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

CDG are defined using the following calculus of dependency types [3] (with $C \in \mathbf{C}$, $H \in \mathbf{C}$ or an anchor, $V \in \mathbf{V}$, a basic type α and a residue of a basic type β):

$\mathbf{L^l}$. $H^{P_1}[H \backslash \beta]^{P_2} \vdash [\beta]^{P_1 P_2}$

$\mathbf{I^l}$. $C^{P_1}[C^* \backslash \beta]^{P_2} \vdash [C^* \backslash \beta]^{P_1 P_2}$

$\mathbf{\Omega^l}$. $[C^* \backslash \beta]^P \vdash [\beta]^P$

[3] We show left-oriented rules. The right-oriented are symmetrical.

D$^{\mathbf{l}}$. $\alpha^{P_1(\swarrow V)P(\nwarrow V)P_2} \vdash \alpha^{P_1PP_2}$, if the potential $(\swarrow V)P(\nwarrow V)$ satisfies the following pairing rule **FA** (*first available*):

FA : P has no occurrences of $\swarrow V, \nwarrow V$.

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

To compute the DS from proofs, these rules should be relativized with respect to the word positions in the sentence. To this end, when a type $B^{v_1 \cdots v_k}$ is assigned to the word in a position i, it is encoded using the `state` $(B, i)^{(v_1,i)\ldots(v_k,i)}$. The corresponding relativized state calculus is shown in [2]. In this calculus, for every proof ρ represented as a sequence of rule applications, one may define the DS `constructed` in this proof for a sentence x and written $DS_x(\rho)$.

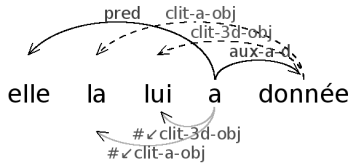
Definition 2. *A* `categorial dependency grammar (CDG)` *is a system* $G = (W, \mathbf{C}, \mathbf{V}, S, \lambda)$, *where* W *is a finite set of words,* $\mathbf{C}$ *is a finite set of dependency names containing the selected name* S *(an axiom),* $\mathbf{V}$ *is a finite set of valency names, and* λ, *called* `lexicon`, *is a finite substitution on* W *such that* $\lambda(a) \subset \mathbf{CAT}(\mathbf{C}, \mathbf{V})$ *for each word* $a \in W$.

For 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 the set* $\Delta(G) =_{df} \{D \mid \exists w\ G(D, w)\}$. $G_1 \equiv_s G_2$ *iff* $\Delta(G_1) = \Delta(G_2)$.

CDG are more expressive than CF-grammars (see [5, 6]) and analyzed in polynomial time. In fact, they are equivalent to real time pushdown automata with independent counters [12]. Importantly, they express discontinuous DS in a direct and natural way. For instance, the DS in Fig. 2 is generated using the following



(*fr.* **she* $it_{\mathbf{g}=fem}$ *to him has given*)

Fig. 2. Non-projective dependency structure

type assignment:
$elle \mapsto [pred],\ \ la \mapsto [\#(\swarrow clit{-}a{-}obj)]^{\swarrow clit\text{–}a\text{–}obj}$,

$lui \mapsto [\#(\swarrow clit{-}3d{-}obj)]^{\swarrow clit-3d-obj}$, $donnée \mapsto [aux{-}a{-}d]^{\nwarrow clit-3d-obj \nwarrow clit-a-obj}$,
$a \mapsto [\#(\swarrow clit-3d-obj)\backslash\#(\swarrow clit-a-obj)\backslash pred\backslash S/aux-a-d]$
(see the proof in Fig. 3). The iterated types also allow one to naturally express re-

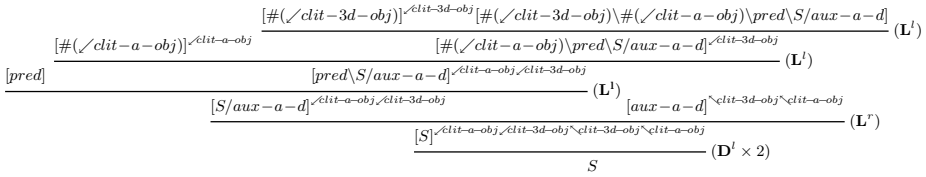
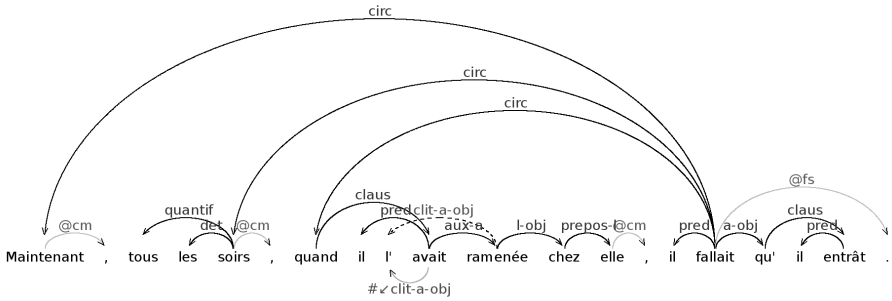


