Filtering Left Dislocation Chains in Parsing Categorial Grammar

Crit Cremers*
Maarten Hijzelendoorn*

Abstract

This paper reports on a way to reduce the complexity of the process of left dislocation (re)construction for categorial grammar in the case of lexically assigned gaps, as an additional restriction on the complexity arising from lexical polymorphism in general. Specifying extraction sites lexically has the advantage that the combinatory explosion can be contained in the preparsing track by a specialized constraint on the expansion of sequences of categories. This constraint is called the *Left Dislocation Chain Filter* and is implemented by a Finite State Transducer. It is shown that the Filter can reduce the number of full string assignments under consideration prior to parsing with an average of one half to one order of magnitude, depending on the nature of the sentence.

1 Parsing Left Dislocation

Left dislocation is a very common, almost universal phenomenon in natural languages. It establishes the relation between an element at the left periphery of a clause and a particular, lexically open position in its right context. The leftward nature of dislocation is explained in Kayne (1994). The most prominent of these relations are invoked by so called wh-elements at the leftmost edge of questions and relative clauses. If a language has left dislocation, however, many other constituents can occur in a left dislocated position. Here are some examples from Dutch; the distinguished position in the right context is marked by t.

(1) Mimi vroeg zich af [wie] Jan dacht dat *t* zou gaan winnen
Mimi wondered (herself) who Jan thought that would go win
'Mimi wondered who Jan thought would win'

(2) [De film [die] ik heb *t* gezien] kan jij niet *t* gezien hebben
The movie that I have seen can you not seen have
'You cannot have seen the movie I have seen'

*Department of General Linguistics, Leiden University

In (2) we see two dislocations, one (a wh-induced type) within the boundaries of the other (a fronting of a 'normal' constituent). We will call the structure relating the left dislocated constituent and the empty position a left dislocation chain. We will refer to the two positions involved as the landing and the launch site, respectively, pursuing the dislocation metaphor. The lexical material at the landing site is also often referred to as the 'filler', and the other position as the 'gap'.

The properties and parameters of left dislocation chains are surely among the major topics of syntactic research in our days. It has become abundantly clear that there are major restrictions on these chains - represented by weak and strong islands for extraction - although there are many positions that may be part of one.

Natural language grammar is supposed to establish left dislocation chains, as the interpretation of left dislocated constituents is determined by their chain. Parsing natural language grammars therefore involves the (re)construction of left dislocation. The nature of this construction will correlate to the grammar's specification of dislocation, but the problem of parsing dislocation is quite general. At least the launch site of a chain is not explicitly marked - leaving aside prosodic information - and often, the lexical material in the landing site is not recognizable as being dislocated during lexical look-up. Consequently, a parser must actively compute the left dislocation chain in accordance with the grammatical nature of the relation; Van de Koot (1990) e.g. has an insightful treatment of the computational problem of left dislocation for Marcus-parsing. The need for the computation is evident: if a parser deduces that a certain noun phrase may be left dislocated, it has to check the possible noun phrase positions in the right context for being the launch site. This kind of parsing problem does only occur if the left-peripheral constituent of a clause is selected by an entity to its right. Adverbial adjuncts, for example, do not necessarily belong to this class of chain inducing entities: they are modifying other constituents, rather than being selected by them. One might, however, find good reasons, along with Bouma and Van Noord (1994), to consider adjuncts as arguments after all. In that case, their occurrence at the left periphery of a clause must be seen as dislocation and gives rise to chains that have to be computed as well. The present study is not biased with respect to this alternative.

In categorial grammar, one can think of at least two ways of establishing left dislocation chains. For categorial grammars exhibiting a full hypothetical logic, the 'gap' is constructed by withdrawing a hypothetical occurrence of a category and thus, by introducing a complex category; this approach is pursued e.g. in Hepple (1990) and Morrill (1994; ch.8).

Alternatively, gaps can be introduced as elements of lexical categories which are related to the filler by some form of 'gap threading' (cf. Pereira and Shieber 1987). This is the approach taken in DELILAH, a grammar/parser system for Dutch developed by the authors. In the latter system, gaps are introduced as such in lexical assignments of categories to words, alternating with assignments providing lexical arguments (see below). In any case, the parser of a categorial grammar has to check several filler-gap combinations in a trial-and-error mode in order to determine a left dislocation chain. This paper reports on a way to reduce the complexity of the process of left dislocation (re)construction for categorial grammar in the case of lexically assigned gaps, as an additional restriction on the complexity arising from lexical polymorphism in general.

