On learning discontinuous dependencies from positive data

Denis Béchet, Alexandre Dikovsky, Annie Foret, Erwan Moreau

To cite this version:

Denis Béchet, Alexandre Dikovsky, Annie Foret, Erwan Moreau. On learning discontinuous dependencies from positive data. G. Jäger and P. Monachesi and G. Penn and S. Winter. 9th conference on Formal Grammar (FG 2004), Aug 2004, Nancy, France. pp.1–16, 2004. <hal-00487058>

HAL Id: hal-00487058

https://hal.archives-ouvertes.fr/hal-00487058

Submitted on 27 May 2010

1

On Learning Discontinuous Dependencies from Positive Data

Denis Béchet, Alexander Dikovsky, Annie Foret and Erwan Moreau

Abstract This paper is concerned with learning in the model of Gold the *Categorial Dependency Grammars* (CDG), which express discontinuous (non-projective) dependencies. We show that rigid and k-valued CDG (without optional and iterative types) are learnable from strings. In fact, we prove that the languages of *dependency nets* coding rigid CDGs have finite elasticity, and we show a learning algorithm. As a standard corollary, this result leads to the learnability of rigid or k-valued CDGs (without optional and iterative types) from strings.

1.1 Introduction

Dependency grammars (DGs) are formal grammars assigning *dependency trees* (DT) to generated sentences. A DT is a tree with words as nodes and *dependencies* - i.e. named syntactic relations between words - as arrows. Being very promising from the linguistic point of view (see Mel'cuk (1988)), the DGs have various advantages as formal grammars. Most important is that in terms of dependencies one can naturally encode word order (WO) constraints *independently of syntagmatic relations*. One of the most important WO constraints is that of DT *projectivity*: projections of all words fill continuous [1] intervals. It is widely believed that DTs are a by-product of head selection in

[1] So "discontinuous" corresponds to "non-projective" in dependency terms.

constituents. This is evidently false for non-projective DTs because the projections of the heads are always continuous. But even in the projective case the difference between the two syntactic structures is deep. Even if sometimes the DTs defined from heads are isomorphous to the DTs defined directly, the syntactic functions corresponding to individual dependencies (i.e. the corresponding primitive dependency types) are different. The types of dependencies determine the distributivity of a word in a given lexico-grammatical class in a specific role (as the governor, as a subordinate, as the subject, as a direct object in the post-position, pronominalized or not, etc.). This is why, the number of primitive types is noticeably greater for dependency relations than for syntagmatic relations. The lexical ambiguity is greater for DGs, which is compensated by high self-descriptiveness of individual dependency types. This explains the differences in traditional analyses of the same constructions in syntagmatic terms (which are more or less based on X-bar syntax) and in dependency terms.

Many DGs are projective, i.e. define only projective DTs (cf. Gaifman (1961), Sleator and Temperley (1993), Lombardo and Lesmo (1996)). This property drastically lowers complexity of DGs. As it concerns symbolic learning, positive results are known exclusively for projective DGs: Moreau (2001), Besombes and Marion (2001), Bechet (2003). In this paper we obtain the first result of symbolic learnability of non-projective DGs from positive data. It holds for a reach class of DGs introduced recently in Dikovsky (2004). At the same time, we describe some cases of unlearnability from strings of such grammars with optional or iterative types.

1.2 Categorial Dependency Grammars

1.2.1 Syntactic types

We extend the definition of types in Dikovsky (2004) to *repetitive* and *optional* types as follows. $\mathbf{C}$ will denote a finite set of elementary categories. Elementary categories may be *iterated* and become *optional*. $\mathbf{C}^{*}=_{df}\{C^{*}\mid C\in\mathbf{C}\}$, $\mathbf{C}^{+}=_{df}\{C^{+}\mid C\in\mathbf{C}\}$, $\mathbf{C}^{?}=_{df}\{C^{?}\mid C\in\mathbf{C}\}$, will denote respectively the sets of *iterative*, *repetitive* and *optional* categories. $\mathbf{C}^{\omega}=_{df}\mathbf{C}^{*}\cup\mathbf{C}^{+}\cup\mathbf{C}^{?}$. All categories in $\mathbf{C}^{\omega}$ are *neutral*. Besides them there are *polarized* categories of one of four oriented polarities: left and right positive $\nwarrow, \nearrow$ and left and right negative $\searrow, \swarrow$. For each polarity v, there is the unique "dual" polarity $\breve{v}$: $\breve{\nwarrow} = \swarrow$, $\breve{\swarrow} = \nwarrow$, $\breve{\nearrow} = \searrow$, $\breve{\searrow} = \nearrow$. Intuitively, the positive categories can be seen as valencies of the outgoing distant dependencies of governors, and the negative categories as those of the incoming distant dependencies of subordinate words. So they correspond respectively to the beginnings and the ends of distant dependencies. For instance, the positive valency $(\nwarrow pre{-}UPON{-}obj)$ marks the beginning of the distant dependency $pre{-}UPON{-}obj$ of a transitive verb governing a left-dislocated object headed by the preposition 'UPON', whereas the end of this dependency: UPON is marked by the dual negative valency $(\swarrow pre{-}UPON{-}obj)$ (cf. *upon what dependency theory we rely*).