Fig. 3. Dependency structure correctness proof

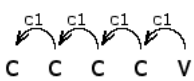
peatable dependencies satisfying the Principle of repeatable dependencies. E.g., the repeatable circumstantial dependency *circ* in Fig. 4 may be determined by the type $[pred\backslash circ^*\backslash S/a-obj]$ assigned to the verb $fallait$ (*had to*). One can



(`fr.` **now all the evenings when he took her home he had to enter* [*M.Proust*])

Fig. 4. Iterated circumstantial dependency

see that such repeatable dependencies are of the kind we call "consecutive" in the Introduction. Iteration-less CDG cannot define such DS. Indeed, the assignments $a \mapsto [\alpha\backslash d]$ and $b \mapsto [d\backslash\beta]$ derive for ab the dependency $a \overset{d}{\longleftarrow} b$. Therefore, the assignments $v \mapsto [c1\backslash S]$, $c \mapsto [c1\backslash c1], [c1]$ will derive for $ccccv$ the sequenced

(not iterated) dependencies as in the DS 

2.2 Dispersed and Choice Iterations

We will consider two different models of repeatable dependencies. One of them, called `dispersed iteration`, represents the case where the subordinates through a repeatable dependency may occur in any position on the left (respectively, on the right) of the governor. The other one, called `choice iteration`, will represent the case where the subordinates through one of several repeatable dependencies may occur in one and the same argument position. To define these models, we extend the primitive types with two new primitives: `dispersed iteration` $\{d_1^*, \ldots, d_k^*\}$ and `choice iteration` $(d_1|\ldots|d_k)^*$, where $d_1, \ldots, d_k$ are dependency names.[4] Respectively we obtain two kinds of extended types.

Definition 3. *1. We call* `dispersed iteration types` *the expressions* B^P *in which* P *is a potential,* $B = [\alpha_1 \backslash L_m \backslash \ldots \backslash L_1 \backslash H / \ldots / R_1 \ldots / R_n / \alpha_2]$, $L_m, \ldots L_1$, H, $R_1 \ldots, R_n$ *are not iterated primitive types and* α_1, α_2 *are dispersed iterations (possibly empty, i.e.* $k = 0$*).*[5]
2. We call `choice iteration types` *the expressions* B^P *where* P *is a potential,* $B = [L_m \backslash \ldots \backslash L_1 \backslash H / \ldots / R_1 \ldots / R_n]$, H *is a not iterated primitive type and* $L_m, \ldots L_1$, $R_1 \ldots, R_n$ *are choice iterations or not iterated primitive types.*
3. Grammars using only dispersed iteration types are called `dispersed iteration CDG`*, those using only choice iteration types are called* `choice iteration CDG`*.*

Here are the respective extensions of the CDG calculus:
1. Choice iteration rules:
IC$^{\mathbf{l}}$. $C^{P_1}[(\alpha_1|C|\alpha_2)^* \backslash \beta]^{P_2} \vdash [(\alpha_1|C|\alpha_2)^* \backslash \beta]^{P_1 P_2}$.
ΩC$^{\mathbf{l}}$. $[(\alpha_1|C|\alpha_2)^* \backslash \beta]^P \vdash [\beta]^P$
LC$^{\mathbf{l}}$ and **DC$^{\mathbf{l}}$** as **L$^{\mathbf{l}}$** and **D$^{\mathbf{l}}$** in the CDG calculus.
2. Dispersed iteration rules:
LD$^{\mathbf{l}}$. $H^{P_1}[\{\alpha\} \backslash H \backslash \beta / \{\gamma\}]^{P_2} \vdash [\{\alpha\} \backslash \beta / \{\gamma\}]^{P_1 P_2}$
ID$^{\mathbf{l}}$. $C^{P_1}[\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^{P_2} \vdash [\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^{P_1 P_2}$
ΩD$^{\mathbf{l}}$. $[\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^P \vdash [\{\alpha_1, \alpha_2\} \backslash \beta / \{\gamma\}]^P$
DD$^{\mathbf{l}}$ as **D$^{\mathbf{l}}$** in the CDG calculus.
The order of elements in dispersed and choice iterations is irrelevant.

It is not difficult to simulate the dispersed iteration CDG through choice iteration CDG. Both are analyzed in polynomial time. As it concerns their weak generative power, both are conservative extensions of the CDG.