2 Lexical Ambiguity and Parsing Categorial Grammar

Lexical ambiguity is known to be a major threat to efficient parsing of natural language. Barton, Berwick and Ristad (1987: ch.3) demonstrate that the combination of simple agreement and lexical ambiguity makes natural language parsing NP-complete, i.e. a standard problem in the class of computationally intractable problems. Evidently, agreement can be taken to go proxy for all kinds of mutual dependencies between phrases in a sentence. Dependencies like agreement are at the heart of natural language, and adequate grammars must account for it. As a consequence, an adequate grammar of natural language can hardly be parsed efficiently if it has to allow for lexical ambiguity. In this vein, Johnson (1991) proves that even the Tomita algorithm for generalized LR parsing shows exponential complexity when applied to lexically ambiguous grammars. In this proof, the exponent is determined by the size of the grammar. These results confirm the observation in Gazdar and Mellish (1989: p.169) that “The cubic worst-case time efficiency problem for natural language parsers (...) is completely dwarfed in practice by a much more serious problem, that of pervasive natural language ambiguity”.

Unfortunately, this statement is fully applicable to categorial grammar. Categories can be seen as combinatorial agendas. Every category imposes a set of requirements on its context. A string of categories represents a well-formed sentence only if these requirements turn out to converge. A certain degree of lexical ambiguity – or rather: polymorphism *per* lexical atom – seems inevitable. In categorial grammar, for example, differences in subcategorization (a verb may select an infinitival as well as a tensed complement), word order (a finite verb may have its complements to the left or to the right) or double functionality (a word might be a preposition or a particle) must lead, in some stage of the parsing process, to branching possibilities and an increase of search space. As an example of a lexical item which introduces many combinatorial agendas, consider the case of Dutch *willen*, ‘to want’. The lexicon of Dutch has to specify for *willen* at least the following different categories, which are casted here in a neutral format:

(3)

(i)	vp/vp	(infinitival form with vp-complement)
(ii)	vp/s_sub	(infinitival form with tensed sentential complement)
(iii)	s/vp/np	(finite form with vp-complement for verb-second and verb-first main sentences)
(iv)	s/s_sub/np	(as (iii), but with sentential complement)
(v)	s_vn\np/vp	(as (iii), but for verb-final sentences)
(vi)	s_vn\np/s_sub	(as (v), but with tensed complement)

This list is not necessarily complete. For example, if one needs to distinguish declarativity from other sentential modes like questioning, more sentential categories may be added. The variety that arises, cannot be handled by type changing rules. As a matter of fact, none of the types listed in (3) can be deduced from another type in the list by canonical type changing rules, though every finite type is a regular and predictable expansion of one of the basic infinitival types.

In general, we can describe the problem of lexical ambiguity for categorial grammar as follows. Let G_{NL} be a categorial grammar for a language NL, and L a lexicon with initial assignment $A(w_i)$ of nonterminals to the words w_i of NL. For example, A(*willen*) contains at least the categories given in (3). Let $S = w_1 \dots w_n$ be a sentence over L. Deciding whether S is in NL amounts to searching some sequence C $= c_1 \dots c_n$, with $c_i \in A(w_i)$ such that C is derivable under G_{NL}. The solution to the problem may require checking the derivability of many such Cs. Basically, for a certain S the number of sequences the derivability of which must be checked is $\Pi_1^n |A(w_i)|$, the Cartesian product over w_i, which is exponentially dependent on n. $\Pi_1^n |A(w_i)|$ defines the search space for parsing S. The search space should not be defined by spurious ambiguity, however. It makes sense to require for each c in the lexical assignment of some word w that there is a sentence in NL containing w, which can only be derived under G_{NL} if c is in $A(w)$. In this vein, we require every initial assignment of a category to a word to be necessary with respect to G_{NL}.
Note that the search space is not influenced by the algorithm of the theorem prover itself. One can, however, represent lexical ambiguity as a complex category by means of additional type constructors, as is suggested e.g. by Morrill (1994: ch.6). To an ambiguous term a conjunction of categories and/or a disjunction of arguments is assigned. At each deduction step in which this complex category is involved, the theorem prover has to choose which of the coordinated types is activated. No hope is offered, though, as to the efficiency of this procedure.

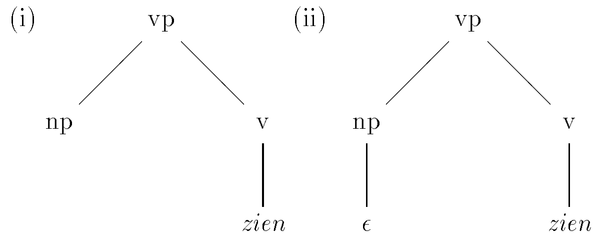
This approach to ambiguity shows, however, that there is, to a certain degree, a trade-off between lexical ambiguity and properties of the grammar. In particular, a grammar may assign nonterminals to certain lexically assigned nonterminals, by having monadic rules or theorems of the type $c \rightarrow c'$. One can think of the famous lifting rule $x \rightarrow y\backslash(y/x)$ and the Geach rule $x/y \rightarrow (x/z)/(y/z)$. Their presence, however, causes a serious problem for theorem proving itself, as they induce spurious ambiguity (see e.g. Wittenburg 1986, König 1990, and Hepple 1990 for analyses and performance-oriented remedies of this phenomenon). In these cases, the search space for parsing S is partially constructed by the grammar itself. Again, the question whether G_{NL} derives S, induces an explosion of queries as to whether G_{NL} derives a certain C.