$\nearrow\mathbf{C}$, $\nwarrow\mathbf{C}$, $\searrow\mathbf{C}$ and $\swarrow\mathbf{C}$ denote the corresponding sets of polarized distant dependency categories. For instance, $\nearrow\mathbf{C} = \{(\nearrow C) \mid C \in \mathbf{C}\}$ is the set of *right positive* categories. $V^+(\mathbf{C}) = \nearrow\mathbf{C} \cup \nwarrow\mathbf{C}$ is the set of positive distant dependency categories, $V^-(\mathbf{C}) = \searrow\mathbf{C} \cup \swarrow\mathbf{C}$ is the set of those negative.

Defining distant dependencies, it is sometimes necessary to express that the subordinate word is the first (last) in the sentence, in the clause, etc., or it immediately precedes (follows) some word. E.g., in French the negative dependency category $\swarrow clit{-}dobj$ of a cliticized direct object must be anchored to the auxiliary verb or to the verb in a non-analytic form. For that we will use specially marked *anchored* negative categories: $Anc(\mathbf{C}) =_{df} \{\#(\alpha) \mid \alpha \in V^-(\mathbf{C})\}$ - our name for negative categories whose position is determined relative to some other category - whereas the negative categories in $V^-(\mathbf{C})$ will be called *loose*.

Definition 1 *The set $Cat(\mathbf{C})$ of* categories *is the least set verifying the conditions:*

1. $\mathbf{C} \cup V^-(\mathbf{C}) \cup Anc(\mathbf{C}) \subset Cat(\mathbf{C})$.
2. *For $C \in Cat(\mathbf{C})$, $A_1 \in (\mathbf{C} \cup \mathbf{C}^\omega \cup Anc(\mathbf{C}) \cup \nwarrow\mathbf{C})$ and $A_2 \in (\mathbf{C} \cup \mathbf{C}^\omega \cup Anc(\mathbf{C}) \cup \nearrow\mathbf{C})$, the categories $[A_1\backslash C]$ and $[C/A_2]$ also belong to $Cat(\mathbf{C})$.*

We suppose that the constructors $\backslash$, $/$ are associative. So every complex category α can be presented in the form:

$$\alpha = [L_k\backslash \ldots L_1\backslash C/R_1 \ldots /R_m].$$

For instance, $[\#(\swarrow clit\_dobj)\backslash subj\backslash S/auxPP]$ is one of possible categories of an auxiliary verb, which defines it as the host word for a cliticized direct object, requires the local subject dependency on its left and, on its right, the local dependency $auxPP$ with a subordinate.

1.2.2 Grammar definition

Definition 2 *A* categorial dependency grammar (CDG) *is a system $G = (W, \mathbf{C}, S, \delta)$, where W is a finite set of words, $\mathbf{C}$ is a finite set of elementary categories containing the selected* root category *S, and δ - called* lexicon *- is a finite substitution on W such that $\delta(a) \subset Cat(\mathbf{C})$ for each word $a \in W$.*

Now we will extend to new types the definitions of the language and DT language generated by a CDG. The language and DT language generated by a CDG are defined using a provability relation $\vdash$ on strings of categories. The core part of this definition are the rules of polarized dependency valencies control. The idea behind these rules is that in order to establish a distant dependency between two words with dual dependency valencies, the negative valency must be *loose*. The anchored negative valencies can serve only to anchor a distant subordinate to a host word. As soon as the correct position of the subordinate is identified, its valency becomes loose and so available to the governor.

In the definition below we suppose that to each occurrence of a category $\Gamma_1 C \Gamma_2$ corresponds a DT D of category C. $r(D)$ denotes the root of D. For space reasons, we present only the rules for left constructors. The rules for right constructors are similar.

Definition 3 *Provability relation* $\vdash$:
Local dependency rule:

L. $\Gamma_1 C[C\backslash\alpha]\Gamma_2 \vdash \Gamma_1\alpha\Gamma_2$. *If C is the category of D_1 and $[C\backslash\alpha]$ is that of D_2, then α becomes the category of the new DT : $D_1 \cup D_2 \cup \{r(D_1) \overset{C}{\longleftarrow} r(D_2)\}$.*

ω-dependency rules:

I. $\Gamma_1 C[C^*\backslash\alpha]\Gamma_2 \vdash \Gamma_1[C^*\backslash\alpha]\Gamma_2$. *If C is the category of D_1 and $[C^*\backslash\alpha]$ is that of D_2, then $[C^*\backslash\alpha]$ in the consequence becomes the category of the new DT : $D_1 \cup D_2 \cup \{r(D_1) \overset{C}{\longleftarrow} r(D_2)\}$.*

R. $\Gamma_1 C[C^+\backslash\alpha]\Gamma_2 \vdash \Gamma_1[C^*\backslash\alpha]\Gamma_2$. *If C is the category of D_1 and $[C^+\backslash\alpha]$ is that of D_2, then $[C^*\backslash\alpha]$ becomes the category of the new DT : $D_1 \cup D_2 \cup \{r(D_1) \overset{C}{\longleftarrow} r(D_2)\}$.*

O. $\Gamma_1 C[C^?\backslash\alpha]\Gamma_2 \vdash \Gamma_1\alpha\Gamma_2$. *If C is the category of D_1 and $[C^?\backslash\alpha]$ is that of D_2, then α in the consequence becomes the category of the new DT : $D_1 \cup D_2 \cup \{r(D_1) \overset{C}{\longleftarrow} r(D_2)\}$.*

Ω. $\Gamma_1[C\backslash\alpha]\Gamma_2 \vdash \Gamma_1\alpha\Gamma_2$ *for all $C \in \mathbf{C}^* \cup \mathbf{C}^?$* [2].