3 Incremental Learning

Learning. With every grammar $G \in \mathcal{C}$ is related an `observation set` $\Phi(G)$ of G. This may be the generated language $L(G)$ or an image of the constituent or dependency structures generated by G. Below we call `training sequence` for G an enumeration of $\Phi(G)$. An algorithm A is an `inference algorithm`

[4] Both are used in the flat type expressions of the `compacted` CDG in [9] designed for large scale wide scope grammars.
[5] We suppose that $[\{\}\backslash\beta] = [\beta]$.

for $\mathcal{C}$ if, for every grammar $G \in \mathcal{C}$, A applies to its training sequences σ of $\Phi(G)$ and, for every initial subsequence $\sigma[i] = \{s_1, \ldots, s_i\}$ of σ, it returns a `hypothetical 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 $\mathcal{A}(\sigma[T]) \equiv G$.[6] The grammar $\lim_{i\to\infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ returned at the stabilization step is the `limit grammar`. A `learns` $\mathcal{C}$ if it learns every grammar in $\mathcal{C}$. $\mathcal{C}$ is `learnable` if there is an inference algorithm learning $\mathcal{C}$.

Incremental Learning. Selecting a partial order $\preceq_{\mathcal{C}}$ on the grammars of a class $\mathcal{C}$ compatible with the inclusion of observation sets ($G \preceq_{\mathcal{C}} G' \Rightarrow \Phi(G) \subseteq \Phi(G')$), we can define the following notion of incremental learning algorithm on $\mathcal{C}$.

Definition 4. *Let $\mathcal{A}$ be an inference algorithm for $\mathcal{C}$ and σ be a training sequence for a grammar G.*
1. $\mathcal{A}$ is `monotonic` *on σ if $\mathcal{A}(\sigma[i]) \preceq_{\mathcal{C}} \mathcal{A}(\sigma[j])$ for all $i \leq j$.*
2. $\mathcal{A}$ is `faithful` *on σ if $\Phi(\mathcal{A}(\sigma[i])) \subseteq \Phi(G)$ for all i.*
3. $\mathcal{A}$ is `expansive` *(or* `consistent`*) on σ if $\sigma[i] \subseteq \Phi(\mathcal{A}(\sigma[i]))$ for all i.*
For $G_1, G_2 \in \mathcal{C}$, $G_1 \equiv_s G_2$ iff $\Phi(G_1) = \Phi(G_2)$.

Theorem 1. *Let σ be a training sequence for a grammar G. If an inference algorithm $\mathcal{A}$ is monotonic, faithful, and expansive on σ, and if $\mathcal{A}$ stabilizes on σ then $\lim_{i\to\infty} \mathcal{A}(\sigma[i]) \equiv_s G$.*

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

4 Incremental Learning of Disperse Iteration

In paper [2], we present an incremental learning algorithm for K-**star revealing** CDG which do not distinguish between the dependencies consecutively repeated at least K times and those consecutively repeated any number of times.

Below we change the definition of K-**star revealing** in order to adapt it to the dispersed iteration. We use $\Delta(G)$ as the observation set $\Phi(G)$. So the limit grammar will be `strongly` equivalent to the target grammar G. The notion of incrementality we use is based on a partial "flexibility" order $\preceq_{disp}$ on dispersed iteration CDG. Basically, this PO corresponds to grammar expansion in the sense that $G_1 \preceq_{disp} G_2$ means that G_2 defines no less dependency structures than G_1 and at least as precise dependency structures as G_1. It is the reflexive-transitive closure of the following preorder $<_{disp}$.

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

Definition 5. *1. All occurrences of a dependency name d on the left can be replaced by a single left dispersed iteration of d:*
$[\{fl_1^*, \dots, fl_p^*\}\backslash l_m \backslash \cdots \backslash d \backslash l_i \backslash \cdots \backslash d \backslash \cdots \backslash l_1 \backslash g / r_1 \cdots / r_n / \{fr_1^*, \dots, fr_q^*\}]^P$
$<_{disp}$ $[\{fl_1^*, \dots, fl_p^*, d^*\}\backslash l_m \backslash \cdots \backslash l_1 \backslash g / r_1 \cdots / r_n / \{fr_1^*, \dots, fr_q^*\}]^P$.

2. Symmetrically, all occurrences of a dependency name d on the right can be replaced by a single right dispersed iteration of d.

3. $\tau <_{disp} \tau'$ 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 $t' <_{disp} t''$.

4. $\lambda <_{disp} \lambda'$ for two type assignments λ and λ', if $\lambda(w') <_{disp} \lambda'(w')$ for a word w' and $\lambda(w) = \lambda'(w)$ for all words $w \neq w'$.

5. $\preceq_{disp}$ is the PO which is the reflexive-transitive closure of the preorder $<_{disp}$.

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

Proposition 1. *Let G_1 and G_2 be two CDG such that $G_1 \preceq_{disp} G_2$. Then $\Delta(G_1) \subseteq \Delta(G_2)$ and $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$.*

Below we adapt to the dispersed iteration the basic definitions from [2].

Definition 6.
`Vicinity`*: Let D be a DS in which an occurrence of a word w has : the incoming local dependency 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 discontinuous dependencies $p_1(d_1), \dots, p_n(d_n)$, where $p_1, \dots, p_n$ are polarities and $d_1, \dots, d_n \in \mathbf{V}$ are valency names. Then the* `vicinity` *of w in D is the type*

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

in which P is a permutation of $p_1(d_1), \dots, p_n(d_n)$ in a standard lexicographical order, for instance, compatible with the polarity order $\nwarrow < \searrow < \swarrow < \nearrow$.