Because of the logic of categorial grammar, theorem proving is a genuine model for parsing these grammars (Hepple 1990). Propositions, which have to be checked for being a theorem, are sequences of categories. If a sentence gives rise to more than one sequence, all these sequences - i.e. all these combinations of combinatorial agendas – have to be checked. Of course, the theorem prover itself can be trimmed in several fashions, pertaining on the complexity of the proof-finding process. If, however, the categorial grammar involved is of a (mildly) context-sensitive nature (as are the Combinatory Categorial Grammar of Steedman (1990) - see Joshi *et. al.* (1991) – and the grammar in the present DELILAH system – see below) the theorem prover is confronted with the fact that context-sensitive recognition is PSPACE-complete (Hopcroft and Ullman 1979; ch.13). But the major burden on the parsing process is imposed by the multiplicity of potential theorems, independently of the properties of the deductive system. Below we will discuss how the combinatorial explosion of hypotheses can be controlled in DELILAH.

Here we will concentrate on the processing of a particular source of lexical polymorphism: the possibility that in a locally-defined argument structure one argument may be missing as a result of left dislocation, also known as movement to [Spec, CP]. The categorial lexicon of Dutch may specify as a category of the verb *zien* 'to see' not only $vp\backslash np$ to indicate that it is meant to head a configuration with an np to its left, but also $vp\backslash np^\wedge gap$ to indicate that that object may not be present.

If we consider the category $vp\backslash np$ as introducing a local tree of type (4)(i), to which a local tree with a root np can be adjoined at the node marked as such, the category $vp\backslash np^\wedge gap$ must be seen as the introduction of the local tree (4)(ii) in which this node is barred by emptiness. Structurally, the trees are the same, though they will be treated differently by the rules of grammar (see below).

(4)



Gapped categories are invariantly specified as left arguments, since dislocation is leftward. Therefore, the gapped counterparts to the categories pp/np and vp/vp are $pp\backslash np^\wedge gap$ and $vp\backslash vp^\wedge gap$, respectively.

The additional specification of *zien* as a verb that may lack an adjacent object is predictable – the objects of transitive verbs are generally available for extraction – but not trivial. Not all np arguments occurring in some lexically assigned category are candidates for extraction; the np in a possessive determiner (np's n), for example, is not. Moreover, it is useful to store the information that, within a certain complex category, at most one argument is available for extraction. The verb *geven* 'to give' has one category specifying a bitransitive infinitive, but also two additional categories specifying the separate extractability of each argument of that infinitive – only one argument can be dislocated, of course. Consequently, in infinitival position *geven* has three lexical options, instead of one; the number of finite lexical categories associated with *geven* multiplies accordingly.

The specification of extractability, then, may increase the number of categories assigned to a particular lexical item and contribute to a search space explosion for parsing. As noted above, one could choose not to specify launching sites lexically, but to compute the possibilities while parsing. For Lambek's categorial grammar, there is the option of hypothetical reasoning, as in Morrill (1994), and in other frameworks one can implement other forms of gap threading, as in Stabler (1992). But deriving all possible extraction sites is not necessarily more efficient than specifying them. In fact, we will show that specifying extraction sites lexically has the advantage that the combinatory explosion can be contained in the pre-parsing track by a specialized constraint on the expansion of sequences of categories. Moreover, the lexical approach complies with the argument by Johnson and Kay (1994) that gap hypotheses in a derivation must be licensed by lexical items in order to assure termination of the parsing process.

To a large extent, the art of parsing is finding secure means to restrict the number of possible assignments. Given the exponential function $\Pi_1^n |A(w_i)|$, efficiency requires serious pruning of the search space. Optimally, the means to achieve this are anchored in the grammar that is to be applied. Resource-sensitive categorial grammar offers some options for pre-checking assignments. The parsing system DELILAH incorporates an instance of a bracket-free, mildly context-sensitive categorial grammar. It deals with various forms of discontinuity, like free coordination, verb raising and long distance dependencies.

The grammar basically consists of one cancelling operation, generalized composition (cf. Steedman 1990, Joshi. *et al.* 1991), operating on two triples of the form Head \ LeftArgumentList / RightArgumentList.

Heads are basic types; they may be cancelled while the argument lists of their categories merge with the argument lists of the category that provokes the cancelling. The formalism and its applications are discussed in Cremers (1993; ch.2). A related, but slightly less expressive formalism is defined in Milward (1995) as AB Categorial Grammar with Associativity (AACG). DELILAH's grammar may be taken to have at least mildly context-sensitive power, as it extends the concept of generalized composition, which Joshi *et al.* (1991) prove to be in that class.