Anchored dependency rule:

A. $\Gamma_1\#(\alpha)[\#(\alpha)\backslash\beta]\Gamma_2 \vdash \Gamma_1\alpha\beta\Gamma_2$, $\#(\alpha) \in Anc(\mathbf{C})$.

Distant dependency rule:

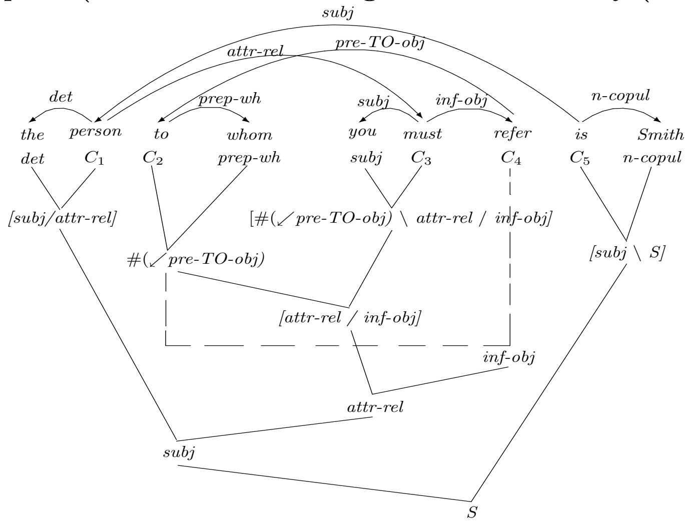
D. $\Gamma_1(\swarrow C)\Gamma_2[(\nwarrow C)\backslash\alpha]\Gamma_3 \vdash \Gamma_1\Gamma_2\alpha\Gamma_3$.
The rule applies if there are no occurrences of subcategories $\swarrow C$, $\#(\swarrow C)$ and $\nwarrow C$ in Γ_2. If $\swarrow C$ is the category of D_1 and $[(\nwarrow C)\backslash\alpha]$ is that of D_2, then α becomes the category of the new DT : $D_1 \cup D_2 \cup \{r(D_1) \overset{C}{\longleftarrow} r(D_2)\}$.

[2]The DTs rest unchanged when no instruction.

$\vdash^*$ *denotes the reflexive-transitive closure of* $\vdash$.

Definition 4 *A DT D is assigned by a CDG $G = (W, \mathbf{C}, S, \delta)$ to a sentence w (denoted $G(D, w)$) if D is defined as DT of category S in a proof $\Gamma \vdash^* S$ for some $\Gamma \in \delta(w)$.*
The DT-language *generated by G is the set of DTs $\Delta(G) = \{D \mid \exists w \in W^+ \; G(D, w)\}$. The* language *generated by G is the set of sentences $L(G) = \{w \in W^+ \mid \exists D \; G(D, w)\}$.*

Example 5 (PP-movement in English from Dikovsky (2004))

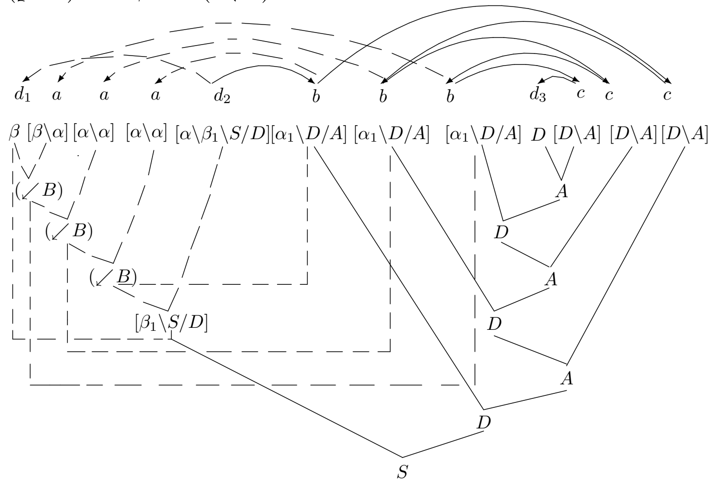


This movement is expressed using the following categories:
$C_1 = [det\backslash subj/attr{-}rel] \in \delta(person)$,
$C_2 = [\#(\swarrow pre{-}TO{-}obj)/prep{-}wh] \in \delta(to)$,
$C_3 = [subj\backslash\#(\swarrow pre{-}TO{-}obj)\backslash attr{-}rel/inf{-}obj] \in \delta(must)$,
$C_4 = [(\nwarrow pre{-}TO{-}obj)\backslash inf{-}obj] \in \delta(refer)$,
$C_5 = [subj\backslash S/n{-}copul] \in \delta(is)$,
$det \in \delta(the)$, $prep{-}wh \in \delta(whom)$, $subj \in \delta(you)$, *and* $n{-}copul \in \delta(Smith)$.

The following example cited from Dikovsky (2004) shows that CDGs can generate non-CF languages and are more expressive than dependency grammars generating projective DTs.