Multiple occurrences of a dependency name d in a vicinity $V(w, D)$ correspond to a dispersed iteration $\{f_1^*, \dots, f_p^*, d^*\}$ in the type assigned to w in a proof of D. For instance, in the DS in Fig. 4 the vicinity of the participle *ramenée* is $[aux - a / l - obj]^{\nwarrow clit - a - obj}$. In this DS, $[pred \backslash circ \backslash circ \backslash circ \backslash S / a - obj]$ is the vicinity of the verb *fallait*. This vicinity may be provided by the type assignment $fallait \mapsto [\{circ^*\} \backslash pred \backslash S / a - obj]$.

Definition 7. *Let $K > 1$ be an integer. We define a CDG $\mathcal{C}^K_{disp}(G)$, the* `dispersed` K`-star-generalization` *of G, by recursively adding for every word w and every dependency name d the types*

$$[\{fl_1^*, \dots, fl_p^*, d^*\} \backslash l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \dots, fr_q^*\}]^P$$

and

$$[\{fl_1^*, \dots, fl_p^*\} \backslash l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \dots, fr_q^*\}]^P$$

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

$$t = [\{fl_1^*, \dots, fl_p^*\} \backslash l_m \backslash \cdots d \backslash l_i \backslash \cdots d \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \dots, fr_q^*\}]^P,$$

and t has at least K **occurrences of** *d as left arguments. Symmetrically, we also add the corresponding types if the K occurrences of d appear in the right part of t.*

For instance, with $K = 2$, for the type $[\{x^*\}\backslash a\backslash b\backslash a\backslash S/a]$, will be added the type $[\{a^*, x^*\}\backslash b\backslash S/a]$. The size of $\mathcal{C}^K_{disp}(G)$ can be exponential with respect to the size of G.

Definition 8. *Let $K > 1$ be an integer. CDG G is* **dispersed K-star revealing** *if $\mathcal{C}^K_{disp}(G) \equiv_s G$*

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

- $G([\{a^*\}\backslash b\backslash S/a/b])$, $G([a\backslash b\backslash S])$ and $G([a\backslash b\backslash S/\{a, b\}])$ are all the three dispersed 2-star revealing,
- neither of $G([a\backslash a\backslash S])$, $G([a\backslash b\backslash a\backslash S])$, $G([a\backslash S/b/b])$ is dispersed 2-star revealing.

By Definition 7, we may suppose that no subtype may have K left or K right occurrences in a type of a dispersed K-star revealing grammar.

Theorem 2. *The class $\mathcal{CDG}^{K\rightarrow *}_{disp}$ of dispersed K-star revealing CDG is (incrementally) learnable from DS.*

A proof of this theorem is shown in the Appendix. Algorithm $\mathbf{TGE}^{(K)}_{disp}$ learning the dispersed K-star revealing CDG is shown in Fig. 5.

Remark 1. This algorithm transforms every observed words' vicinity v in DS belonging to $\sigma[i]$ into the least type t such that $v \preceq_{disp} t$. The grammar $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$ is computed in linear time with respect to $|\sigma[i]|$. This algorithm may be easily transformed into a square time algorithm computing in the limit the minimal length grammar among all dispersed K-star revealing grammars strongly equivalent to the target grammar.

5 Incremental Learning of Choice Iteration

In this section, we show an algorithm incrementally learning choice iteration CDG. For the rest of this section, every iterated type d^* is considered as the choice of a unique iterated type (written $(d)^*$). Above this, the lists of types in choice iterations will be considered as sets of types. Thus $(a|b)^* = (b|a)^* = (a|b|a)^*$. The incremental algorithm of [2] may diverge when applied to a choice iteration. For instance, for the CDG $A \mapsto [a]$, $B \mapsto [b]$, $C \mapsto [S/(a|b)^*]$, this algorithm will compute for C all the iteration-less types corresponding to all possible alternations of a and b: $[S]$, $[S/a]$, $[S/b]$ $[S/a/b]$, $[S/b/a]$, $[S/a/b/a]$, $[S/b/a/b]$, etc. It will also compute all possible alternations of a^* and b^*: $[S]$, $[S/a^*]$, $[S/b^*]$ $[S/a^*/b^*]$, $[S/b^*/a^*]$, $[S/a^*/b^*/a^*]$, $[S/b^*/a^*/b^*]$, etc. Thus, the algorithm will diverge. By the way, the algorithm $\mathbf{TGE}^{(K)}_{disp}$ of the preceding

