"CDG LAB": a Toolbox for Dependency Grammars and Dependency Treebanks Development

Abstract

We present "CDG LAB", a toolkit for development of dependency grammars and treebanks. It uses the Categorial Dependency Grammars (CDG) as a formal model of dependency grammars. CDG are very expressive. They generate unlimited dependency structures, are analyzed in polynomial time and are conservatively extendable by regular type expressions without loss of parsing efficiency. Due to these features, they are well adapted to definition of large scale grammars. CDG LAB supports the analysis of correctness of treebanks developed in parallel with evolving grammars.

1 Introduction

There are two main technologies of automatic syntactic analysis of natural language: 1. grammatical parsing i.e (symbolic or statistical or mixed) parsing of a hand-crafted grammar belonging to a family of formal grammars disposing of a general purpose parser; 2. data-driven parsing, i.e. parsing with statistical parsers trained over annotated data. Both technologies need a large amount of expensive expert linguistic data. The hand-crafted wide coverage grammars are notoriously expensive and only very few of them had been successfully realized and applied to unrestricted material (cf. (Bouma et al., 2000; Riezler et al., 2002)). Besides this, they are prone to explosion of spurious ambiguity when parsed with general purpose parsers. On the other hand, training of statistical parsers needs voluminous high quality treebanks such as the Penn Treebank (Markus et al., 1993). Training data of this size and quality are in fact as expensive as the hand-crafted grammars and also need a long-term hand work. Even if the results obtained in the statistical parsing during the last fifteen years are very encouraging, their quality and adequacy depends on those of the hand-crafted annotated data. This is a vital issue for dependency grammars which suffer from the shortage of high quality training data. The several existing dependency treebanks (DTB) such as the Prague Dependency Treebank of Czech (Hajicova et al., 1998), the TIGER treebank of German (Brants and Hansen, 2002) or the Russian treebank (Boguslavsky et al., 2000) only partially solve the problem. First of all, they serve for particular languages. Secondly, even for these languages, the DTB use a particular inventory of dependency relations. At the same time, there is no consensus on such inventories. So the DTB are dependent on the choice of underlying syntactic theories, which makes problematic their reuse. The translation technologies (cf. (Hockenmaier and Steedman, 2007)) consisting in acquisition of dependency structures from high quality constituent structure treebanks also do not resolve the problem because, for technical reasons, they often flatten the genuine dependency structures and introduce into them multiple distortions. For all these reasons, there is a need in efficient and inexpensive methods and tools of development of wide coverage grammars and of training corpora.

Below we present "CDG LAB", a toolkit supporting parallel development of wide scope dependency grammars and of DTB. It uses Categorial Dependency Grammars (CDG) as a formal model of dependency grammars.

The CDG, a class of first-order type categorial grammars generating unlimited dependency structures (DS), were introduced in (Dikovsky, 2004). Since then, they were intensively studied (e.g., see (Dekhtyar and Dikovsky, 2004; Béchet et al., 2004; Dekhtyar and Dikovsky, 2008; Dekhtyar et al., 2010; Béchet et al., 2010)). CDG are very expressive. In particular, very simple CDG generate such non-CF languages as $L^{(m)} = \{a_1^n a_2^n ... a_m^n \mid n \geq 1\}$ for all $m > 0$ and $MIX =$

$\{w \in \{a,b,c\}^+ \mid |w|_a = |w|_b = |w|_c\}$. They are equivalent to real time pushdown automata with independent counters (Karlov, 2008). Recently, interesting sufficient conditions of learning CDG in the limit were found (Béchet et al., 2004; Béchet et al., 2010; Béchet et al., 2011).

CDG also have important advantages which make them a convenient and natural means of definition of wide scope dependency grammars. First, they are completely lexicalized, as it is the case of all categorial, and more generally, type logical grammars (Bar-Hillel et al., 1960; Lambek, 1961; Lambek, 1999; Steedman, 1996). Second, the CDG types directly encode DS with repeatable and unlimited discontinuous dependencies (see below). Third, they are parsed in a polynomial time (Dekhtyar and Dikovsky, 2004; Dekhtyar and Dikovsky, 2008). Fourth, an extension of CDG by regular type expressions (`RTE`) specially designed for large scale grammars is defined (Dikovsky, 2009; Dikovsky, 2011) and is implemented in the CDG parser presented below. Moreover, for this extension there is a supported by "CDG LAB" method of incremental bootstrapping of large scale grammars from dependency structures (Dikovsky, 2011).

The plan of this paper is as follows. Section 2 presents the basics of CDG and of their extension by RTE. Then the architecture and the main functionalities of "CDG LAB" are described in Section 3.

2 Categorial Dependency Grammars

CDG define `projective` DS (as in Fig. 1) i.e. DS in which dependencies do not cross, and also `discontinuous` DS, as in Fig. 2, in which they may cross. In these graphs, the nodes correspond to the words of the sentence (their `precedence` order in the sentence is important) and the arcs represent the dependencies: named binary relations on words. Formally, a DS of a sentence x is a linearly ordered cycle-free graph with labelled arcs and the words of x as nodes. We consider connected DS with the root node. When in a DS D there is an arc $w_1 \xrightarrow{d} w_2$, we say that d is a dependency between w_1 and w_2, w_1 is the `governor` and w_1 is `subordinate` to w_1 through d. E.g., in is subordinate to was in Fig. 1 and $donnée$ governs la through $clit-a-obj$ and lui through $clit-3d-obj$.

