CGN to Grail

Extracting a Type-logical Lexicon From the CGN Annotation

Michael Moortgat and Richard Moot

Utrecht institute of Linguistics OTS

Abstract

The tag set for the CGN syntactic annotation is designed in such a way as to enable a transparent mapping to the derivational structures of current 'lexicalized' grammar formalisms. Through such translations, the CGN tree bank can be used to train and evaluate computational grammars within these frameworks.

In this paper we will discuss some preliminary work on the mapping between the CGN annotation graphs and the proof net format of the Grail parser/theorem prover (Moot 2001, Moot 1999). Grail is a general grammar development environment for type-logical categorial grammars (TLG, (Moortgat 1997, Morrill 1994, Carpenter 1998)). To a large extent, there is a straightforward transfer between the type-logical format and the analyses provided by other lexicalized grammar formalisms such as LTAG (lexicalized Tree Adjoining Grammars, (Sarkar 2001)) and MG (computational versions of Minimalist Grammars, (Stabler 1997)). An attractive feature of TLG, which is not shared by these other frameworks, is its full support for hypothetical reasoning.

In this paper, we exploit the hypothetical reasoning facilities to extract a type-logical grammar from the CGN annotation graphs. This task can be naturally divided in two subtasks. The first of these consists in solving type equations: in the TLG setting this means breaking up the CGN annotation graph into the subgraphs that correspond to lexical type assignments. In the presence of discontinuous dependencies, the lexical type assignments will not always be compatible with surface word order. The second subtask then consists in calibrating the lexicon in such a way that it has controlled access to the structural reasoning component of the grammar.

1 Introduction

The aim of the Spoken Dutch Corpus CGN (Corpus Gesproken Nederlands) is to build a database of contemporary spoken Dutch. The final version will contain around 10 million words. In addition to providing 1000 hours of audio, the corpus will be annotated in various ways: the entire corpus will be tagged for part-of-speech information, parts of the corpus will receive a phonetic transcription and be tagged for prosodic information, and so on.

In this paper we will focus on the syntactic annotation component. A total of one million words of the corpus will be syntactically annotated, part manually, part automatically. The annotation software used to this purpose is the *annotate* tool (Plähn 2000), which was developed at the university of Saarbrücken and which has been successfully used for the annotation of the NEGRA corpus.

For data exchange, *annotate* provides a line-oriented, ASCII based export format that can be efficiently processed by applications that want to make use of the annotation information. The philosophy behind the CGN syntactic annotation schema is to provide an informationally rich, but theory-neutral annotation level. The export format then allows users to convert the annotation information in the way which is most convenient for them; that is, we want to *derive* theory specific notions from the theory neutral export format.

As an illustration of this approach, we present a translation of the CGN annotation graphs into the type-assignments and proof nets of Type Logical Grammar (TLG, (Moortgat 1997, Morrill 1994, Carpenter 1998)). To a large extent, type-logical analyses are compatible with analyses within other lexicalized computational grammar formalisms—see for example (Moot 2000) on the encoding of TAGs into TLG proof nets, and (Vermaat 1999) on the embedding of Stabler-style Minimalist Grammars into TLG. But as a logic-based framework, TLG has extra inferential possibilities. In this paper, we will exploit the full support for *hypothetical reasoning* to induce a type-logical grammar from the CGN syntactic annotation. The first step is to extract a type-logical lexicon from the CGN annotation graphs: the approach described in §4 provides an algorithm based on hypothetical reasoning for solving the type-assignment equations. To make the solutions for these type equations compatible with surface word order, some form of structural reasoning is necessary. The CGN annotation graphs provide a tree bank to determine from data what the appropriate structural package would be. In §5, we discuss the fine-tuning of the interface between the lexicon and the structural reasoning component of the grammar.

2 CGN Annotation Graphs

The CGN annotation uses directed acyclic graphs (DAGs), where the vertices are labeled by syntactic categories and the edges labeled by dependency relations. We will only give a brief exposition of the basic ideas behind the annotation format, full details can be found in (Hoekstra, Moortgat, Schuurman and van der Wouden 2001) in this volume.

The annotation graphs allow us to specify structures which are unlike 'typical' tree-based grammatical descriptions. DAGs are allowed to be disconnected, DAGs can have discontinuous constituents or 'crossing branches' and DAGs can have multiple dependencies, where a single constituent plays a grammatical role in more than one domain. An example of a CGN annotation graph, which we will use as a running example throughout this article, is given in Figure 1. The direction of the edges is implicit in the graph; if we would draw the direction of the edges explicitly, they would all point downwards. Note that the *wh* word 'wat' (what) gives an example of multiple dependencies: it has the grammatical role of [whd], the head of the *wh* question WHQ, but at the same time it functions as the direct object [obj1] in the INF domain. The advantage of this form of annotation is that phrasal category and dependency information can be expressed without traces or other syntactic elements without phonological realization.