Algorithm $\mathbf{TGE}^{(K)}_{disp}$ (type-generalize-expand):
Input: $\sigma[i]$ (σ being a training sequence).
Output: CDG $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$.
let $G_H = (W_H, \mathbf{C}_H, \mathbf{V}_H, S, \lambda_H)$ 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 $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$
let $LT := \{l_1\} \cup \cdots \cup \{l_m\}$
let $LF := \{d : d \in LT, card(\{i : 1 \leq i \leq m, l_i = d\}) \geq K\}$
let $RT := \{r_1\} \cup \cdots \cup \{r_n\}$
let $RF := \{d : d \in RT, card(\{i : 1 \leq i \leq n, r_i = d\}) \geq K\}$
let $t_w := [\{lf_1^*, \ldots, lf_p^*\} \backslash l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'} / \{rf_1^*, \ldots, rf_q^*\}]^P$
where $\{lf_1, \ldots, lf_p\} = LF$, $\{rf_1, \ldots, rf_q\} = RF$,
where $l'_{m'}, \ldots, l'_1$ is the sublist of $l_m, \ldots, l_1$ without elements in LF
where $r'_1, \ldots, r'_{n'}$ is the sublist of $r_1, \ldots, r_n$ without elements in RF.
$\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // `expansion`
end end

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

section will converge on this CDG and compute a grammar strongly equivalent to the following one: $A \mapsto [a]$, $B \mapsto [b]$, $C \mapsto [S/\{a^*, b^*\}]$ (equivalent to the original CDG). Unfortunately, $\mathbf{TGE}^{(K)}_{disp}$ will not converge for all choice iteration CDG. So we will look for generalizations specific for the choice iteration. In particular, one should not distinguish between several *consecutive* choice iterations and their *union*. For instance, x^* followed by $(y|z)^*$ should be "equivalent" to $(x|y|z)^*$. When applied to the vicinity $[S/a/c/b/b/a]$, in the case of $K = 2$, the learning algorithm should generalize it to $[S/a^*/c/b^*/a^*]$ and then the segment $/b^*/a^*$ should be replaced by a single choice iteration giving the type $[S/a^*/c/(a|b)^*]$.

Incremental learning of choice iteration is based on a PO $\preceq_{ch}$ on CDG, which is the reflexive-transitive closure of the following preorder $<_{ch}$.

Definition 9.

1. *All occurrences of a dependency name d on the left (or on the right) can be replaced by a single iteration of d^*:*
   $[l_m \backslash \cdots \backslash d \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P <_{ch} [l_m \backslash \cdots \backslash d^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   (similar on the right).
2. *Two consecutive choice iterations can be merged:*
   $[l_m \backslash \cdots \backslash (a_1 | \cdots | a_p)^* \backslash (b_1 | \cdots | b_q)^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   $<_{ch} [l_m \backslash \cdots \backslash (a_1 | \cdots | a_p | b_1 | \cdots | b_q)^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   (similar on the right).
3. *Same as 3–5 in Definition 5.*

Algorithm $\mathbf{TGE}_{ch}^{(K)}$ (type-generalize-expand):
Input: $\sigma[i]$ (σ being a training sequence).
Output: CDG $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$.
let $G_H = (W_H, \mathbf{C}_H, \mathbf{V}_H, S, \lambda_H)$ 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 $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$
let $LT := \{l_1\} \cup \cdots \cup \{l_m\}$
let $LI := \{d : d \in LT, card(\{i : 1 \leq i \leq m, l_i = d\}) \geq K\}$
let $RT := \{r_1\} \cup \cdots \cup \{r_n\}$
let $RI := \{d : d \in RT, card(\{i : 1 \leq i \leq n, r_i = d\}) \geq K\}$
for $1 \leq i \leq m$, **let** $l'_i = l_i$ **if** $l_i \notin LI$ **else** $l'_i = l_i^*$
for $1 \leq i \leq n$, **let** $r'_i = r_i$ **if** $r_i \notin RI$ **else** $r'_i = r_i^*$
let $t'_w := [l'_m \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_n]^P$ // (end of) step I
(loop) **while** $\exists i$ such that $l'_i = (x_1 | \cdots | x_p)^*$ and $l'_{i+1} = (y_1 | \cdots | y_q)^*$
$l'_i := (x_1 | \cdots | x_p | y_1 | \cdots | y_q)^*$ and remove l'_{i+1}
(loop) **while** $\exists i$ such that $r'_i = (x_1 | \cdots | x_p)^*$ and $r'_{i+1} = (y_1 | \cdots | y_q)^*$
$r'_i := (x_1 | \cdots | x_p | y_1 | \cdots | y_q)^*$ and remove r'_{i+1}
let $t_w = [l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'}]$ // (end of) step II
$\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // expansion
end end

Fig. 6. *Inference algorithm* $\mathbf{TGE}_{ch}^{(K)}$

Again, it is not difficult to prove that the expressive power of choice iteration CDG monotonically grows with respect to $\preceq_{ch}$.