As all categorial grammars, the CDG are

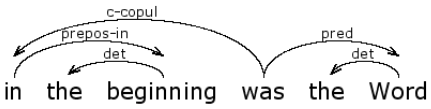
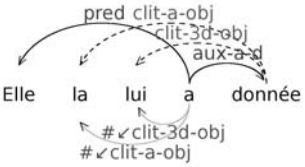


Figure 1: Projective DS



(*fr.* $^*she\ it[fem.]\ to\ him\ has\ given$)

Figure 2: Non-projective DS

completely lexicalized and may be seen as assignments of types to words in a dictionary W. CDG types are expressions of the form

$$t = [l_1 \backslash l_2 \backslash \ldots \backslash H / \ldots / r_2 / r_1]^P.$$

A type assigned to a word $w \in W$ defines its dependencies in a rather straightforward way: its subtypes H, $l_1, l_2, \ldots$, $\ldots, r_2, r_1$ represent the claims for w to be related to other words through projective dependencies and P, called `potential` of t, defines all discontinuous dependencies of w. In particular, the `head` subtype H, claims that w should be subordinate to a word through dependency H. When w should be the root of a DS, $H = S$ (S is a special symbol called `axiom`). The `left` subtypes $l_1, l_2, \ldots$ define the left projective dependencies of w (i.e. the dependencies through which w governs the words occurring in the sentence on its left). The `right` subtypes $\ldots, r_2, r_1$ define the rifght projective dependencies of w. For instance, the projective DS in Fig. 1 is uniquely defined by the type assignment:
$in \mapsto [c-copul/prepos-in]$, $the \mapsto [det]$,
$Word \mapsto [det \backslash pred]$, $beginning \mapsto$
$[det \backslash prepos-in]$, $was \mapsto [c-copul \backslash S / pred]$.

The order of left and right subtypes determines the order of the subordinate words: for two words w_i, w_j preceding to w and subordinate to w through dependencies l_i and l_j respectively, $i < j$ iff w_i is closer to w than w_j (similar for right).

Left and right subtypes may also be `iterated`. The iterated subtypes define `repeatable` dependencies. E.g., $l_i = d^*$ means that w may have on it left $0, 1, 2, \ldots$ occurrences of

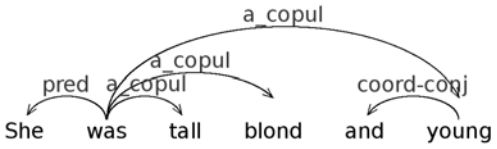


Figure 3: Iterated dependency

words subordinate to it through dependency d. We also use `optional` dependencies $l_i = d?$. Assignment of type $[d?\backslash\alpha]$ is equivalent to assignment of $[d\backslash\alpha]$ and $[\alpha]$. E.g., the DS in Fig. 3 is defined by the type assignment:
$she \mapsto [pred]$, $was \mapsto [pred\backslash S/a\_copul*]$,
$tall$, $blond$, $young \mapsto [coord-conj?\backslash a\_copul]$,
$and \mapsto [coord-conj]$.

The potential P is the concatenation of so called `polarized valencies` of w. The polarized valencies are of four kinds: $\swarrow v$, $\searrow v$ (`negative`) and $\nwarrow v$, $\nearrow v$ `positive`. E.g., if a word w has valency $\swarrow v$, this intuitively means that its governor through dependency v must occur somewhere on the right. Two polarized valencies with the same valency name v and orientation, but with the opposite signs are `dual`. Together they define discontinuous dependency v. The order of polarized valencies in the potential P is irrelevant (so one may choose a standard lexicographic order). For instance, in the DS in Fig. 2, the potential $\nwarrow clit-a-obj \nwarrow clit-3d-obj$ of the participle $donnée$ means that it needs somewhere on its left a word subordinate through dependency $clit-a-obj$ and also another word subordinate through dependency $clit-3d-obj$. At the same time, the accusative case clitic la ($it[fem.]$) has potential $\swarrow clit-a-obj$ and the dative case clitic lui ($to\ him$) has potential $\swarrow clit-3d-obj$. The proper pairing of their dual valencies with those of the participle defines two discontinuous dependencies between the participle and its cliticized complements.

Finally, in order to define for a word subordinate through a discontinuous dependency its adjacency to a `host` word, the CDG types use expressions $\#(\swarrow v)$, $\#(\searrow v)$ called `anchor` subtypes (or just `anchors`), in which v is a valency name. In particular, so that a word w_0 would be a (left position) host for a word w_1, subordinate to some other word through a negative discontinuous dependency v, the anchor $\#(Av)$ (A being one of the four orientations) should be the head subtype of the type of w_1 and at the same time a left subtype of the type of w_0. Similar for right position. E.g., the DS in Fig. 2 is defined by the following assignment of types to words:
$elle \mapsto [pred]$,
$la \mapsto [\#(\swarrow clit-a-obj)]^{\swarrow clit-a-obj}$,
$lui \mapsto [\#(\swarrow clit-3d-obj)]^{\swarrow clit-3d-obj}$,
$donnée \mapsto [aux-a-d]^{\nwarrow clit-a-obj \nwarrow clit-3d-obj}$,
$a \mapsto [\#(\swarrow clit\text{–}3d\text{–}obj)\backslash\#(\swarrow clit\text{–}a\text{–}obj)\backslash pred \backslash S/aux-a-d]$.
Due to the anchor subtypes $\#(\swarrow clit-3d-obj)$, $\#(\swarrow clit\text{–}a\text{–}obj)$ in the type of the auxiliary verb a (has), it serves as the host verb for both clitics and also defines their precedence order.

Derivability of DS in CDG is formalized through a type calculus which is defined in all papers cited above. Here, in the place of a formal definition we explain its intuitive meaning and illustrate it by an example.