Another point worth noting is that we produce very flat annotation structures; a syntactic domain is only introduced when it is necessitated by a new head.

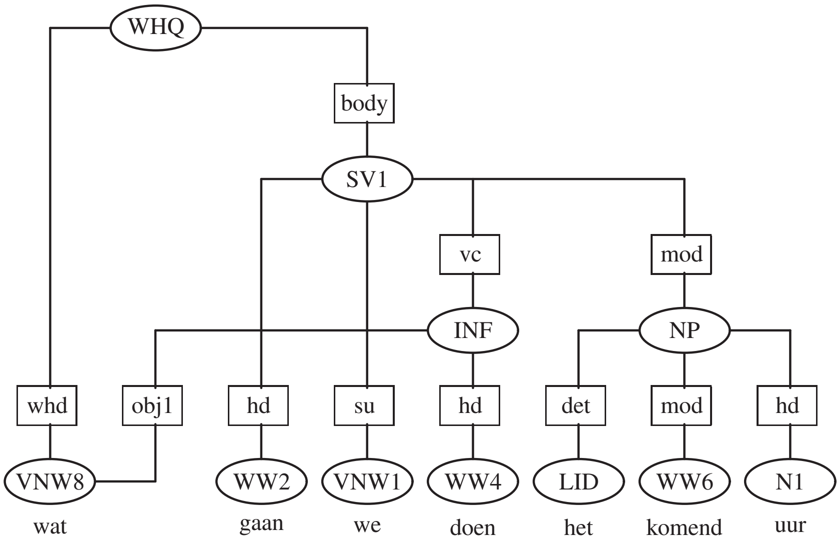


Figure 1: A CGN annotation graph

3 Categorial Proof Nets

The categorial proof nets presented in this section are essentially the same as those used by (Moot and Puite 2001). We refer the reader to that paper for formal results, showing soundness and completeness of these proof nets with respect to the sequent calculus for multimodal categorial grammar of (Moortgat 1997), and give only an informal introduction here.

Definition 1 *A* categorial proof net system *consists of the following:*

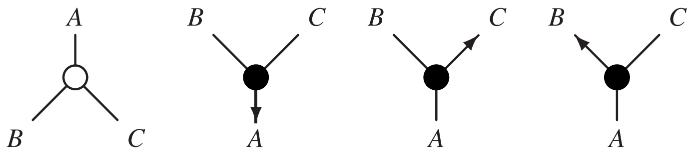
[Terminals] *Terminals are the lexical words of our grammar. We denote a word as a terminal by enclosing it in a square box, as follows.*

$$\boxed{\text{alcohol}} \qquad \boxed{\text{Apeldoorn}} \qquad \boxed{\text{reclame}} \qquad \cdots$$

[Nonterminals] *Nonterminals denote the syntactic types of expressions.*

$$s,\ n,\ np,\ \ldots$$

[Constructors] *Finally, we have constructors which allow us to make complex expressions out of terminals and nonterminals. There are four basic binary constructors.*



The downward branching constructor, the main *constructor, has no constraints associated with its use.*

The upward branching constructors, which we will call auxiliary *and which we draw with a black center, denote* constraints *on the use of their lexical entry, in a sense to be made precise later. The only difference between the three auxiliary constructors is which of the three points it connects is the output, as indicated by the arrow.*

More formally, a constructor C is a tuple $\langle Type, P, Q, p, q\rangle$, where $Type$ is either 0 (black) or 1 (white), P is a sequence of vertices which are 'above' C, Q is a sequence of vertices which are 'below' C, p is a subsequence of P, the outputs which are 'above' C, and q is a subsequence of Q, the outputs which are 'below' C, such that length(p) + length$(q) \leq 1$, that is, a constructor has at most one output.

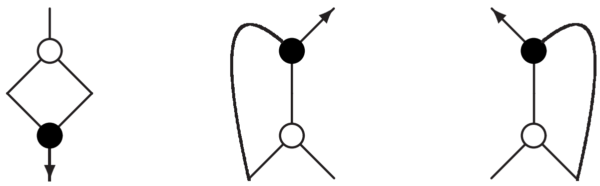
In this notation, we write the four constructors as $\langle 1, [A], [B, C], [], []\rangle$, $\langle 0, [B, C], [A], [], [A]\rangle$, $\langle 0, [B, C], [A], [C], []\rangle$ and $\langle 0, [B, C], [A], [B], []\rangle$.