Example 6 *Let $G_0 = (\{a, b, c, d_1, d_2, d_3\}, \mathbf{C}_0, S, \delta_0)$, where δ_0 is defined by:*

$a \mapsto [\beta\backslash\alpha], [\alpha\backslash\alpha],$ $\quad d_1 \mapsto \alpha,$
$b \mapsto [\alpha_1\backslash D/A],$ $\quad d_2 \mapsto [\alpha\backslash\beta_1\backslash S/D],$
$c \mapsto [D\backslash A],$ $\quad d_3 \mapsto D,$
where $\alpha = \#(\swarrow B)$, $\alpha_1 = (\nwarrow B)$, $\beta = \#(\swarrow C)$ *and* $\beta_1 = (\nwarrow C)$.
$L(G_0) = \{d_1a^nd_2b^nd_3c^n \mid n > 0\}$.
A proof of $d_1a^3d_2b^3d_3c^3 \in L(G_0)$, *in which* $\alpha = \#(\swarrow B)$, $\alpha_1 = (\nwarrow B)$, $\beta = \#(\swarrow C)$ *and* $\beta_1 = (\nwarrow C)$*:*

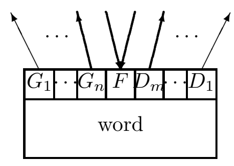

d_1 a a a d_2 b b b d_3 c c c
β $[\beta\backslash\alpha]$ $[\alpha\backslash\alpha]$ $[\alpha\backslash\alpha]$ $[\alpha\backslash\beta_1\backslash S/D]$ $[\alpha_1\backslash D/A]$ $[\alpha_1\backslash D/A]$ $[\alpha_1\backslash D/A]$ D $[D\backslash A]$ $[D\backslash A]$ $[D\backslash A]$
$(\swarrow B)$
$(\swarrow B)$
$(\swarrow B)$
$[\beta_1\backslash S/D]$
A
D
A
D
A
D
S

1.2.3 Language of (untyped) dependency nets

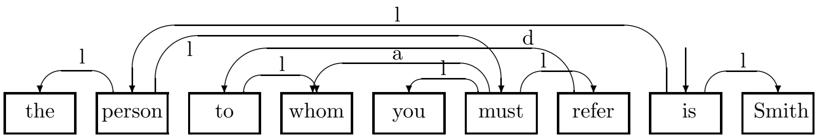
A CDG connects the words of a sentence by oriented edges that correspond to: local dependencies, anchored dependencies, discontinuous dependencies. Thus, the parsing of a sentence can be summarized by a dependency net built from nodes connected by dependencies.

Definition 7 *An untyped node is a list of vertices called* **slots** *associated to a word. A node is an untyped node where each slot corresponds to the elementary categories of one category of the word.*

Here is the node corresponding to $[G_1\backslash\cdots\backslash G_n\backslash F/D_m/\cdots/D_1]$:



Definition 8 *An (untyped) dependency net is a list of (untyped) nodes connected by local (l), anchored (a) and distant (d) dependencies that correspond to a parsing of the sentence.*



One of the slots is not connected and serves as the main conclusion of the dependency net (i.e. the elementary category S). It appears on the figure as an arrow without origin that ends on "is". The dependency tree (DT) corresponding to a dependency net is obtained by erasing anchored dependencies and adding categories on local and distant dependencies (categories appear on nodes in dependency nets and on edges in dependency trees).

Definition 9 *An (untyped) dependency net N is assigned by a CDG G if there exists a parsing of the list of words of N that corresponds to the (untyped) nodes and the dependencies of N.*
The language of dependency nets of G, denoted $\mathcal{NL}(G)$ is the set of dependency nets assigned by G.
The language of untyped dependency nets of G, denoted $\mathcal{UNL}(G)$ is the set of untyped dependency nets assigned by G.

Definition 10 *CDGs that associate at most k nodes to each symbol are called* k-valued. *1-valued grammars are also called* rigid.

1.3 Learnability, finite elasticity and limit points

A class of grammars $\mathcal{G}$ is learnable iff there exists a learning algorithm ϕ from finite sets of words to $\mathcal{G}$ that converges, for any $G \in \mathcal{G}$ and for any growing partial enumeration of $\mathcal{L}(G)$, to a grammar $G' \in \mathcal{G}$ such that $\mathcal{L}(G') = \mathcal{L}(G)$.

Learnability and unlearnability properties have been widely studied from a theoretical point of view. A very useful property for our purpose is the finite elasticity property of a class of languages. This term was first introduced in Wright (1989), Motoki et al. (1991) and, in fact, it implies learnability. A very nice presentation of this notion can be found in Kanazawa (1998).

Definition 11 (Finite Elasticity) *A class $\mathcal{CL}$ of languages has* infinite elasticity *iff $\exists (e_i)_{i\in N}$ an infinite sequence of sentences, $\exists (L_i)_{i\in N}$ an infinite sequence of languages of $\mathcal{CL}$ such that $\forall i \in N \;:\; e_i \notin L_i$ and $\{e_0,\ldots,e_{i-1}\} \subseteq L_i$. A class has* finite elasticity *iff it has not infinite elasticity.*

Theorem 12 [Wright 1989] *A class that is not learnable has infinite elasticity.*

Corollary 13 *A class that has finite elasticity is learnable.*

Finite elasticity is a very nice property because it can be extended from a class to every class obtained by a *finite-valued relation*[3]. We use here a version of the theorem that has been proved in Kanazawa (1998) and is useful for various kinds of languages (strings, structures, nets) that can be described by lists of elements over some alphabets.