In (5) we present the general scheme for a left-to-right cancelling. (Of course, right-to-left cancelling also exists.)

(5) *DELILAH's Grammar Format*

If a string with category

PrimaryHead \ LeftList / [SecondaryHead^Operator | RestRightList]

occurs to the left of a string with category

SecondaryHead \ OtherLeftList / OtherRightList,

combine these strings – under some restrictions triggered by Operator with respect to the content of the argument lists – to a string of category

PrimaryHead \ NewLeftList / NewRightList,

where NewLeftList and NewRightList stem from appending the two left lists and the two right lists, respectively, in either one of two possible orders, which encode either continuity or discontinuity.

In a rule notation we get (6), where $append_o$(L1,L2) is some append-operation triggered by some operator $^\wedge o$, yielding a list whenever it is defined for L1 and L2, and non-executable otherwise. In the latter case, the two categories to the left of the arrow cannot reduce to one by cancellation of Sec.

(6) Prim\LList1/[Sec^o|RList1]   Sec\LList2/RList2 →
      Prim\append$_o$(LList1,LList2)/append$_o$(RList1,RList2)

A string is considered to be a well-formed sentence, iff the lexical hypotheses (categorial agenda's) can be reduced by recursive applications of (6) to one category *s\[]/[]*. This grammar strictly preserves directionality (cf. Steedman 1990), only cancels elementary types, and does not use hypothetical reasoning. Instead, because the composition rule takes into account the full internal structure of both the primary and the secondary category, non-peripheral extraction can be treated without specialized operators like the up and down arrows of Moortgat (1988). Dislocation and other forms of word order variation can be handled by composition alone. (7) shows two options of adjunction to a verb; both options are available if the operator $^\wedge o$ is defined for the relevant internal structure of the secondary category at the stage of cancelling *vp*.

(7) (i) vp\[]/[vp^o]   np\[]/[]   vp\[np^_]/[]   ⇒
      vp\[]/[vp^o]   vp\[]/[]   ⇒
      vp\[]/[]
   (ii) np\[]/[]   vp\[]/[vp^o]   vp\[np^_]/[]   ⇒
      np\[]/[]   vp\[np^_]/[]   ⇒
      vp\[]/[]

The combination of directionality and the fact that the grammar only cancels basic types - as does Milward's (1995) AACG - has interesting consequences for parsing. For a given prefix $P(i)$ of assignments to the first i words of a sentence, we can decide whether it makes sense to add to it a certain lexical category of the $i+1$th word, that is, we can decide whether $P(i)$ plus that certain category can be the prefix $P(i+1)$ of a sequence of categories which may be parsed succesfully. If not, that particular extension of $P(i)$ is rejected, and with it all the (virtual) sequences in the set with cardinality $\Pi_1^n |A(w_i)|$ that have it as a prefix. Technically, these prefixes are best considered to be paths of a tree which is built tier-by-tier during lexical look-up: a directed acyclic graph with categories as vertices and edges between neighbours in a sequence. At the extreme vertex of every path, information is accumulated on the type pattern of the path. A new category is added as a vertex connected to a path and extending that path only if its agenda is not incompatible with the information at the 'preceding' vertex. When the category is added as a new vertex, it stores the updated information on the extended path. If no category of a certain word is compatible with an existing path, the path is pruned; all remaining (active) paths are of equal length. The main instrument here is an operationalization of Count Invariance (Van Benthem 1986, Moortgat 1988, König 1990): the property that sequences of complex types can be derived only if they exhibit a certain balance of primitive types. This is the central strategy used in DELILAH to restrict the search space, even in the context of coordination (see Cremers and Hijzelendoorn 1997 and Cremers 1989). Thus, the search space is considerably limited *on-line*. As a matter of fact, while building the tree, the pruning rate exceeds the growth factor of the search space (cf. Cremers and Hijzelendoorn 1997). As $\Pi_1^n |A(w_i)|$ explodes with n, the proportion *#active-paths-of-length-n* $/\Pi_1^n |A(w_i)|$ decreases exponentially. The decrease of this proportion justifies the construction of the pruned tree of viable prefixes-up-to-n. For a numerical illustration of this effect, consider the parsing of the sentence

(8) Wie zegt de man, die ik wilde laten werken, dat hem gedwongen heeft mij met de poppen te laten spelen?
Who says the man that I wanted let work that him forced has me with the dolls to let play?
'Who does the man, I wanted to work, say that forced him to let me play with the dolls?'