Definition 2 *A* lexical entry *for a categorial proof net system is a free tree made from constructor nodes such that*

1. *every root node (there can be multiple root nodes because of the auxiliary constructors) is labeled with a nonterminal symbol.*
2. *every leaf is labeled with either a nonterminal or a terminal symbol.*
3. *at least one leaf of every lexical entry is labeled with a terminal symbol.*

Because the auxiliary constructors have more than one parent node, they prevent the graph we are constructing from being a *rooted* tree. To remove these auxiliary constructors, we define the following graph contractions on the graphs in our system.

Definition 3 *We define the following* graph contractions *on categorial proof net systems, one for each of the auxiliary constructors. Whenever we find one of the following three configurations of constructors, we can contract this configuration to a single point.*



Note that all contractions are of the same general form: they combine an auxiliary constructor with a main constructor on both ends not marked by the arrow and in a way which respects the up-down and left-right ordering of the nodes.

Also note that drawing these graphs on a plane sometimes requires us to bend one of the connections, because we want to keep all up-down and left-right distinctions explicit in the graph. These bends disappear if we draw the graphs on a cylinder.

Definition 4 *A* grammatical expression *of type t in a proof net system is a graph which contracts to a rooted tree T, with t as its root, and where all leaves are labeled with terminals.*

Example 1 *An example lexicon for a categorial proof net system is given in Figure 2. We have simple lexical entries, like 'Albanië' (Albania) and 'politie' (police), which simply assign a syntactic category to a word, but also more complex lexical entries, like 'de' (the) which combines with a syntactic expression of category n to its right to form a syntactic expression of category np. Similarly, the transitive verb 'steunt' (supports) combines with an np to its right and with an np to its left to form an expression of type s.*

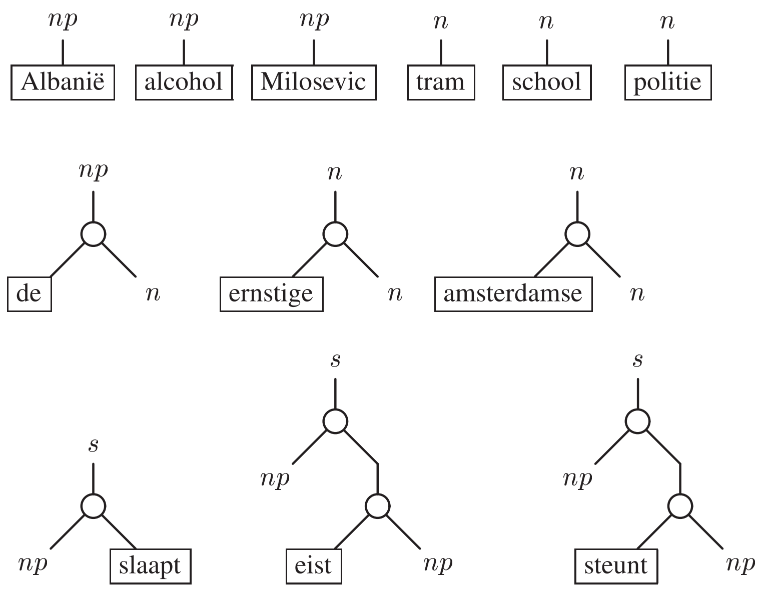


Figure 2: Some simple lexical graphs

We can use the auxiliary constructors to assign quantifiers like 'iemand' (someone) the lexical graph given in Figure 3.

This lexical entry indicates that 'iemand' selects an s to produce an s, where we can use an np inside this s, subject to the condition that the special constructor can

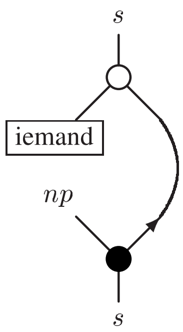


Figure 3: Lexical graph for a generalized quantifier

be contracted according to Definition 3. The difference between the assignment of 'iemand' to that of simple np*'s like 'Albanië' is that the assignment above allows 'iemand' to take scope at sentence level as a generalized quantifier.*