A projective dependency d from a word w_2 to a word w_1 is constructed when a phrase x_1 with the head w_1 has type d^{P_1}, a phrase x_2 with the head w_2 has type $[d\backslash\beta]^{P_2}$ and the two phrases become adjacent forming the composite phrase $x = x_1x_2$. In this case x has the head w_2 and obtains the derived type $[\beta]^{P_1P_2}$. Similar for right subtypes. In the special case where d is an anchor, the corresponding anchor dependency is constructed. All DS we show are constructed by a CDG parser. It diplays the anchor dependencies below the sentence for a better readability.

A discontinuous dependency v from a word w_2 to a word w_1 in a phrase $x = x_1w_1x_2w_2x_3$ is constructed when in the potential P of a derived type t^P of x there are two dual valencies named v: $P = \gamma_1 \swarrow v\ \gamma \nwarrow v\gamma_1$ such that $\swarrow v$ is a negative valency in the potential of a type originally assigned to w_1, $\nwarrow v$ is a positive valency in the potential of a type originally assigned to w_2 and the part γ separating these valencies has neither occurrences of $\swarrow v$, nor of $\nwarrow v$. Put simply, this rule called **FA** (`first available`) says that w_1 is the closest to w_2 word in x having the dual valency v. Similar for right dependencies.

A CDG G is defined by its dictionary W and its `lexicon` λ, an assignment of finite sets of types to words in W. G defines a DS D of a sentence $x = w_1 \dots w_n$ and x is generated by G (denoted $D \in \Delta(G)$ and $x \in L(G)$) if it is possible to assign through λ a type t_i to everry word w_i so that the obtained type string $t_1 \dots t_n$ were reducible to

the axiom S. $L(G)$ is the `language` and $\Delta(G)$ is the `structure language` generated by G.

Let us see how DS in Fig. 2 may be defined using the type assignment shown above.

First, we may eliminate the left anchor subtype $\#(\swarrow clit\text{–}3d\text{–}obj)$ in the type of the auxiliary verb a using the type of the clitic lui. As a result, we generate the anchor dependency $\#(\swarrow clit-3d-obj)$ from a to lui and the derived type of the string $lui\ a$ becomes $[\#(\swarrow clit{-}a{-}obj)\backslash pred\backslash S/aux{-}a{-}d]^{\swarrow clit\text{–}3d\text{–}obj}$. In this type, we may eliminate the anchor subtype $\#(\swarrow clit\text{–}a\text{–}obj)$ using the type of the clitic la. This will generate the anchor dependency $\#(\swarrow clit{-}a{-}obj)$ from a to la. The derived type of the sequence $la\ lui\ a$ is $[pred\backslash S/aux{-}a{-}d]^{\swarrow clit\text{–}a\text{–}obj \swarrow clit\text{–}3d\text{–}obj}$. Now we may eliminate the left subtype $pred$ of the derived type using the type of the subject $elle$. This generates the projective dependency from a to $elle$ and $elle\ la\ lui\ a$ obtains the derived type $[S/aux-a-d]^{\swarrow clit\text{–}a\text{–}obj \swarrow clit\text{–}3d\text{–}obj}$. Then using the type of the participle $donnée$, we may eliminate the right subtype of this type. This generates the projective dependency $aux-a-d$ from a to $donnée$ and assigns to the sentence the derived type $[S]^{\swarrow clit\text{–}a\text{–}obj \swarrow clit\text{–}3d\text{–}obj \nwarrow clit\text{–}a\text{–}obj \nwarrow clit\text{–}3d\text{–}obj}$. Application of the rule **FA** to the dual valencies $\swarrow clit-3d-obj$ and $\nwarrow clit-3d-obj$ generates the discontinuous dependency $clit-3d-obj$ from $donnée$ to lui and derives for the sentence the type $[S]^{\swarrow clit\text{–}a\text{–}obj \nwarrow clit\text{–}a\text{–}obj}$. Finally, applying this rule to the dual valencies $\swarrow clit-a-obj$ $\nwarrow clit-a-obj$ we generate the DS in Fig. 2 because the discontinuous dependency $clit{-}a{-}obj$ from $donnée$ to la is generated and the derived type is S.

Extended CDG. CDG are a theoretical model not adapted to wide coverage grammars. The main problem with wide coverage is the excessive sharing of subtypes in types. For lexicons running to hundreds of thousands of lexical units it results in a combinatorial explosion of spurious ambiguity and in a strong parsing slowdown. Wide coverage grammars face many hard problems, e.g. those of compound lexical entries including complex numbers, compound terms, proper names, etc. and also that of flexible precedence order. (Dikovsky, 2009) proposes an extension of CDG adapted to wide coverage grammars.

The extended CDG use classes of words in the place of words and use restricted regular expressions defining sets of types in the place of types. I.e., the dictionary W is covered by classes: $W = \bigcup_{i\in I} C_i$ and the lexicon λ assigns sets of regular expressions to classes. At that:

- all words in a class C share the types defined by the expressions assigned to C,
- every word has all types of the classes to which it belongs.

The extended CDG use flat (i.e. bounded depth) `regular type expressions` (`RTE`) we describe below. In these expressions, C, C_i are dependency names or anchors, B is a `primitive` type, i.e. a dependency name, or an anchor or an iterated or optional type, and H is a choice.

Choice: $(C_1|\dots|C_k).\ (C) = C$.
Optional choice: $(C_1|\dots|C_k)?.\ (C)? = C?$.
Iteration: $(C_1|\dots|C_k)^*.\ (C)^* = C^*$.
Distributed subtypes expressing flexible order.
Left: $[\{\alpha_1, B, \alpha_2\}\backslash\alpha\backslash H/\beta]^P$.
Right: $[\alpha\backslash H/\beta/\{\alpha_1, B, \alpha_2\}]^P$.
Two-way: $\{\alpha_1, B, \alpha_2\}[\alpha\backslash H/\beta]^P$.