Theorem 14 [Kanazawa 1998] *Let $\mathcal{M}$ be a class of languages over Γ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Gamma^*$ be a finite-valued relation. Then the class of languages $\{R^{-1}[M] = \{s \in \Sigma^* \mid \exists u \in M \wedge (s,u) \in R\} \mid M \in \mathcal{M}\}$ has finite elasticity.*

Definition 15 (Limit Points) *A class $\mathcal{CL}$ of languages has* a limit point *iff there exists an infinite sequence $(L_n)_{n\in N}$ of languages in $\mathcal{CL}$ and a language $L \in \mathcal{CL}$ such that: $L_0 \subsetneq L_1 \subsetneq \cdots \subsetneq L_n \subsetneq \cdots$ and $L = \bigcup_{n\in N} L_n$ (L is* a limit point *of $\mathcal{CL}$).*

If the languages of the grammars in a class $\mathcal{G}$ have a limit point then the class $\mathcal{G}$ is *unlearnable.*[4]

1.4 Limit points for CDGs with optional or iterative categories

Limit point constructions.

Definition 16 (G_n, G'_n, G_*, G'_*) *Let S, A, B be three elementary categories. We define by induction:*

$$\begin{array}{ll}
C_0 = S & C'_0 = S \\
C_{n+1} = (C_n/A^?) & C'_{n+1} = (C'_n/A^*)/B^* \\
G_0 = \{a \mapsto A, c \mapsto C_0\} & G'_0 = \{a \mapsto A, b \mapsto B, c \mapsto C'_0\} \\
G_n = \{a \mapsto A, c \mapsto [C_n]\} & G'_n = \{a \mapsto A, b \mapsto B, c \mapsto [C'_n]\} \\
G_* = \{a \mapsto [A/A^?], c \mapsto [S/A^?]\} & G'_* = \{a \mapsto A, b \mapsto A, c \mapsto [S/A^*]\}
\end{array}$$

These constructions yield two limit points as follows.

Theorem 17 *We have:*

$\mathcal{L}(G_n) = \{ca^k \mid k \leq n\}$ *and* $\mathcal{L}(G_*) = c\{a\}^*$

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

Corollary 18 *They establish the non-learnability from strings for the underlying classes of (rigid) grammars : those allowing optional categories ($A^?$) and those allowing iterative categories (A^*).*

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

[4]This implies that the class has infinite elasticity.

Proof for optional categories: Only three rules apply to G_n, G_* in this case : **L**. (local dependency rule), **O**. and **Ω**. (ω-dependency rules). These rules enjoy the subformula property.

- $\mathcal{L}(G_0)$: (1) it clearly contains c (of category S) ; (2) since no rule applies to the elementary categories $\{A, S\}$, it contains only c.
- $\mathcal{L}(G_n)$ $(n > 0)$: we consider ca^k and denote by $\Delta_{n,k}$ its category assigned by G_n. We have :
  $\Delta_{n,0} = [C_n] = [S/\underbrace{A^?/\dots/A^?}_{n}] \vdash^* S$ (by **Ω**. rule, n times)
  $\Delta_{n,k} = [S/\underbrace{A^?/\dots/A^?}_{n}]\underbrace{A\dots A}_{k} \vdash [S/\underbrace{A^?/\dots/A^?}_{n-1}]\underbrace{A\dots A}_{k-1}$ (if $k > 0$, by **O**. rule)
  (1) By induction, for all $k \leq n$, $\Delta_{n,k} \vdash^* S$, that is $ca^k \in \mathcal{L}(G_n)$ when $k \leq n$. (2) Let us consider $w \in \mathcal{L}(G_n)$ of a category Δ. w cannot start with an a (a category A on the left of Δ could not disappear, due to the use of right constructors only); w cannot contain several c, (no cancelation of S is possible since none occurs under a constructor) ; thus $w = ca^k$ for some $k \geq 0$. If $k > n$, then after one step $\Delta_{n,k}$ necessarily leads to $\Delta_{n-1,k-1}$ or $\Delta_{n-1,k}$ as above : by induction starting from the case $\mathcal{L}(G_0)$ $k > n$ is thus not possible. Therefore, $w = ca^k$ with $k \leq n$.
- $\mathcal{L}(G_*)$: (1) it contains ca^k because of $[S/A^?] \vdash S$, $[S/A^?][A/A^?] \vdash [S/A^?]A \vdash S$ and
  $[S/A^?]\dots[A/A^?][A/A^?] \vdash [S/A^?]\dots[A/A^?]A \vdash [S/A^?]\dots A$
  (2) $w \in \mathcal{L}(G_*)$ has exactly one c (at least one to provide S, and no more as explained above); it cannot start with an a (otherwise a type part would rest before S) ; therefore $w = ca^k$. ■

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

- $\mathcal{L}(G'_0)$: (1) it clearly contains c (category S) and (2) only c since no rule applies to $\{A, B, S\}$.
- $\mathcal{L}(G'_n)$ $(n > 0)$. We have $D'_n = C'_{n-1}/A^*$ and $C'_n = D'_n/B^*$.
  (1) For $w \in \{c(b^*a^*)^k \mid k \leq n\}$, we have $w \in \mathcal{L}(G'_n)$ by :
  $[C'_n]B\Delta \vdash [C'_n]\Delta$ and $[D'_n]A\Delta \vdash [D'_n]\Delta$ (by **I**. rule) and $[C'_n] \vdash [D'_n] \vdash [C'_{n-1}]$ (by **Ω**. rule)
  (2) Let $w' \in \mathcal{L}(G'_n)$. As above for G_n, w' cannot start with an a or a b (right constructors only); and w' cannot contain several c (no S under a constructor) ; thus $w' = cw''$, where $w'' \in \{b, a\}^*$.