Under DELILAH's present lexicon, the search space $\Pi_1^n |A(w_i)|$ for this sentence consisting of 20 words contains 822,528,000 possible assignments (paths). Building and pruning the tree with a remainder of 109 paths takes 6,120 ms cpu time (excluding time for garbage collecting, stack shifting, or in system calls) on a Silicon Graphics Indigo R4000 workstation; this includes the time taken by the special pruning algorithm for gapped categories to be discussed below. Parsing these 109 non-rejected sequences takes 360 ms, say 3 ms for each. Since there is no principal difference between paths which were pruned and paths which survived pruning, we can deduce that parsing the whole tree of sequences would have taken 822,528,000 times 3 ms is 2,467,584,000 ms. This is about 411,000 times as much as DELILAH needed to construct-and-prune the search space.

It is worth noting that the pruning does not put any claim on the parsing strategy that is applied. The dynamic application of Count Invariance only selects what has to be parsed, not how this task is performed. In particular, since not all the remaining paths will differ from each other at any node, one can think of a form of chart parsing to exploit the remaining hypotheses space. On the other hand, little is known about efficient parsing of context-sensitive grammars.

Under the grammar sketched above, left dislocation is solved syntactically by bringing together the left dislocated constituent and a unique gap. The gap is transported leftwards by generalized composition: in fact, a gap is 'inherited' by every category of every string that contains the word introducing the gap. Finally, the gap is the only left argument of the constituent to the right of the dislocated phrase. In the example derivation (9) with a left dislocated noun phrase, X and Z are sequences of categories. Only a few stages of the derivation are made explicit; they are connected by subsequent application of generalized composition (5–6).

(9) np\[]/[] s\[]/[...] X y\[...]/[vp^o] vp\[...,np^gap]/[...] Z ⇒
np\[]/[] s\[]/[...] X y\[...,np^gap]/[] ⇒
np\[]/[] s\[]/[x^o] x\[np^gap]/[] ⇒
np\[]/[] s\[np^gap]/[] ⇒
s\[]/[]

Except for special circumstances, which we will not consider here (e.g. parasitic gaps; for a treatment see Morrill 1994), gaps and dislocated constituents are related one-to-one.

3 Filtering Left Dislocation Chains

Although Count Invariance (exploiting the resource sensitivity of certain categorial systems) is operationalized in DELILAH for on-line reduction of search space prior to proper parsing, it cannot discriminate between configurations (possible prefixes of sequences of lexical assignments) other than by the mere occurrence of basic types. Gaps have to be marked in the lexicon for the category they represent. If a gap is to be bound to a preposed *np*, it has to be marked for this binding as a gap or variable of that particular type. Consequently, the introduction of gaps may increase the number of categories of a certain lexical item that looks for a particular type to the left; gaps are always bound by left dislocated constituents (cf. Kayne 1994). As for the application of Count Invariance, there is no difference between a category *vp\[np]/[]* and its gapped relative *vp\[np^gap]/[]*: they expand the same tree in terms of number, labels, and structure of nodes (cf. (4)). If Count Invariance allows for the attachment of one of them to a prefix $P(i)$, it also allows for the attachment of the other. In this case, the number of sequences to be checked for grammaticality is doubled by the mere presence of a gapped category in the lexicon for the $i+1$th word in the sentence. In general, Count Invariance is underspecified with respect to the grammatical extension of prefixes, as it is applied prior to parsing.

The main contribution of gapped categories to the extension of the search space is caused by the gap's location being undetermined. In a given string of words, there are many possible candidates that introduce a gapped category, though in the final parse only one of the candidates will turn out to be the real gapped category. Sequences like the following are hardly candidates for succesful parsing if only one of the categories introduces a finite domain.

(10) np ... s\[np^gap]/_ ... vp\[np^gap]/_ ...

It makes sense, then, to look for additional means to prevent spurious accumulation of gapped categories in a sequence, just to accommodate the gap's indeterminacy *a priori*.

For every prefix of a sequence of lexical categories, we use a Finite State Transducer (FST) in DELILAH to keep track of the maximal number of gaps that must be made available in the suffix of that sequence. This FST performs its tasks in the very same pre-parsing track in which Count Invariance is used dynamically to prune the search space. The timing for the pre-parsing procedure for (8) given above, included the operations of the FST.

The general idea is as follows. Every prefix $P(i)$ of a sequence of assignments S(n) is associated deterministically with a state of an FST and a number generated by that state. Recall that these prefixes can be seen as paths of a tree under construction. The number that the FST provides, is associated with the extreme vertex of that path. This number indicates how many gap 'slots' are available for a suffix to that prefix. The states of the FST represent the relevant features of a particular $P(i)$. The $i+1$th word may introduce a category for which the FST is defined in that state, or not. If the category is in the domain of the state of the FST with which $P(i)$ is associated, it will force the FST to move. This move may involve adding 1 or subtracting 1 from the counter, or even a reset of the counter. Addition moves are forced by types which introduce a syntactic domain from which extraction is permitted. Finite verbs, for example, force addition moves. Subtraction moves are forced by categories containing gaps, or – in certain states – by categories that indicate closure of extraction domains. No subtraction move can be made if the counter is zero. In that case, the FST fails, and the category will not be added to $P(i)$ to form $P(i+1)$, since $P(i+1)$ cannot be associated with a state of the FST. The category may be added to some other prefix $P(i)$, though. Also, another category from the $i+1$th word's lexical assignment may be added to $P(i)$ to form a $P(i+1)$.