As the original CDG, the extended CDG are formalized by a calculus wich has special rules for every kind of RTE (see (Dikovsky, 2009; Béchet et al., 2010; Béchet et al., 2011)). Below we informally explain the intuitive meaning of the operators used in the RTE.

The choice unites several alternative types into one. For instance, assigning the type $[(C_1|C_2)\backslash\beta]^P$ is equivalent to assign two types $[C_1\backslash\beta]^P$ and $[C_2\backslash\beta]^P$. An element (a word or a phrase) to which is assigned RTE $[(C_1|\dots|C_k)^*\backslash\beta]^P$ allows any sequence of elements of the alternative types $C_1, \dots, C_k$ immediately on its left. On the contrary, assignment to an element of the type $[\{\alpha_1, B, \alpha_2\}\backslash\alpha\backslash H/\beta]^P$ means that an element of type B must be present in some left position. E.g. the assignments $w_0 \mapsto [\{d\}\backslash b\backslash a\backslash S]$, $w_1 \mapsto [a]$, $w_2 \mapsto [b]$, $w_3 \mapsto [d]$ define DS of sentences: $w_3w_1w_2w_0$, $w_1w_3w_2w_0$, $w_1w_2w_3w_0$ in which $w_0 \xrightarrow{d} w_3$, $w_0 \xrightarrow{a} w_1$ and $w_0 \xrightarrow{b} w_2$. The right distributed RTE is similar. The two-way distributed RTE claims that an element of type B were found in some left or right position.

The RTE and the classes do not extend the expressive power of CDG. At the same time, they dramatically reduce the grammar size. Of course, unfolding of an extended CDG may exponentially blow up its size. However, due to the extended type calculus they can be parsed directly, without unfolding. In fact, the polynomial time parsing al-

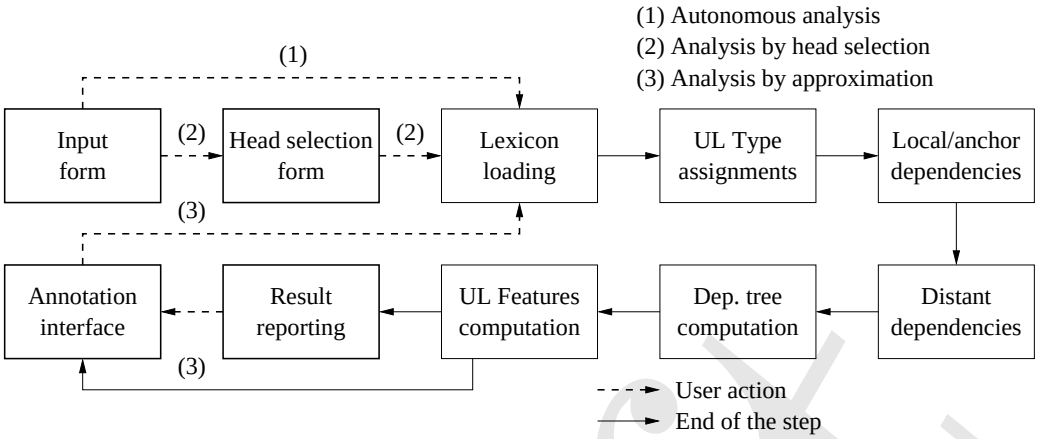


Figure 4: Architecture of the CDG parser

gorithm of (Dekhtyar and Dikovsky, 2008) can be adapted to the extended CDG.

3 CDG Lab

CDG Lab is a kit of tools supporting parsing with extended CDG, development and maintainance of dependency treebanks (`DTB`) and development and test of large scale extended CDG. The core element of CDG Lab is the parser of the extended CDG implemented in Steel Bank Common Lisp. Recently was issued its version 3.1 (below we will call this parser `Parser-3.1`).

All input and output data of Parser-3.1 are XML-structures. It may analyse sentences, text corpora and DTB either with an extended CDG integrated with an external morpho-syntactic dictionary (`lexical base grammar`) or only with the internal grammar lexicon (`text grammar`). For instance, for French is developed a large scale extended CDG (Dikovsky, 2011). Its version 3.1 (called below "French Text CDG") is integrated with the open MS-dictionary of French Lefff 3.0 (Sagot, 2010) containing 536,375 lexical units (`LU`). In CDG Lab, Lefff is kept in object-relational database PostgreSQL. A correspondence between the classes of the French Text CDG and the categories of Lefff is implemented through several hundreds of SQL queries. The integral grammar is called below "French LB CDG".

Depending on the User command, Parser-3.1 may be used in one of four modes:

- Analysis by head selection
- Analysis by approximations
- DS analysis

Figure 5: Query Form

- Autonomous analysis

Fig. 4 shows a scheme of functioning of Parser-3.1 in these modes. Sentences are introduced through the input form (see Fig. 5). Through this form, the User may set various parameters, e.g. the maximal parsing time, the maximal number of DS to return, a graphical representation of DS, a language register (corresponding to specific choices of discontinuous dependencies common to official documents or to scientific or literary prose, to periodicals or to the spoken language), etc. The input sentence is lexically analysed. Composite forms are decomposed into separate tokens, as in the case of $l'homme$ (the man), which is segmented into three tokens: $l, '$ and $homme$, for which are found in the lexicon all possible lemmas. In this