$w' \in \{c(b^*a^*)^k \mid k \leq n\}$ follows by induction on n and on the length of types Γ words $w \in \{b,a\}^*$ from the following assertion :

(i) if $[C'_n]\Gamma \vdash S$, then $w \in \{(b^*a^*)^k \mid k \leq n\}$ and (ii) if $[D'_{n+1}]\Gamma \vdash S$ then $w \in \{a^*(b^*a^*)^k \mid k \leq n\}$.

For $n = 0$, (i) is clear from $\mathcal{L}(G'_0) = \{c\}$. For (ii), if $\Gamma = B\Gamma'$, we get the first step with the only possibility of $[D'_{n+1}]B\Gamma' \vdash [C'_n]B\Gamma'$. For (ii), if $\Gamma = A\Gamma'$, we have two possibilities $[D'_{n+1}]A\Gamma' \vdash [C'_n]A\Gamma'$ or $[D'_{n+1}]A\Gamma' \vdash [D'_{n+1}]\Gamma'$. For (i), if $\Gamma = A\Gamma'$, we get the first step with the only possibility of $[C'_n]A\Gamma' \vdash [D'_n]A\Gamma'$. For (i), if $\Gamma = B\Gamma'$, we have two possibilities $[C'_n]B\Gamma' \vdash [D'_n]B\Gamma'$ or $[C'_n]B\Gamma' \vdash [C'_n]\Gamma'$. This implies (i), (ii) by induction on n or a shorter type.

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

1.5 Finite elasticity of rigid $\mathcal{UNL}$

This section is concerned with languages of untyped dependency nets rather than grammars of strings. The following theorem is essential because it implies that the corresponding class of rigid CDG (without optional and iterative categories) has finite elasticity and thus is learnable from strings. This result can also be extended to the class of k-valued CDG for every k.

Theorem 19 *Rigid CDGs define a class of languages of untyped dependency nets that has finite elasticity.*

Proof: We use a result of Shinohara Shinohara (1990, 1991) that proves that *formal systems* that have *finite thickness* have also finite elasticity. In Shinohara (1991) this result is applied to *length-bounded elementary formal system with at most k rules* and also to *context sensitive languages* that are definable by at most k rules. Formal systems in Shinohara (1991) describe not only languages of strings but also languages of terms. They can be applied to typed or untyped dependency nets which can be seen as well-bracketed strings (each dependency is associated to an opening and a closing (typed) bracket). For the class of rigid untyped dependency net grammars, a sketch of proof is as follows:

1. **Definition.** A CDG $G_1 = (W_1, \mathbf{C}, S, \delta_1)$ is *included* in a CDG $G_2 = (W_2, \mathbf{C}, S, \delta_2)$ (notation $G_1 \subseteq G_2$) iff $W_1 \subseteq W_2$ and $\forall x \in W_1$, $\delta_1(x) \subseteq \delta_2(x)$.
2. **Definition and lemma.** The mapping $\mathcal{UNL}$ from CDG to untyped dependency net languages is monotonic: if $G_1 \subseteq G_2$ then

$\mathcal{UNL}(G_1) \subseteq \mathcal{UNL}(G_2)$.

3. **Definition.** A grammar G is *reduced with respect to a set X of untyped dependency nets* iff $X \subseteq \mathcal{UNL}(G)$ and for each grammar $G' \subseteq G$, $X \not\subseteq \mathcal{UNL}(G')$. Intuitively, a grammar that is reduced with respect to X covers all the structures of X and has no redundant expressions.
4. **Lemma.** For each finite set $X \subseteq \mathcal{UNL}(G)$, there is a finite set of rigid untyped dependency net languages that correspond to the grammars that are reduced with respect to X. This is the main part of the proof. In fact, if a rigid untyped dependency net grammar $G = (W, \mathbf{C}, S, \delta)$ using types Tp is reduced with respect to X then each word that does not appear in one of the untyped dependency net of X must be associated through δ to the empty set. All other words must be associated to exactly one type of Tp (the grammar is rigid). The left and right numbers of slots are given by the occurrences of the word in the untyped dependency nets and they must be the same for all the occurrences because the language we try to learn corresponds to a *rigid* untyped dependency net grammar. If the sum of the left and right arities of each word in X is bound by m, and if n is the number of words that appear in X, the number of equivalent grammars[5] is bound by the number of partitions of a set of $n \times m$ elements.
5. **Definition.** Monotonicity and the previous property define a system that has *bounded finite thickness.*
6. **Theorem.** Shinohara proves in Shinohara (1991) that a formal system that has bounded finite thickness has finite elasticity.
7. **Corollary.** Rigid untyped dependency net languages have finite elasticity.

A learning algorithm for rigid untyped dependency net grammars

The learning algorithm is based on Buszkowski's original algorithm for rigid (classical) categorial grammars Buszkowski and Penn (1989). Since a rigid grammar G assigns only one type to each word, $X \subseteq \mathcal{UNL}(G)$ implies that all occurrences of a word w appearing in X must be used with the same type $t \in Tp$. Thus G is reduced with respect to X if it simply contains no useless word.