Definition 10. *Let $K > 1$ be an integer. We define a CDG $\mathcal{C}_{ch}^K(G)$, the* `choice` K`-star-generalization` *of G, by recursively adding for every word w and every dependency name d:*

1. *the type $[l_m \backslash \cdots \backslash d^* \backslash \cdots \backslash d^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ (every occurrence of d on the left is replaced by d^*) when w has a type assignment $w \mapsto t$, where $t = [l_m \backslash \cdots \backslash d \backslash \cdots \backslash d \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$, and d appears at least K times in $l_m, \ldots, d, \ldots, l_1$ or d appears at least one time in a choice iteration of $l_m, \ldots, l_1$,*
2. *the type $[l_m \backslash \cdots \backslash (x_1 | \cdots | x_p | y_1 | \cdots | y_q)^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ when w has a type assignment $w \mapsto t$, where*
$$t = [l_m \backslash \cdots \backslash (x_1 | \cdots | x_p)^* \backslash (y_1 | \cdots | y_q)^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P,$$
3. *the type $[l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ when w has a type assignment $w \mapsto t$, where $t = [l_m \backslash \cdots \backslash d^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$,*
4. *the type $[l_m \backslash \cdots \backslash (x_1 | \cdots | x_p)^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ when w $w \mapsto t$, where $t = [l_m \backslash \cdots \backslash (d | x_1 | \cdots | x_p)^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ and $p > 0$.*

Symmetrically, are also added the corresponding types for the right part of t.

For instance, when $K = 2$, for the type $[b\backslash a\backslash b\backslash a\backslash S/a]$, will be added the type $[b^*\backslash a^*\backslash b^*\backslash a^*\backslash S/a]$. Then will also be added $[(a|b)^*\backslash b^*\backslash a^*\backslash S/a]$, $[b^*\backslash (a|b)^*\backslash a^*\backslash S/a]$, $\ldots$, $[(a|b)^*\backslash S/a]$. Finally, the single iterations a^* or b^* will be erased or $(a|b)^*$ will be transformed into a^* or b^*. The size of $\mathcal{C}_{ch}^K(G)$ can be exponential with respect to the size of G.

Definition 11. *Let $K > 1$ be an integer. CDG G is* **choice K-star revealing** *if $\mathcal{C}_{ch}^K(G) \equiv_s G$*

Remark 2. The generalization induced by points (3) and (4) does not introduce new DS even when G is not **choice K-star revealing**. These points serve only to simplify the proof of learnability.

For instance, if $G(t)$ is the CDG with the assignments: $A \mapsto [a], B \mapsto [b], C \mapsto t$, where t is a type, then we can prove that:

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

Now, without loss of generality we may suppose that in every type of a choice K-star revealing grammar, a subtype d is used at most $K-1$ times on the left and $K-1$ on the right. Besides this, there shouldn't be two consecutive choice iterations.

Theorem 3. *The class $\mathcal{CDG}_{ch}^{K\rightarrow *}$ of choice K-star revealing CDG is (incrementally) learnable from DS.*

A proof of this theorem is shown in the Appendix.

Remark 3. In fact, a similar proof shows that the CDG in $\mathcal{CDG}_{disp}^{K\rightarrow *}$ and $\mathcal{CDG}_{ch}^{K\rightarrow *}$ with potentials of a uniformly bounded length are learnable from strings in the sense of Gold for every K. Meanwhile, Theorems 2 and 3 are stronger because they provide incremental learning algorithms.

Algorithm $\mathbf{TGE}_{ch}^{(K)}$ learning the choice K-star revealing CDG is shown in Fig. 6.

Remark 4. This algorithm transforms every observed words' vicinity v in DS belonging to $\sigma[i]$ into the least type t such that $v \preceq_{ch} t$. The grammar $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$ is computed in linear time with respect to $|\sigma[i]|$. This algorithm too may be transformed into a square time algorithm computing in the limit the minimal length grammar among all choice K-star revealing grammars strongly equivalent to the target grammar.

6 Conclusion

In this paper, we propose two different ways of defining repeatable dependencies through iteration: the dispersed iteration and the choice iteration. Both conform

to the linguistic Principle of repeatable dependencies. We adapt the K-star-revealing condition in [2] to these new models and show that in both cases the CDG satisfying the K-star-revealing are incrementally learnable in the limit from dependency structures. This kind of grammatical inference directly corresponds to the problem of deterministic extraction of a dependency grammar from a dependency treebank.

In this paper, the dispersed and the choice iterations are considered separately. Considered together, they should be related as follows:
$[l_1 \backslash \cdots \backslash l_m \backslash (d_1 | \cdots | d_k)^* \backslash l_{(m+1)} \backslash \cdots \backslash l_n \backslash \beta]^P \preceq [\{d_1^*, \ldots, d_k^*\} \backslash l_1 \backslash \cdots \backslash l_m \backslash l_{(m+1)} \backslash \cdots \backslash l_n \backslash \beta]^P$.
Both constructions are necessary in application CDG. For instance, in English, the type $[(modif|attr)^* \backslash det \backslash obj / claus]$ is admissible for nouns in the syntactic role of a complement, whereas the type $[\{modif^*, attr^*\} \backslash det \backslash obj / claus]$ is not: it allows modifiers and attributes to precede the noun's determinant. At the same time, the type $[pred \backslash circ^* \backslash S / iobj / dobj / \{circ^*\}]$ is admissible for main di-transitive verbs. It allows circumstantials of a verb to occur in any position with respect to its direct and indirect complements. The next step should be to study the learnability of CDG using both primitives.