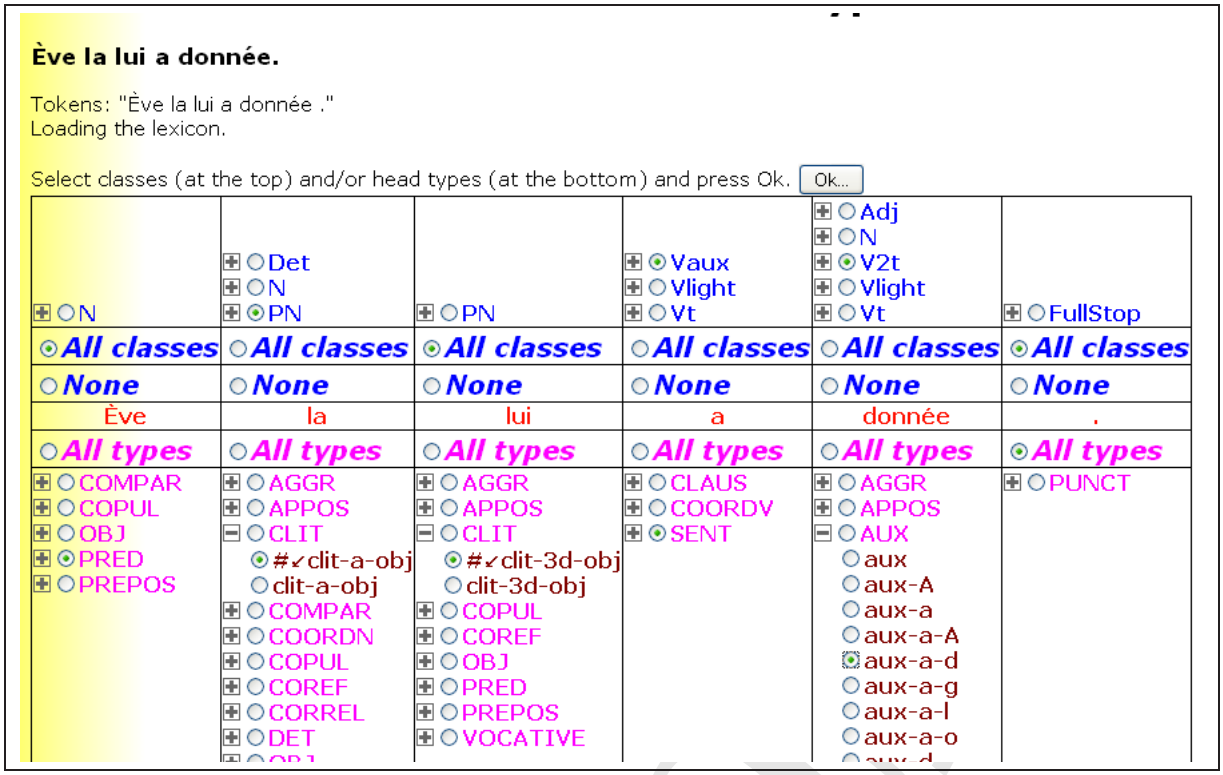


Figure 6: Selection Form

example, the association is ambiguous: *l'* may be a clitic or a determiner. Are also detected all possible variants of composite LU (in particular, complex numbers and names recognized through regular expressions, multi-word LU, such as *à la* (of the kind), *à travers* (through) etc.) and unknown terms are identified.

The transitions to and from head selection form are followed only in the mode of Analysis by head selection. Functioning in the mode of Analysis by approximations is iterative. It goes round result reporting and passes from annotation interface directly to lexicon loading. Other transitions are common for all modes. So we comment them in the head selection mode.

Analysis by Head Selection. In this mode is proposed `selection form` (see Fig. 6), in which the User may select the proper composite LU (if and when several possibilities are detected) and for every LU, to select one of possible classes and one of possible dependency relation groups (or of their elements). The selected dependency is nothing but the head subtype of the possible types of the LU. The latter selection is in fact decisive. It corresponds to the strong constraint that the LU is subordinate through the selected dependency (or limits the choice of such dependencies to the elements of the selected group).

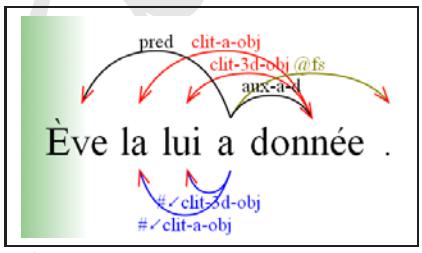


Figure 7: Resulting DS

This selection drastically limits the search-space. As it concerns the Parser-3.1 and the French LB CDG, it reduces the number of possible analyses by two-three orders of magnitude, i.e. in the place of a thousand of possible DS only about ten are found, most often differing between them in positions of repeatable dependencies (such as *modif* (`modifier`), *attr* (`attribute`) or *circ* (`circumstantial`). E.g., in this example the genuine DS will be immediately found due to this selection (see Fig. 7).

Then is created the sentence's workspace (`WS`), an XML-structure representing the subgrammar corresponding to all detected LU to which are affected the classes, the types and the features values compatible with the pre-selection. After this