The algorithm ϕ_1 takes an input sequence $\mathbf{s} = N_1, \ldots, N_l$ of untyped dependency nets and returns a CDG that corresponds to the smallest

[5]Equivalent grammars are grammars that are associated to the same language. A sufficient condition is the existence of a bijective relation between the primitive types of both grammars.

rigid untyped dependency net language compatible with $\mathbf{s}$. It returns a failure if the sequence corresponds to no rigid untyped dependency net language [6]. Here is the algorithm ϕ_1 :

1. For each word w, collect in $\mathbf{s}$ its occurrences together with its left and right arities. Fail if a word is used with different arities.
2. With each word w in $\mathbf{s}$, associate $n+m+1$ variables corresponding to its n left argument categories, its m right argument categories and its head type: $X^w_{-n}, \ldots, X^w_{-1}, X^w_0, X^w_1, \ldots, X^w_m$ (X^w_0 corresponding to the head type).
3. Infer from $\mathbf{s}$ the equality constraints for the variables corresponding to the beginnings and the ends of the same dependencies. Respect the orientation: for each word w and each of its argument category, the corresponding dependency must be oriented from w to a word with the corresponding head type. Return a failure if this condition is not fulfilled.
4. Resolve the resulting equality system and associate an elementary category with each variables' equivalence class $\overline{X^w_i}$.
5. Return the CDG $G_{\mathbf{s}}$ such that for each word w in $\mathbf{s}$ with associated variables $X^w_{-n}, \ldots, X^w_{-1}, X^w_0, X^w_1, \ldots, X^w_m$, the lexicon of $G_{\mathbf{s}}$ assigns to w the category $[\overline{Y^w_{-n}}\backslash_{-n} \ldots \backslash_{-2}\overline{Y^w_{-1}}, \backslash_{-1}\overline{Y^w_0}/_1\overline{Y^w_1}/_2 \ldots /_m\overline{Y^w_m}]$, in which:
   •Y^w_i is X^w_i if the corresponding dependency is local and everywhere in $\mathbf{s}$ there is exactly one incoming / outgoing dependency in this slot. Besides this, $\backslash_i$ is $\backslash$ if $i \neq 0$ (respectively, $/_i$ is $/$ in the case of right argument).
   •Otherwise, $Y^w_i = (X^w_i)^?$ if there is at most one outgoing dependency in this slot and at least one net with no outgoing dependency in this slot in $\mathbf{s}$, or $Y^w_i = (X^w_i)^*$ if there is a net with several outgoing dependencies and at least one net with no outgoing dependency in this slot, or finally, $Y^w_i = (X^w_i)^+$ if there is always more than one outgoing dependency and there is at least one net with several outgoing dependencies in this slot. Besides this, $\backslash_i$ is $\backslash$ (respectively, $/_i$ is $/$).
   •Y^w_i is $\#(\swarrow X^w_i)$ (resp. $\#(\searrow X^w_i)$) if the left (respectively, right) slot is the end of an anchored dependency.
   •Y^w_i is $\nwarrow X^w_i$ (respectively, $\nearrow X^w_i$) if the left (respectively, right) slot is the beginning of a distant dependency.
   •Y^w_i is $\swarrow X^w_i$ (respectively, $\searrow X^w_i$) if the slot is the end of a left (respectively, right) loose distant dependency.

[6] I.e. when this sequence is not included in at least one of the rigid untyped dependency net languages.

Theorem 20 *ϕ_1 learns rigid untyped dependency net grammars.*

Proof: ϕ_1 is monotonic: if $\mathbf{s}_1 \subseteq \mathbf{s}_2$ then $\phi_1(\mathbf{s}_2)$ returns a failure or $\phi_1(\mathbf{s}_1)$ and $\phi_1(\mathbf{s}_2)$ succeeds and $\mathcal{UNL}(\phi_1(\mathbf{s}_1)) \subseteq \mathcal{UNL}(\phi_1(\mathbf{s}_2))$. This is a consequence of the fact that the equality system corresponding to $\mathbf{s}_1$ is a subset of the equality system corresponding to $\mathbf{s}_2$. Let G be a rigid CDG and $(N_i)_{i\in N}$ be an infinite sequence of untyped dependency nets that enumerates $\mathcal{UNL}(G)$. For $i \in N$, $\phi_1(N_0, \ldots, N_i)$ does not return a failure because G is rigid so there exists a way to assign a unique type to each word in the untyped dependency nets of $\mathcal{UNL}(G)$. Because ϕ_1 is monotonic, $(G_i = \phi_1(N_0, \ldots, N_i))_{i\in N}$ defines an infinite sequence of growing languages $\mathcal{UNL}(G_0) \subseteq \mathcal{UNL}(G_1) \subseteq \cdots$. The property of finite elasticity implies that this sequence must converge to a language L_∞ that must be (equal or) a superset of $\mathcal{UNL}(G)$ since the sequence enumerates $\mathcal{UNL}(G)$. In fact, for $i \in N$, G verifies the equality system used by ϕ_1 with $N_0, \ldots, N_i$ as input, so $\mathcal{UNL}(\phi_1(N_0, \ldots, N_i)) \subseteq \mathcal{UNL}(G)$. Thus, we also have $L_\infty \subseteq \mathcal{UNL}(G)$ and the sequence of languages converges. Because if G_1 and G_2 are two grammars such that $G_1 \subseteq G_2$ and $\mathcal{UNL}(G_1) = \mathcal{UNL}(G_2)$ that are reduced with respect to $\mathcal{UNL}(G_1) = \mathcal{UNL}(G_2)$ then $G_1 = G_2$: the sequence of grammars converges. ■

1.6 k-valued CDGs without optional or iterative category are learnable from strings

We can define a finite-valued relation between the set of untyped dependency nets that are images of a k-valued CDG through $\mathcal{L}$. The class of rigid untyped dependency net languages having finite elasticity, we can apply Theorem 14 and see that k-valued CDGs without optional or iterative categories are learnable from strings. In fact, we define two relations and use Theorem 14 twice: first time from rigid untyped dependency net languages to rigid string languages, and the second time from rigid string languages to k-valued string languages.