Thus, the moves of the FST are triggered by types occurring in a category of the $i+1$th input word for a given prefix of assignments $P(i)$. In its actual form, DELILAH activates the FST only if the category that is a hypothetical extension of a prefix of a sequence of assignments is either 1. headed by a main sentential type, 2. has a finite sentential type as an argument, 3. contains a gapped type, or 4. is headed by prepositions. The first two triggers will be evident: they are the ones that indicate finite domains, the main extraction fields. They force the FST to make an addition move. The third one is also evident: it is the one that forces a subtraction move. PPs need a special treatment in order to account for certain consequences of pied-piping. They have to make available an additional gap option, apart from the option that allows for their own dislocation which is introduced in a standard way.

Here is a description of the FST and some comments. The triggering types are:

head_main_s (hms)	the type that is the head of a finite verb in main sentences; e.g. s in $s\backslash[np^\wedge gap]/[vp]$
head_embedded_s (hes)	the type that is the head of a finite verb in embedded sentences; e.g. s_vn in $s\_vn\backslash[np\ np]/[\ ]$
head_pp (pp)	the head of a prepositional category; e.g. pp in $pp\backslash[\ ]/[np]$

right_argument_embedded_s (raes)	the type that is a right hand side argument of a complementizer, announcing the start of an embedded sentence; e.g. *s_vn* in $np\backslash[np]/[s\_vn]$
gap (gap)	the left hand argument of any category that is marked for gap; e.g. $pp^\wedge gap$ in $vp\backslash[np\ pp^\wedge gap]/[\ ]$

Of each type, the head is processed first, then its left searching arguments, and finally its right searching arguments. The transition scheme is given in (11) as a relation between a triple <state, type, number> and a pair <state, number>. Some very idiosyncratical transitions are left out for transparency reasons.

(11) *Left Dislocation Chain Filter*

initialization: <a, 0>	
<a, hms, X> ⇒ <e, 1>	finite main verb resets gap options
<a, hes, X> ⇒ <a, X>	finite embedded verb does not change options; necessary for coordinated structures
<a, gap, X> ⇒ <a, X−1 > if X > 0	consumption of gap option
<a, raes, X> ⇒ <b, X+1 >	introducing a new domain for dislocation
<b, raes, X> ⇒ <b, X+1 >	idem
<b, hes, X> ⇒ <b, X>	as before
<b, gap, X> ⇒ <b, X−1 > if X > 0	as before
<c, hms, X> ⇒ <e, 1>	all options not yet consumed are lost; number of options reset to one
<c, hes, X> ⇒ <d, X>	just a state transition
<c, raes, X> ⇒ <b, X+1 >	as before
<c, gap, X> ⇒ <c, X−1 > if X > 0	consumption of gap option
<d, hes, 0> ⇒ <a, 0>	re-initialization; the end of a domain is introduced; all options used
<d, hes, X> ⇒ <c, X−1 >	end of domain; gap option for that domain not used; number of options decreases
<d, raes, X> ⇒ <b, X>	idem, but also start of new domain: options not changed
<d, hms, X> ⇒ <e, 1>	as before
<d, gap, X> ⇒ <c, X−1 > if X > 0	consumption
<e, raes, X> ⇒ <b, X+1 >	introduces new domain with additional option
<e, gap, 1> ⇒ <a, 0>	consumption
<S, pp, X> ⇒ <[S], X+1 >	for every state S a special state [S] is needed for pied-piping phenomena, introducing an additional gap option
<[S], gap, X> ⇒ <S, X−1 >	consumption in ‘pied-piping’ state
<[S], ϵ, X> ⇒ <S, X−1 >	empty move otherwise; the extra option is cancelled

A category C will be added to a prefix of assignments $P(i)$ in state S with number N if the FST is defined in S for the types in C. A simple example is given in (12).

(12) Ik wil elke man een boek geven
 I want every man a book give
 'I want to give every man a book'

Let the string of categories in (13) be a prefix of assignments (one of possibly many more) to the first six words, up to *boek*, i.e. $P(6)$.