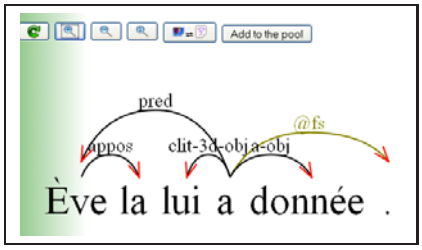


Figure 8: Incorrect DS

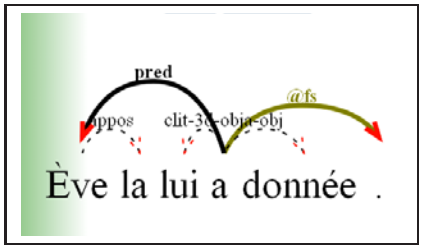


Figure 9: First annotated DS

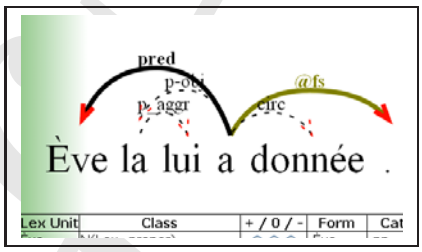
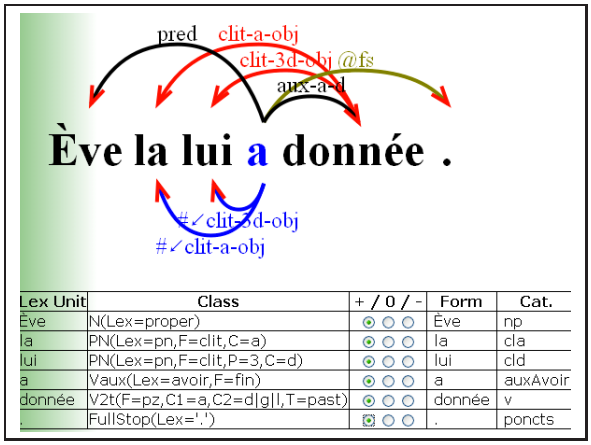


Figure 10: Next approximation

follow the steps common to all modes. First are computed and registered in the triangular matrix all possible projective (and anchor) dependencies. This computation is done by an CKY-like algorithm adapted to extended CDG. In the resulting matrix (more precisely, in the submatrix in which the axiom S can be proved) are independently computed and registered all possible pairings of dual valencies providing discontinuous dependencies. It should be noted that the pairing principle may be chosen for a CDG and for a particular discontinuous dependency. By default, this is the **FA** rule. But in rare exceptional situations, such as that of unlimited cross-serial dependencies in subordinate clauses in Dutch, a different rule **FC** (`first cross`) defined in (Dikovsky, 2007) may be used. This independence of computations of projective and discontinuous dependencies is founded on the fundamental `projection independence property` of CDG proved for the rule **FA** in (Dekhtyar and Dikovsky, 2004; Dekhtyar and Dikovsky, 2008) and for rule **FC** in (Dikovsky, 2007). Till the end of this step the parsing algorithm is polynomial. The resulting triangular matrix is in fact a packed chart from which it is possible to enumerate all possible DS of the sentence. Given that the number of these DS may be exponential with respect to the size of the matrix, the next step is exponential in space in the worst case. In this step, the DS are generated from the matrix in a certain order and the feature values are assigned to LU in every generated DS. Finally, the parser generates the HTML report page, which includes various useful statistics. An XML structure representation of every DS including all necessary information, in particular the CDG classes and the feature values is also generated and saved to be used by other programs.

Analysis by Approximations. This important mode represents another User-guided strategy of parsing. It allows to find the needed DS starting from any obtained DS by consecutive approximations computed from User's annotations in the DS. There are three possible annotations of dependency relations: `positive`, `negative` and `neutral`. The positively annotated dependencies are those adequate. They will be kept during the whole sequence of approximations (if not discarded). The neutrally annotated dependencies are kept till they are compatible with the positively annotated ones. The negatively annotated dependencies are to be eliminated from the DS. When used in this mode, the Parser computes for every DS the total number of positively annotated dependencies and that of negatively annotated dependencies. The obtained DS are sorted first by negative annotations' weight (the less negative annotations the better) then by the positive annotations' weight (the more positive annotations the better).

Suppose, that the approximations start from the (partially incorrect) DS of the sentence $\grave{E}ve\ la\ lui\ a\ donn\acute{e}e$ ($Eve\ it[fem.]\ to\ him\ has\ given$) shown in Fig. 8. There are only two correct dependencies in this DS: the predicative one: $pred$ and the punctuation dependency $@fs$. We annotate both positively (this annotation being displayed by boldface arcs). The other three dependencies $appos$, $clit-3d-obj$ and $a-obj$ are erroneous. We annotate them as negative (which is displayed by broken arcs). So we obtain the first annotated DS shown in Fig. 9.



Lex Unit	Class	+ / 0 / -	Form	Cat.
Ève	N(Lex=proper)	⊙ ○ ○	Ève	np
la	PN(Lex=pn,F=clit,C=a)	⊙ ○ ○	la	cla
lui	PN(Lex=pn,F=clit,P=3,C=d)	⊙ ○ ○	lui	cld
a	Vaux(Lex=avoir,F=fin)	⊙ ○ ○	a	auxAvoir
donnée	V2t(F=pz,C1=a,C2=d|g|l,T=past)	⊙ ○ ○	donnée	v
	FullStop(Lex='.')	⊙ ○ ○	.	poncts

Figure 11: Final approximation

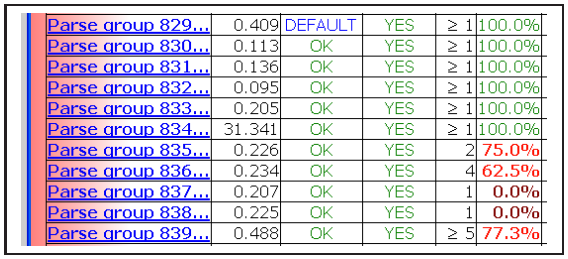
From this annotated DS, the Parser computes the next approximation shown in Fig. 10, which is also incorrect. It has the same two correct dependencies and three other incorrect: $p{-}obj$, $p{-}aggr$ and $circ$. We annotate the three as negative, as it is shown in Fig. 10.

From this annotated DS, the parser finally computes the right one shown in Fig. 11. Not only this final approximation is correct, but it is also annotated as such. This difference is very important for the other mode of use of Parser-3.1, that of DS analysis.