Lemma 21 *String languages of rigid CDG without optional or iterative categories have finite elasticity.*

Proof: Given any string $x = w_1 \ldots w_n$, the maximum number of dependencies drawn over x is $2n - 2$, because there is at most one local or distant dependency and one anchored dependency coming in each word (except the main word). Given any untyped dependency net and any node in this dependency net, all slots are connected to at least one dependency. Therefore there exists a finite number of untyped link dependency nets corresponding to a string x. The class of rigid untyped link dependency nets has finite elasticity, so by theorem 14 the class of rigid CDG string languages without optional or iterative categories also has finite elasticity. ∎

Lemma 22 *k-valued CDGs without optional or iterative category have finite elasticity.*

Proof: This is very standard. The finite-valued relation associates k-valued CDG over W and rigid CDG over $W \times \{1, \ldots, k\}$. A rigid CDG over $W \times \{1, \ldots, k\}$ corresponds to the k-valued CDG where the types associated to $(a, 1), \ldots, (a, k)$ are merged into the same entry for a. ∎

1.7 Conclusion and perspectives

We have proved that a class of rigid and k-valued non-projective DGs has finite elasticity and so is learnable from strings and we have obtained some unlearnability results, as summarized below.

Class	Learnable from strings		Finite elasticity on strings		Finite elasticity on structures		Finite-valued relation
A^*	no	$\not\Rightarrow$	no		yes	$\not\Rightarrow$	no
$A^?$	no	$\not\Rightarrow$	no		yes	$\not\Rightarrow$	no
A^+	yes	$\Leftarrow$	yes	$\Leftarrow$	yes		yes

The positive results may be compared to other learnability results in the same domain in particular in the field of k-valued categorial grammars. For instance, Kanazawa's positive result on classical categorial grammars corresponds to the learnability of the subclass of projective CDGs. On the other hand, some more complex but rather close systems like rigid Lambek calculus or pregroups have been proved to be not learnable from strings Foret and Le Nir (2002a,b). One of possible reasons of this effect might be that - in contrast with the CDGs - in these classes of grammars, reasoning from strings, one can not bound the number of "interactions" (axiom links in terms of proof nets) between two words. This remark may lead to other learnable classes of logical categorial grammars laying between classical categorial grammars and Lambek calculus.

References

Bechet, Denis. 2003. k-valued link grammars are learnable from strings. In *Proccedings of the 8th conference on Formal Grammar (FGVienna).*

Besombes, Jérôme and Jean-Yves Marion. 2001. Identification of reversible dependency tree languages. In L. Popelínský and M. Nepil, eds., *Proceedings of the 3d Workshop on Learning Language in Logic*, pages 11–22. Strasbourg, France.

Buszkowski, Wojciech and Gerald Penn. 1989. Categorial grammars determined from linguistic data by unification. Tech. Rep. TR-89-05, Department of Computer Science, University of Chicago.

Dikovsky, Alexander. 2004. Dependencies as categories. In G.-J. Kruiff and D. Duchier, eds., *Proc. of Workshop "Recent Advances in Dependency Grammars". In conjunction with COLING 2004*. Geneva, Switzerland.

Foret, Annie and Yannick Le Nir. 2002a. Lambek rigid grammars are not learnable from strings. In *COLING'2002, 19th International Conference on Computational Linguistics*. Taipei, Taiwan.

Foret, Annie and Yannick Le Nir. 2002b. On limit points for some variants of rigid lambek grammars. In *ICGI'2002, the 6th International Colloquium on Grammatical Inference*, no. 2484 in Lecture Notes in Artificial Intelligence. Springer-Verlag.

Gaifman, Haïm. 1961. Dependency systems and phrase structure systems. Report p-2315, RAND Corp. Santa Monica (CA). Published in: Information and Control, 1965, v. 8, n 3, pp. 304-337.

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

Lombardo, Vincenzo and Leonardo Lesmo. 1996. An earley-type recognizer for dependency grammar. In *Proc. 16th COLING*, pages 723–728.

Mel'cuk, I. 1988. *Dependency Syntax: Theory and Practive*. State University of New York Press.

Moreau, Erwan. 2001. *Apprentissage des grammaires catgorielles et de dpendances*. Master's thesis, Université de Nantes, France. (in French).

Motoki, Tatsuya, Takeshi Shinohara, and Keith Wright. 1991. 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.: Morgan Kaufmann.

Shinohara, T. 1990. Inductive inference from positive data is powerful. In *The 1990 Workshop on Computational Learning Theory*, pages 97–110. San Mateo, California: Morgan Kaufmann.

Shinohara, T. 1991. Inductive inference of monotonic formal systems from positive data. *New Generation Computing* 8 (4):371–384. Special Issue on Algorithmic Learning Theory for ALT'90.

Sleator, D. and D. Temperley. 1993. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*.

Wright, K. 1989. Identifications of unions of languages drawn from an identifiable class. In *The 1989 Workshop on Computational Learning Theory*, pages 328–333. San Mateo, Calif.: Morgan Kaufmann.