Example 2 *We can derive 'iemand slaapt' (someone sleeps) to be a well-formed expression of type* s *by the derivation shown in Figure 4. First, we connect the bottom* s *of 'iemand' to the* s *of slaapt, which results in the structure on the left. After identifying the two* np *nonterminals, the structure will look as shown in the middle of Figure 4. Note that this structure is of the proper form to apply the contraction for the auxiliary constructor, as indicated by the dotted box around the redex. The resulting tree after the contraction is pictured on the right.*

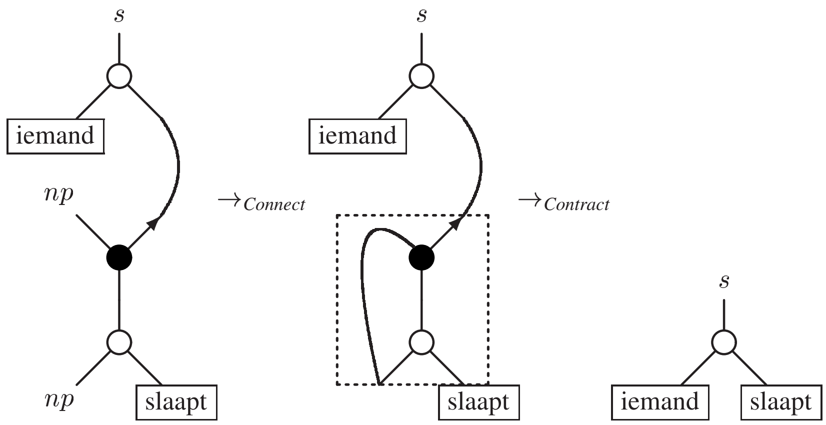
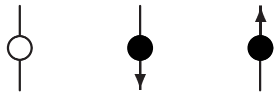


Figure 4: Derivation of 'iemand slaapt'

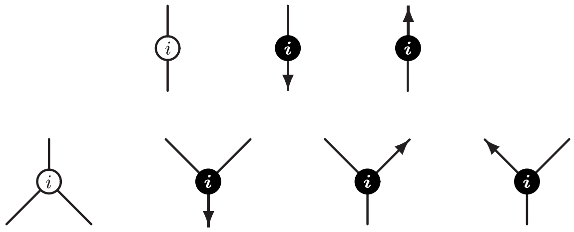
To give a direct correspondence to the multimodal sequent calculus, the full system described in (Moot and Puite 2001) is more extensive than the one described above in a number of ways. We will see later that we need some of these extensions for our proposed translation.

First of all, we allow our lexical graphs to have unary branches, which look as follows.

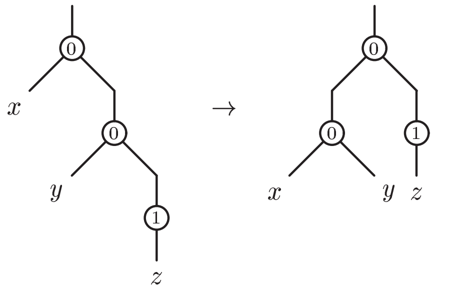


As with the binary constructors, the only difference between the unary auxiliary constructors is in the arrow which indicates the output connection. We can contract a main and an auxiliary unary connector if they are connected at the point which is not the output of the auxiliary constructor, just like with the binary constructors.

Secondly, we allow our constructors to have different modes of composition by writing an index i, out of a finite set of possible indices I, inside the constructor, as follows.



Finally, in addition to the contractions we allow a grammar to specify *structural conversions* which convert one tree of main constructors into another tree of main constructors with the same leaves. An example of a structural conversion, where 0 and 1 are elements of I, would be the following.



There is a straightforward correspondence between the nets described in this section in terms of unary and binary constructors and the type language of TLG

(Moortgat 1997), where we build complex types out of atoms by means of unary and binary connectives (indexed for composition modes, in the case of a multimodal system):

$$\mathsf{Type} ::= \mathsf{Atom} \mid \Diamond\mathsf{Type} \mid \Box\mathsf{Type} \mid \mathsf{Type}/\mathsf{Type} \mid \mathsf{Type} \bullet \mathsf{Type} \mid \mathsf{Type}\backslash\mathsf{Type}$$

As an example, type assignments corresponding to the lexical nets from (1) would take the following form:

$$\text{de} \vdash np/n \quad \text{amsterdamse} \vdash n/n \quad \text{steunt} \vdash (np\backslash s)/np \quad \ldots$$

This correspondence extends to the contractions and structural operations defined on nets. The deductive counterpart of the graph contractions of Definition 3 are the residuation laws below.