DS, DTB and Grammar Analysis. In fact, Parser-3.1 considers every DS as annotated. The case where there are no annotations is considered as that with weight 0. Moreover, not only the dependencies, but also the LU may be annotated. The LU may have only two annotations: `positive` and `neutral`. Annotating a LU w positively is equivalent to positively annotate all dependencies in the sub-structure with the root w. This is seen in Fig 11, were the positive annotation of the root (displayed in a contrasting color) implies the positive annotations of dependencies (displayed in boldface). More than that, the class and the feature values assigned to every LU in DS may also be annotated as `positive`, `negative` or `neutral`. In the fragment of the class/feature table shown in Fig 11, it is seen that not only the dependencies, but also the class/feature assignments for its LU are all annotated as positive. So this analysis is 100% correct. It is using this integral annotation weight, that Parser-3.1 evaluates the DS. Now, for two DS of the same sentence it is possible to measure the difference of their weights. This simple measure turns out to be an efficient means of analysis of DTB and of the CDG used while their development. Every sentence processed by Parser-3.1 using French LB CDG obtains its `status`. The status includes the analysis result ('NO', when there are no parses, 'YES' otherwise) and for a parsable sentence, also the maximal number of returned DS and the difference (in percents) of annotation weights (of dependencies and of LU) between the best obtained DS and the one present in the DTB (if any) before this sentence processing. When the grammar is updated, the DS of sentences in the DTB become potentially irrelevant. For this case, Parser-3.1 has a special function of `re-parsing` of a DTB, which computes the difference between the DS before and after update, comparing their statuses. The User may choose between keeping or not the same head subtypes while re-parsing. Using this function, one may easily find all sentences to be revised.

Another way round, this test applies to the grammar itself. The French Text CDG was created using Structural Bootstrapping Method (Dikovsky, 2011), a method specific to the extended CDG and consisting in an `incremental` transformation of DS of a sample of sentences σ into an extended CDG $G(\sigma)$ generating these sentences. The incrementality is interpreted in the strong sense: $\Delta(G(\sigma)) \subseteq \Delta(G(\sigma \cup \{s\}))$ for every new sentence s. The bootstrapping of French Text CDG was basically incremental in this sense, except three important revisions which were not. Taking in mind the size and the complexity of this grammar (it consists of more than 2800 RTE distributed between 185 lexicon classes, has 84 projective and 20 discontinuous dependencies), it was a very hard task to find all sentences in the sample wrongly analysed using the updated grammar. Indeed, to find them, it was necessary to look through thousands of DS of hundreds of sentences in order to find linguistically adequate DS (the simple existence of a generated DS is of course not sufficient). The situation has completely changed after the implementation of Parser-3.1. Indeed, now all sentences in the sample are initially annotated. The procedure of re-parsing of the sample finds all inconsistencies in several minutes.

In Fig. 12 we show a fragment of the table representing the results of re-parsing applied to a DTB. In this table:



Parse group 829...	0.409	DEFAULT	YES	≥ 1	100.0%
Parse group 830...	0.113	OK	YES	≥ 1	100.0%
Parse group 831...	0.136	OK	YES	≥ 1	100.0%
Parse group 832...	0.095	OK	YES	≥ 1	100.0%
Parse group 833...	0.205	OK	YES	≥ 1	100.0%
Parse group 834...	31.341	OK	YES	≥ 1	100.0%
Parse group 835...	0.226	OK	YES	2	75.0%
Parse group 836...	0.234	OK	YES	4	62.5%
Parse group 837...	0.207	OK	YES	1	0.0%
Parse group 838...	0.225	OK	YES	1	0.0%
Parse group 839...	0.488	OK	YES	≥ 5	77.3%

Figure 12: Re-parse results

- the first column is the reference to the DS of a sentence,
- the second column shows the (folded) characteristics of parsing complexity,
- in the third column, 'OK' means that all LU of the sentence are present in the grammar lexicon and 'DEFAULT' means that there is at least one LU absent in the lexicon and replaced by the default unit,
- in the fourth column, 'YES' means that the re-parsing was successful in the sense that a successful analysis was found and 'NO' means the contrary,
- in the fifth column is given the maximal number of DS requested for re-parsing ($\geq k$ means that there are more than the k requested DS),
- the sixth column shows the part (in percents) of annotation status coincidence of the DS before and after the update (so it is 0% for new sentences).

Autonomous Analysis. This is the mode of non-User-guided analysis. Parser-3.1 may be used as a general purpose parser for the extended CDG. In CDG Lab there is a possibility to upload one's own grammar or to introduce it through the Sandbox. For itself, Parser-3.1 is rather efficient. Even when used with the French LB CDG, it is capable to analyse half a thousand of sentences of various complexity in about 10 minutes. The problem is that it generates the DS not in the order of their adequacy. With ambiguous CDG, such as French LB, it generates hundreds of spurious structures per sentence. So for very long and complex sentences, it is practically impossible to know whether a reasonable DS was computed. This is why we consider Parser-3.1 as a tool of development of DTB using head subtype selection, approximations and re-parsing.

DTB Development. The annotation based development of DTB in CDG Lab leads to a notable change in the point of view on the quality of treebanks. It is now the grammar, implementing a set of linguistic subjective expert knowledge, which will serve as the "golden standard". As to the DTB, they should all be correct with respect to the grammar and should be tested for correctness after every non-incremental grammar update. By definition, the incremental grammar updates preserve correctness of DS.

Besides the means based on DS annotation, CDG Lab has rather standard means for creation and updates of DTB and for search of DS by projective and discontinuous dependency names and by LU in the sentences.