(13) np s\[np^gap]/[vp] np\[]/[n] n np\[]/[n] n

That prefix, just one among others that may have survived thus far, will be associated with state <a,0>. This can be seen as follows. The FST is entered in state <a,0>. The first type, *np*, is not a triggering type; the FST does not move. The head of the second type, *s* is an hms; the FST moves to state <e,1>, creating an option for a gap to be consumed later. The left searching category $np^{\wedge}gap$ is a gap, which brings the FST back to state <a,0>. The right searcher *vp* has no effect, because it is not a triggering type. The same is true for the heads and searchers of the third, fourth, fifth and sixth type in (13). Now *geven* will be considered. Among its lexical categories we find some that contain gaps, like $s\backslash[np^{\wedge}gap\ np\ np]/[\ ]$ and $vp\backslash[np\ np^{\wedge}gap]/[]$. Most of them will be rejected for concatenation to that prefix, since a category containing a gap cannot be processed from <a,0>. The only option for a gapped category of *geven* would be one that is headed by a finite main sentence type (hms), for this category would bring the FST in state <e,1>, introducing a gap option. This option is rejected, however, because of other filtering mechanisms apart from the left dislocation FST. Only categories headed by *vp* with no gapped argument remain as possible continuations of the prefix. The number of assignments that has to be parsed (checked for derivability) is seriously cut down.

4 The Effect of Chain Filtering

We have tested the effect of chain filtering by computing three values for a certain set of sentences:
(a) the number of full string assignments left under chain filtering
(b) the number of full string assignments left without chain filtering
(c) the number of full string assignments in case the lexicon would have no gaps.
In the latter case chains must be identified by hypothetical reasoning. In our approach gaps are fixed per full string assignment: no additional hypotheses as to the possible occurrences of gaps are necessary while parsing.

In (figure 1) at the end of the paper the table of results is given; the results are presented as natural logarithms to express their order of magnitude. They are ordered with respect to the length of the sentences in the test set (column a). All counts are submitted to a cluster of other filtering devices which are not related to left dislocation, but which do a major job at distinguishing viable from inviable prefixes, as was discussed for example (8). The numbers of full string assignments, which survived the general filtering devices including and excluding the Chain Filter, mark a rather undetermined stage in the processing of the sentences; the strings counted below are not necessarily all parsed. Most of the sentences are coordinated sentences, which complicates any filtering of prefixes: the selection is *pre*-parsing, the coordinates are not yet determined at that stage, and this indeterminacy must be reflected in weakened application of the FST (which is not spelled out above).

The exponents given in column (f) of the table hold the main result of chain filtering. They indicate the difference between the number of sequences of categories that must be processed if chain filtering is applied (column d) and the number of sequences of categories that must be processed if chain filtering is not applied (column e). Both numbers can be compared to the number of sequences that would survive Count Invariance if the lexicon would not contain gapped categories (column c). In that case, however, DELILAH will not be able to parse left dislocation any longer. Column (b) lists the Cartesian product over w_i, i.e. without any filtering, but including gapped categories. It defines the upper bound for the exponents in the columns (c), (d) and (e).

From these figures one can see that the Left Dislocation Chain Filter has a measurable effect on the number of sequences that must be parsed. It can reduce the number of full string assignments under consideration prior to parsing with an average of one half to one order of magnitude (column f), depending on the nature of the assignments, i.e. the nature of the sentence. In the final case, for example, the application of the Chain Filter reduces the number of full string assignments at this particular stage of processing by a factor of $e^{1.39} = 4$. All possible analyses are kept in store, however; in that respect, chain filtering is as conservative as necessary.

The number of sequences left after chain filtering is still considerably larger than the number a parser checking dislocation by hypothesising gaps would have to consider. This is not surprising. Specifying gaps lexically introduces at least n additional sequences for every gap-less sequence of assignments with n extractable arguments. It depends on the parsing procedure to what extent this complicates the parsing of the sentence. DELILAH can parse these additional assignments marked for gaps deterministically: the number of assignments is the only factor affecting the complexity of the solution to the problem of left dislocation.

It is by no means clear that the Left Dislocation Chain Filter is stated in the best possible way. It must be stressed, however, that in the presence of coordination – all but three of the sentences measured above are coordinated ones – filtering left-dislocated chains is weakened by necessary precautions with respect to across-the-board phenomena. As long as one does not know what is coordinated exactly, the substring to the right of a coordinating element may have to accommodate all the chains that were possibly established at the left of the coordinator. In this respect, the results show that chain filtering keeps performing under difficult conditions.

Acknowledgements

We would like to thank two anonymous reviewers for their valuable comments, Rob Goedemans for checking our English, and Ton van der Wouden for translating the text into LaTeX.