$$\Diamond A \vdash B \quad \text{iff} \quad A \vdash \Box B$$

$$A \vdash C/B \quad \text{iff} \quad A \bullet B \vdash C \quad \text{iff} \quad B \vdash A\backslash C$$

Structural conversions on the nets correspond to structural postulates (non-logical axioms) in the deductive presentation. Below, as an example, the postulate corresponding to the structural rewriting we gave before. Note that the structural conversions perform, from the logical point of view, a type of backward chaining proof search and therefore the direction of the structure postulate needs to be reversed.

$$(A \bullet_0 B) \bullet_0 \Diamond_1 C \vdash A \bullet_0 (B \bullet_0 \Diamond_1 C)$$

Since it is proved in (Moot and Puite 2001) that the proof nets and the type language of TLG are two ways of expressing the same information, we will feel free in the rest of the paper to shift between the nets and the formula presentation where this is appropriate.

4 Extracting the lexicon

The remainder of this paper will be devoted to translating the DAGs of the CGN corpus into the proof nets of the previous section. This translation is parametric in a number of ways.

Firstly, we need to be able to identify the *functor* of every local domain. Usually, this will be the head, indicated by the edge label hd, but in some cases we might want to diverge from this.

Secondly, we need to be able to identify the *modifiers* of every domain. Usually modifiers will be indicated by the edge label mod but we might want to assign some other syntactic roles a modifier function.

A final parameter is whether we want to translate the annotated CGN graphs into a set of lexical graphs or into a single graph where all connections of nonterminals are already explicit. The first choice will be useful to test the predictions of the annotated corpus on new sentences, whereas the second choice will be useful to see which structural conversions we need to add to the system to produce the right word order. We return to this issue in §5.

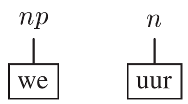
4.1 Basic Entries

When translating the basic entries, the issue of granularity surfaces. The tags for some words differ only in the morphological information. For example, at the level of the leaves of the CGN annotation graphs, the syntactic category VNW for ‘voornaamwoord’ (pronoun) has 19 different instantiations depending on whether it is a personal pronoun, a reflexive pronoun, a demonstrative pronoun, and so on. Do we want to distinguish all of these in the translation, or do we want to conflate some of these distinctions?

Determining the appropriate level of granularity is a matter of grammatical fine-tuning. For expository purposes, we stay at a rather coarse level in this paper. We will translate the syntactic categories of our example sentence into nonterminal categories as follows.

VNW1 $\rightarrow np$
VNW8 $\rightarrow np$
N1 $\rightarrow n$
INF $\rightarrow$ *inf*
SV1 $\rightarrow sv1$
WHQ $\rightarrow whq$

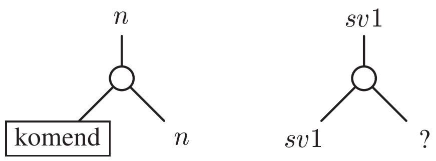
With this translation in hand, we can immediately assign two of the words of the example a lexical entry, as shown below.



4.2 Modifiers

The example sentence of Figure 1 has two modifiers: ‘komend’ (next) is a modifier at the np level, whereas the phrase ‘het komend uur’ (the next hour) is a sentence level modifier. We repeat the relevant part of Figure 1 in Figure 5.

The n modifier is lexically anchored and is in fact the same lexical graph used for noun modifiers in the example lexicon in Figure 2. The translation for the $sv1$ modifier is still partial, since it depends on the translation of the functor of the NP domain.



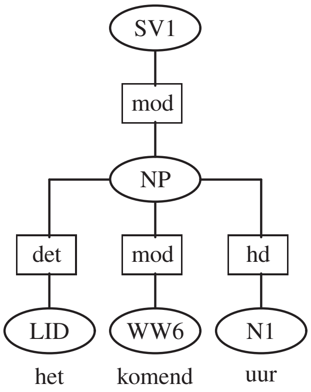
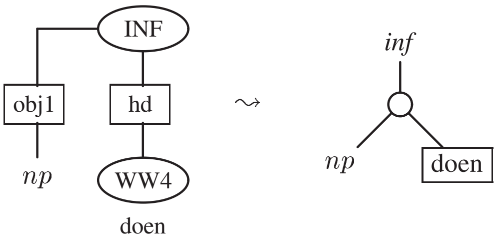


Figure 5: Modifiers from Figure 1

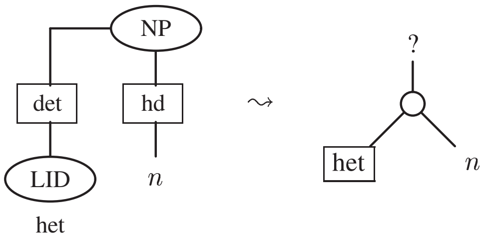
4.3 Functors

For functors, we again have to make a choice: do we want to follow the surface structure as much as possible and basically generate the words of the sentence in the right order, or do we want to assign functors a structure which is as canonical as possible, which would reduce the number of different lexical assignments and require us to *derive* the other possibilities from this canonical structure via some appropriate form of structural rewriting. For the moment we choose the first option and we defer the discussion of structural reasoning to §5.