Grammar Development. Besides the described above general purpose means supporting non-incremental grammar updates, CDG Lab has some means specific for CDG of French integrated with Lefff 3.0. In particular, it has several functions for completion of the lexicon of these CDG. Basically, there are two problems: the first is to automatically complete the lexicon by all forms of a missing word (this concerns mainly the verbs), the second is to compute the argument frame of a missing word from that of a present word. For the former problem, CDG Lab has several functions based on updates of the lexicon of French Text CDG. The latter problem concerns the deverbals. The work on completions of this kind is in progress.

4 Conclusion

CDG Lab combines several means of incremental parallel development of wide scope dependency grammars and of dependency treebanks provably correct with respect to the grammars. These means were successfully tested while development of a wide scope categorial dependency grammar of French and of an experimental dependency treebank. Some of these means are general purpose. E.g., the annotation weight difference test applies to any kind of structural incremental development based on expert annotations. Some other, such as head sybtype selection and consecutive approximations, may be used with many classes of dependency grammars and implemented in tabular dependency grammar parsers. Some means are specific to the Parser-3.1 and to the French CDG integrated with Lefff. Several important means of this Toolkit are still under construction, but even this experimental version has proved its high efficiency.

References

Y. Bar-Hillel, H. Gaifman, and E. Shamir. 1960. On categorial and phrase structure grammars. *Bull. Res. Council Israel*, 9F:1–16.

Denis Béchet, Alexander Dikovsky, Annie Foret, and Erwan Moreau. 2004. On learning discontinuous dependencies from positive data. In *Proc. of the 9th Intern. Conf. "Formal Grammar 2004" (FG 2004)*, pages 1–16, Nancy, France.

Denis Béchet, Alexander Dikovsky, and Annie Foret. 2010. Two models of learning iterated dependencies. In *Proc. of the 15th Conference on Formal Grammar (FG 2010)*, LNCS, to appear, Copenhagen, Denmark. [online] `http://www.angl.hu-berlin.de/FG10/fg10\_list\_of\_papers`.

Denis Béchet, Alexander Dikovsky, and Annie Foret. 2011. On dispersed and choice iteration in incrementally learnable dependency types. In *Proc. of the 6th Int. Conf. "Logical Aspects of Computational Linguistics" (LACL'2011)*. Accepted paper.

I. Boguslavsky, S. Grigorieva, N. Grigoriev, L. Kreidlin, and N. Frid. 2000. Dependency treebank for russian: Concept, tools, types of information. In *Proc. of the 18th Int. Conf. on Comput. Ling. (COLING'2000)*.

G. Bouma, G. van Noord, and R. Malouf. 2000. Alpino: Wide-coverage computational analysis of dutch. In *Proc. of the Conf. Computational Linguistics in the Netherlands*, pages 45–59.

S. Brants and S. Hansen. 2002. Developments in the tiger annotation scheme and their realization in the corpus. In *Proc. of the Second Int. Conf. on Language Ressources & Evaluation (LREC'02)*.

Michael Dekhtyar and Alexander Dikovsky. 2004. Categorial dependency grammars. In *Proc. of Intern. Conf. on Categorial Grammars*, pages 76–91, Montpellier.

Michael Dekhtyar and Alexander Dikovsky. 2008. Generalized categorial dependency grammars. In *Trakhtenbrot/Festschrift*, LNCS 4800, pages 230–255. Springer.

Michael Dekhtyar, Alexander Dikovsky, and Boris Karlov. 2010. Iterated dependencies and kleene iteration. In *Proc. of the 15th Conference on Formal Grammar (FG 2010)*, LNCS, to appear, Copenhagen, Denmark. [online] `http://www.angl.hu-berlin.de/FG10/fg10\_list\_of\_papers`.

Alexander Dikovsky. 2004. Dependencies as categories. In *"Recent Advances in Dependency Grammars"*. *COLING'04 Workshop*, pages 90–97.

Alexander Dikovsky. 2007. Multimodal categorial dependency grammars. In *Proc. of the 12th Conference on Formal Grammar*, pages 1–12, Dublin, Ireland.

Alexander Dikovsky. 2009. Towards wide coverage categorial dependency grammars. In *Proc. of the ESSLLI'2009 Workshop on Parsing with Categorial Grammars. Book of Abstracts*, Bordeaux, France.

Alexander Dikovsky. 2011. Categorial Dependency Grammars: from Theory to Large Scale Grammars. Submitted paper.

E. Hajicova, J. Panevova, and P. Sgall. 1998. Language ressources need annotations to make them really reusable. In *Proc. of First Int. Conf. on Language Ressources & Evaluation (LREC)*, pages 713–718.

J. Hockenmaier and M. Steedman. 2007. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Boris N. Karlov. 2008. Normal forms and automata for categorial dependency grammars. *Vestnik Tverskogo Gosudarstvennogo Universiteta (Annals of Tver State University). Series: Applied Mathematics*, 35 (95):23–43. (in Russ.).

J. Lambek. 1961. On the calculus of syntactic types. In Roman Jakobson, editor, *Structure of languages and its mathematical aspects*, pages 166–178. American Mathematical Society, Providence RI.

J. Lambek. 1999. Type grammars revisited. In Alain Lecomte, François Lamarche, and Guy Perrier, editors, *Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers*, volume 1582. Springer-Verlag.

M. Markus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330.

S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. 2002. Parsing the wall stret journal using a lexical-functional grammar and discriminative estimation techniques. In *Proc. of the 40th Ann Conf. of the ACL*, pages 271–278.

B. Sagot. 2010. The lefff, a freely available and large-coverage morphological and syntactic lexicon for french. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*.

Mark Steedman. 1996. *Surface structure and interpretation*. MIT Press, Cambridge,Massachusetts.