References

1. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)
2. Béchet, D., Dikovsky, A., Foret, A.: Two models of learning iterated dependencies. In: Proc. of the 15th Conference on Formal Grammar (FG 2010). LNCS, to appear, Copenhagen, Denmark (2010), [online] http://www.angl.huberlin.de/FG10/fg10_list_of_papers
3. Béchet, D., Dikovsky, A., Foret, A., Moreau, E.: On learning discontinuous dependencies from positive data. In: Proc. of the 9th Intern. Conf. "Formal Grammar 2004" (FG 2004). pp. 1–16. Nancy, France (2004)
4. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49, 431–454 (1990)
5. Dekhtyar, M., Dikovsky, A.: Generalized categorial dependency grammars. In: Trakhtenbrot/Festschrift, pp. 230–255. LNCS 4800, Springer (2008)
6. Dekhtyar, M., Dikovsky, A., Karlov, B.: Iterated dependencies and kleene iteration. In: Proc. of the 15th Conference on Formal Grammar (FG 2010). LNCS, to appear, Copenhagen, Denmark (2010), [online] http://www.angl.huberlin.de/FG10/fg10_list_of_papers
7. Dikovsky, A.: Dependencies as categories. In: "Recent Advances in Dependency Grammars". COLING'04 Workshop. pp. 90–97 (2004)
8. Dikovsky, A.: Multimodal categorial dependency grammars. In: Proc. of the 12th Conference on Formal Grammar. pp. 1–12. Dublin, Ireland (2007)
9. Dikovsky, A.: Towards wide coverage categorial dependency grammars. In: Proc. of the ESSLLI'2009 Workshop on Parsing with Categorial Grammars. Book of Abstracts. Bordeaux, France (2009)
10. Gold, E.M.: Language identification in the limit. Information and control 10, 447–474 (1967)
11. Kanazawa, M.: Learnable classes of categorial grammars. Studies in Logic, Language and Information, FoLLI & CSLI (1998)

12. Karlov, B.N.: Normal forms and automata for categorial dependency grammars. Vestnik Tverskogo Gosudarstvennogo Universiteta (Annals of Tver State University). Series: Applied Mathematics 35 (95), 23–43 (2008), (in Russ.)
13. Mel'čuk, I.: Dependency Syntax. SUNY Press, Albany, NY (1988)
14. Shinohara, T.: Inductive inference of monotonic formal systems from positive data. New Generation Computing 8(4), 371–384 (1991)

APPENDIX

Proof of Theorem 2.

Lemma 1. *Let σ be a training sequence for a dispersed K-star revealing CDG G. Then for all i :* $\mathbf{TGE}^{(K)}_{disp}(\sigma[i]) \preceq_{disp} \mathcal{C}^{K}_{disp}(G)$.

Proof. As G is dispersed K-star revealing, G and $\mathcal{C}^{K}_{disp}(G)$ are strongly equivalent. For each DS D in $\Delta(G)$ there is a type assignment in $\mathcal{C}^{K}_{disp}(G)$ to the words appearing in D, which "conforms to" D. More precisely, for a DS D, a word w and vicinity $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$, we choose the smallest type $t_{w,disp}$ among the types assigned to w in $\mathcal{C}^{K}_{disp}(G)$, which yield the same DS D in $\Delta(\mathcal{C}^{K}_{disp}(G))$. $t_{w,disp}$ may be represented as:

$$[\{lf^*_1, \ldots, lf^*_p\} \backslash l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'} / \{rf^*_1, \ldots, rf^*_q\}]^{P'}$$

where P' is a permutation of P.
Let t_w denote the type computed by the algorithm for w in D.
We show that for this type $t_w \preceq_{disp} t_{w,disp}$ (modulo a potentials permutation).
- For each left dispersed iteration d^* in t_w, d occurs at least K times as a left argument in $V(w, D)$. Since $t_{w,disp}$ is minimal for D, d^* must be a member of left dispersed iterations.
- Every dependency name d' occurs less than K times as a left argument of t_w, and less than K times as a left argument in $V(w, D)$. In $t_{w,disp}$, either d' appears the same number of times as a left argument in the same relative positions, or d'^* is a member of left dispersed iterations.
This proves that $t_w \preceq_{disp} t_{w,disp}$.