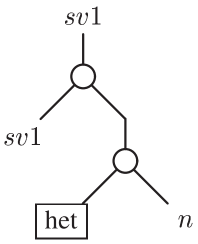
The functor ‘doen’ (to do) selects a direct object np to its left to produce an *inf* category. This is coded by the following lexical graph.



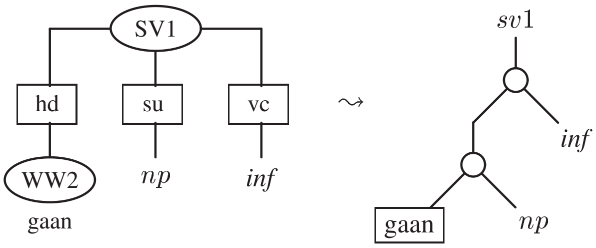
The functor ‘het’ (the) selects a noun to its right to produce the translation of the np category.



As we have seen in the previous section this translation is an $sv1$ modifier, so the final result will select an n to its right to produce an $sv1$ modifier as follows.



The auxiliary verb 'gaan' (go) selects for a subject np and an infinitival complement *inf*, which is translated as follows.



4.4 Multiple Dependencies

Because the auxiliary links for lexical proof structures have more than one parent, it seems evident we can use auxiliary links to encode the multiple dependencies which are possible in the CGN annotation graphs.

The multiple dependency in our example, schematically repeated as Figure 6 fo convenience, will be converted as follows, indicating that the question word 'wat' (what) produces an expression of type whq if it finds a constituent of type $sv1$ to its right with a hypothetical subconstituent of type np in it. For some readers it may be helpful to picture this hypothesis as the 'trace' bound by the question word.

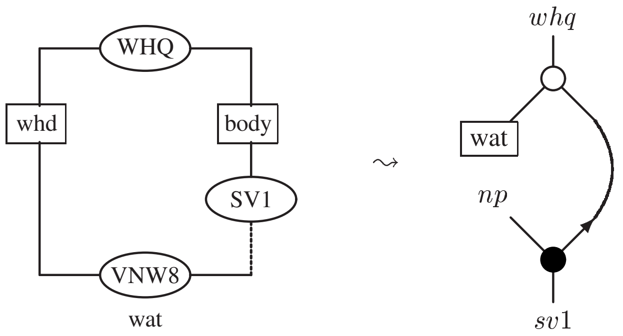


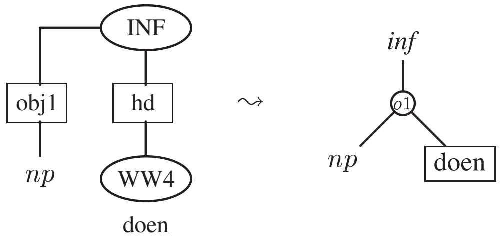
Figure 6: The multiple dependency from Figure 1

The introduction of an auxiliary constructor in the lexical graph commits us to contract this constructor, thus withdrawing the np hypothesis. Creating the appropriate configuration for contraction may require the use of structural conversions. We will return to the topic of structural conversions in §5, where they allow us to derive discontinuous constituents.

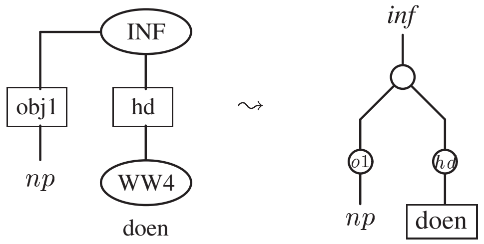
4.5 Edge Labels

So far, we have treated the edge labels as only providing us with the information we need to determine which structures are functors and which structures are modifiers. We now want to refine the translation function to also take the information about the dependency relations into account.