The system DELILAH is available at
http://fonetiek-6.LeidenUniv.nl/hijzlndr/delilah.html.

column a: sentence length in words
column b: natural logarithm (*ln*) of the number of full string assignments (unfiltered; $\Pi_1^n |A(w_i)|$) for a lexicon with gapped categories
column c: (*ln* of the) number of full string assignments (filtered by independent checks) for a lexicon without gapped categories
column d: (*ln* of the) number of full string assignments (filtered) for a lexicon with gapped categories, and **with** application of chain filtering
column e: (*ln* of the) number of full string assignments (filtered) for a lexicon with gapped categories, and **without** application of chain filtering
column f: difference d − e; effect of chain filtering, in orders of magnitude; a negative effect means reduction of the search space

a	b	c	d	e	f
#words	Cart. product over w_i +gapped cats.	#assignments -gapped cats. -Chain Filter	#assignments +gapped cats. +Chain Filter	#assignments +gapped cats. -Chain Filter	Chain Filter effect
7	1.60	0.69	1.09	1.09	0.00
8	4.02	2.07	3.17	3.17	0.00
9	9.91	4.14	6.76	6.83	-0.07
10	4.02	2.07	3.17	3.17	0.00
11	11.38	0.0	2.07	2.77	-0.70
13	10.13	3.17	5.41	5.77	-0.25
14	14.02	3.17	7.32	7.57	-0.25
15	6.10	3.46	4.85	4.85	0.00
16	15.02	4.27	8.58	8.95	-0.37
17	14.45	3.87	7.76	8.41	-0.65
18	18.65	1.79	5.54	7.56	-2.02
19	17.44	5.25	9.77	10.22	-0.45
20	16.60	5.06	10.06	10.63	-0.57
21	13.68	4.85	9.63	10.02	-0.39
22	20.06	3.46	9.10	10.58	-1.48
23	15.83	4.56	9.36	9.80	-0.44
24	8.05	3.46	6.64	6.64	0.00
25	18.31	7.09	13.66	13.99	-0.33
26	9.57	4.85	8.72	8.72	0.00
28	21.03	5.95	12.07	12.58	-0.51
29	26.36	5.50	13.10	14.08	-0.98
32	29.43	1.09	7.24	8.65	-1.41
34	23.12	5.25	12.79	13.64	-0.85
37	28.18	4.85	13.23	14.62	-1.39

Figure 1: **Table of Results**

References

Barton, G., R. Berwick, and E. Ristad (1987). *Computational Complexity and Natural Language.* MIT Press.

Benthem, J. v. (1986). *Essays in Logical Semantics.* Reidel.

Benthem, J. v. (1991). *Language in Action.* North-Holland. SLFM 130.

Bouma, G. and G. van Noord (1994). Constraint-based categorial grammar. In *Proceedings 32nd Annual Meeting of the ACL*, pp. 147–154. ACL.

Cremers, C. (1989). Over een lineaire kategoriale ontleder. *TABU 19*(2), 76–86.

Cremers, C. (1993). *On Parsing Coordination Categorially.* Ph. D. thesis, Leiden University. HIL dissertations 5. Also available at ftp://fonetiek-4.LeidenUniv.nl/pub/cremers/dissi.ps.

Cremers, C. and M. Hijzelendoorn (1997). Pruning search space for parsing free coordination in categorial parsing. To appear in: *Proceedings International Workshop on Parsing Technologies*, MIT 1997.

Gazdar, G. and C. Mellish (1989). *Natural Language Processing in PROLOG.* Addison-Wesley Publ Cy.

Hepple, M. (1990). *The Grammar and Processing of Order and Dependency.* Ph. D. thesis, University of Edinburgh.

Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publ Cy.

Johnson, M. (1991). The computational complexity of glr parsing. In M. Tomita (Ed.), *Generalized LR Parsing*, pp. 53–42. Kluwer.

Johnson, M. and M. Kay (1994). Parsing and empty nodes. *Computational Linguistics 20*(2), 289–300.

Joshi, A., K. Vijai-Shanker, and D. Weir (1991). The convergence of mildly context-sensitive grammar formalisms. In P. Sells, S. Shieber, and T. Wasow (Eds.), *Foundational Issues in Natural Language Processing*, pp. 31–82. MIT Press.

Kayne, R. (1994). *The Antisymmetry of Syntax.* MIT Press.

König, E. (1990). *Der Lambek-Kalkul. Eine Logik für lexikalische Grammatiken.* Ph. D. thesis, Universität Stuttgart. IWBS Report 146.

Koot, J. v. d. (1990). *An Essay on Grammar-Parser Relations.* Ph. D. thesis, University of Utrecht.

Milward, D. (1995). Incremental interpretation of categorial grammar. In *Proceedings 7th EACL*, pp. 119–126. Dublin: EACL.

Moortgat, M. (1988). *Categorial Investigations.* Foris.

Morrill, G. (1994). *Type Logical Grammar.* Kluwer.

Pereira, F. and S. Shieber (1987). *Prolog and Natural Language Analysis.* CSLI.

Stabler, E. (1992). *The Logical Approach to Syntax.* MIT Press.

Steedman, M. (1990). Gapping as constituent coordination. *Linguistics and Philosophy 13*(2), 147–171.

Wittenburg, K. (1986). *Natural Language Parsing with Combinatory Categorial Grammar in a graph-unification-based Formalism.* Ph. D. thesis, University of Texas at Austin.