Lemma 2. *The inference algorithm* $\mathbf{TGE}^{(K)}_{disp}$ *is monotonic, faithful and expansive on every training sequence σ of a dispersed K-star revealing CDG.*

Proof. By definition, the algorithm $\mathbf{TGE}^{(K)}_{disp}$ 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, $\sigma[i] \subseteq \Delta(\mathbf{TGE}^{(K)}_{disp}(\sigma[i]))$. To prove that $\mathbf{TGE}^{(K)}_{disp}$ is `faithful` for $\sigma[i]$ of $\Delta(G) = \Delta(\mathcal{C}^K(G))$, we have to remark that $\mathbf{TGE}^{(K)}_{disp}(\sigma[i]) \preceq_{disp} \mathcal{C}^{K}_{disp}(G)$, using lemma 1.

Lemma 3. *The inference algorithm* $\mathbf{TGE}^{(K)}_{disp}$ *stabilizes on every training sequence σ of a dispersed K-star revealing CDG.*

Proof. G and $\mathcal{C}^{K}_{disp}(G)$ have a finite number of types and of dependency names. $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$ involves a subset of these names, and the same dependency name may be used as argument at most $K-1$ times on the left, and $K-1$ times on the right; there is a finite number of sets of dispersed types; therefore the algorithm must stabilize.

Proof of Theorem 3.

Lemma 4. *Let σ be a training sequence for a choice K-star revealing CDG G. Then for all i : $\mathbf{TGE}_{ch}^{(K)}(\sigma[i]) \preceq_{ch} \mathcal{C}_{ch}^{K}(G)$.*

Proof. As G is choice K-star revealing, G and $\mathcal{C}_{ch}^{K}(G)$ are strongly equivalent. For each DS D in $\Delta(G)$ there is a type assignment in $\mathcal{C}_{ch}^{K}(G)$ to the words appearing in D, which "conforms to" D.

More precisely, for a DS D, a word w and its vicinity

$$V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P,$$

we choose the smallest type $t_{w,ch}$ among the types assigned to w in $\mathcal{C}_{ch}^{K}(G)$, which yield the same DS D in $\Delta(\mathcal{C}_{ch}^{K}(G))$ and to which the rules (1) and (2) in Definition 10 are not applied. $t_{w,disp}$ may be represented as:

$$[l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'}]^{P'}$$

where P' is a permutation of P.

Let t'_w and t_w denote the types computed by the algorithm for w in D at the end of step I and, respectively, of step II.

We show that $t'_w \preceq_{ch} t_{w,ch}$ and $t_w \preceq_{ch} t_{w,ch}$.

- For each left iteration d^* in t'_w, d occurs at least K times as a left argument in $V(w, D)$. Since $t_{w,ch}$ is non-extensible by the rules (1) and (2), there is a minimal length choice iteration in it containing d. If d^* in t'_w corresponds in t_w to a choice iteration $C = (d|...)^*$, then C corresponds to the largest sublist of iterations in t'_w including this d^*. Therefore, it must appear in $t_{w,ch}$ in a choice iteration larger or equal to C.
- Every dependency name d which is a left argument subtype in t'_w or in t_w, occurs less than K times as a left argument in t'_w and t_w and less than K times as a left dependency in $V(w, D)$. Therefore, d appears in $t_{w,ch}$ the same number of times as a left argument subtype or in a minimal length choice iterations including d in the same relative positions.

Thus $t_w \preceq_{ch} t_{w,ch}$.

Lemma 5. *The inference algorithm $\mathbf{TGE}_{ch}^{(K)}$ is monotonic, faithful and expansive on every training sequence σ of a choice K-star revealing CDG.*

Proof. By definition, the algorithm $\mathbf{TGE}_{ch}^{(K)}$ is `monotonic` (the lexicon is always extended). It is `expansive` because for $\sigma[i]$, the types added to the grammar are calculated from and are compatible with the vicinities of the words of $\sigma[i]$. Thus, $\sigma[i] \subseteq \Delta(\mathbf{TGE}_{ch}^{(K)}(\sigma[i]))$. To prove that $\mathbf{TGE}_{ch}^{(K)}$ is `faithful` for $\sigma[i]$ of $\Delta(G) = \Delta(\mathcal{C}_{ch}^{K}(G))$, it suffices to remark that by lemma 4, $\mathbf{TGE}_{ch}^{(K)}(\sigma[i]) \preceq_{ch} \mathcal{C}_{ch}^{K}(G)$.

Lemma 6. *The inference algorithm $\mathbf{TGE}_{ch}^{(K)}$ stabilizes on every training sequence σ of a choice K-star revealing CDG.*

Proof. There is a finite set Σ_{ch} of types and of dependency names in $\mathcal{C}_{ch}^{K}(G)$. The grammar $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$ uses a subset of this set. Let t denote a type assigned by the lexicon of $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$. The size of the multiset of non-repeatable dependency names occurring in t is smaller than $2K \times |\Sigma_{ch}|$ since a dependency name may be used as an argument subtype at most $K-1$ times on the left, and at most $K-1$ times on the right. Any choice iteration has at most $|\Sigma_{ch}|$ elements so the number of choice iteration is finite. No choice iteration may occur next to another choice iteration. Therefore the number of types is finite and the algorithm must stabilize.