In the example below, 'doen' (to do) selects an np which functions as a direct object [obj1]. One possibility is to encode this information into the mode of composition, assigning 'doen' the type $np\backslash_{\mathrm{obj1}} inf$, as in the graph below. Note that the [hd] syntactic relation is implicit in this encoding, in the sense that functors and heads are identified. Note also, that in a language (like Dutch) with both head-initial and head-final phrases, one would have to take the head position into account. This can be done by distinguishing, say, *l(M)* versus *r(M)* for left- versus right-headed structures, assigning the dependency role M to the non-head component. In the case of our head-final transitive infinitive, this yields the type $np\backslash_{\mathrm{r(obj1)}} inf$.



An alternative solution would be to encode the grammatical relations as *unary* branches in the lexical graph. This allows us to code also the [hd] edge label explicitly, as in the graph below. In the remainder, we go for the first option, because we want to reserve the unary connectives for another purpose — they will act as control features for structural reasoning.



4.6 Implementation

As already suggested by the previous sections, the translation from the CGN annotation graphs to the TLG framework can be fully automated. The implementation of the conversion, given in Figure 7 proceeds on the assumption, which does not necessarily hold of general DAGs, but which is true of the DAGs we use for the CGN annotation, namely that every connected component of the DAG has a unique root vertex.

An implementation of the conversion utility is available through ftp at the following URL.

`ftp://ftp.let.uu.nl/pub/users/moot/cgn.tar.gz`

5 Discontinuous dependencies and structural reasoning

As remarked above, the dependency relations coded in the CGN annotation graphs can be at odds with surface order and constituency. In our example of Figure 1, we already see an illustration of such a discontinuous dependency: the secondary edge linking the question word ‘wat’ to the direct object role within the infinitival complement headed by ‘doen’ crosses the finite verb and subject edges.

To make the lexical type-assignments compatible with surface order, we have to combine the categorial base logic with some form of structural reasoning. Earlier versions of categorial grammar were ill equipped to deal with the combination

```
BEGIN
FOR EVERY component c of C
    LOOK UP the formula F corresponding the the unique root vertex v
    TRANSL(v,F)
END FOR
END

PROC TRANSL(v,F)
    IF v is a leaf corresponding to word w
        add w with formula F to the lexicon
    ELSE
        FOR EVERY daughter d with edge label e of v
            IF d is a modifier
                TRANSL(d,F\F)
            ELSE IF d is a complement
                LOOK UP the formula D corresponding to d
                TRANSL(d,f1\r(e1) ... fi\r(ei) D)
                WHERE f1 ... fi are the formulas corresponding to
                    secondary edges of descendants of d
            ELSE IF d is a head
                TRANSL(d,(f1\r(e1) ... fi\r(ei) F)/l(ej) fj .../l(ej) fm)
                WHERE f1 ... fi are the formulas corresponding to
                    complements occurring to the left of d
                WHERE fj ... fm are the formulas corresponding to
                    complements occurring to the right of d
            END IF
        END FOR
    END IF
END PROC
```

Figure 7: The translation algorithm

of logical and structural inference, because they were operating from an essentially one-dimensional perspective on grammatical composition. If there is only one composition operation around in the grammar, attributing structural properties to this operation (such as associativity, or commutativity) has a *global* effect, destroying structural discrimination (for constituency or linear order) throughout the grammar. What is needed instead of such global choices, is lexically controlled, local options for structural reasoning.

The multimodal architecture of TLG provides for this form of structural control. In the presence of multiple modes of composition, one can differentially treat the structural behavior of individual modes and of their interaction. A constituent bearing the [obj1] dependency role, for example, could have a different structural behavior from a subject constituent. The unary type-forming connectives ($\Diamond$ and the residual $\Box$ in the type language) in this respect act as *licensing* features, providing controlled access to structural inferences.

The expressive power of the unary constants is by now well understood. The embedding results of (Kurtonina and Moortgat 1997) show that every corner of the categorial landscape is in effect reachable by means of the unary control features. (If one goes beyond resource-sensitive composition modes, the unary operators even allow an embedding of Intuitionistic Logic, as shown in (Lambek 1993).) Where exactly in this landscape the natural languages have to be localized, is a big open research question in TLG. An annotated corpus such as CGN provides a valuable tree bank to address this question from a data-oriented perspective.

5.1 The structural package

We are currently experimenting with the structural package below (from (Moortgat 1999)) that seems to have a pleasant balance between expressivity and structural constraint. We first discuss the postulates in schematic form — further fine-tuning in terms of mode distinctions for the $\bullet$ and $\Diamond$ operations is straightforward.

$$\begin{array}{lcll}
\Diamond A \bullet (B \bullet C) & \dashv\vdash & (\Diamond A \bullet B) \bullet C & (Pl1) \\
\Diamond A \bullet (B \bullet C) & \dashv\vdash & B \bullet (\Diamond A \bullet C) & (Pl2) \\
\\
(A \bullet B) \bullet \Diamond C & \dashv\vdash & (A \bullet \Diamond C) \bullet B & (Pr2) \\
(A \bullet B) \bullet \Diamond C & \dashv\vdash & A \bullet (B \bullet \Diamond C) & (Pr1)
\end{array}$$

The postulates can be read in two directions. In the *Output* $\vdash$ *Input* direction, they have the effect of *revealing* a $\Diamond$ marked constituent, by promoting it from an embedded position to a dominating position where it is visible for the logical rules. In the *Input* $\dashv$ *Output* direction, they *hide* a marked constituent, pushing it from a visible position to an embedded position. Apart from the $\dashv\vdash$ asymmetry, there is a left-right asymmetry: the Pl postulates have a bias for left branches; for the Pr postulates only right branches are accessible.

We highlight some properties of this package.

Control The postulates operate under $\Diamond$ control. Because the logic doesn't allow the control features to enter a derivation out of the blue, this means they have to be lexically anchored.

Linearity The postulates rearrange a structural configuration; they cannot duplicate or waste grammatical material.

Locality The window for structural reasoning is strictly local: postulates can only see two products in construction with each other (with one of the factors bearing the licensing feature).

Recursion Non-local effects arise through recursion.

5.2 Calibrating the lexicon/syntax interface

In order to give the lexical type assignments of §4 access to structural reasoning, we have to systematically refine them with the licensing control features. We follow the 'key and lock' strategy of (Moortgat 1999), which consists in decorating positive subtypes with a $\Diamond\Box$ prefix. For a constituent of type $\Diamond\Box A$, the $\Diamond$ component provides access to the structural postulates discussed above. At the point where such a marked constituent has found the structural position where it can be used by the logical rules, the control feature can be cancelled through the basic law $\Diamond\Box A \vdash A$ — the $\Diamond$ key unlocking the $\Box$ lock.

We illustrate the effect of the $\Diamond\Box$ decoration on the lexical type assignments for our running example of Figure 1. Note that the positive subtype np in the type assignment to the question word 'wat' (the 'gap' hypothesis) gains access to structural reasoning by means of its *se* decoration (for secondary edge).

doen : $\Diamond_{hd}\Box_{hd}(np\backslash_{r(obj1)}inf)$
gaan : $\Diamond_{hd}\Box_{hd}((s1/_{l(vc)}inf)/_{l(su)}np)$
het : $\Diamond_{det}\Box_{det}(\Diamond_{mod}\Box_{mod}(s1\backslash s1)/_{l(hd)}np)$
komend : $\Diamond_{mod}\Box_{mod}(\Diamond_{mod}\Box_{mod}(s1\backslash s1)/\Diamond_{mod}\Box_{mod}(s1\backslash s1))$
uur : np
wat : $\Diamond_{whd}\Box_{whd}(whq/_{l(body)}(\Diamond_{se}\Box_{se}np\backslash_{r(obj1)}s1))$
we : np

For reasons of space, we shorten the example to 'Wat gaan we doen?' ('what shall we do') — this provides enough information to see how the discontinuous dependency is established, and step through the proof net derivation of the simplified sentence. Figure 8 shows the net with the right connections on the left. Note that the two occurrences of x correspond to the same vertex in the graph. For a successful contraction as required by Definition (4), the direct object hypothesis labeled x has to be moved upward. For this we need the following mode-instantiated version of postulate $Pl2$.

$$\Diamond_{se}A \bullet_{r(obj1)} (B \bullet_{l(vc)} C) \vdash B \bullet_{l(vc)} (\Diamond_{se}A \bullet_{r(obj1)} C) \quad (Pl2)$$

After the $Pl2$ structural rewriting, the unary and binary redexes are all in the right configuration for contraction, as shown in Figure 8 on the right. The resulting tree is displayed in Figure 9.

Note that in this derivation, only the licensing feature on the hypothesis np subtype for 'wat' played an active role — the inert control features in that type assignment could be simplified away. We can anticipate that the *mod* feature for the sentential modifier 'het komend uur' ('the next hour') will be active too, if we want to derive our running example and the variant 'wat willen we het komend uur gaan doen' from the same type assignments. In this variant the modifier separates the infinitival complement from its head — a structural conversion that can be accomplished by $Pr2$ (in the 'hiding' $\dashv$ direction).

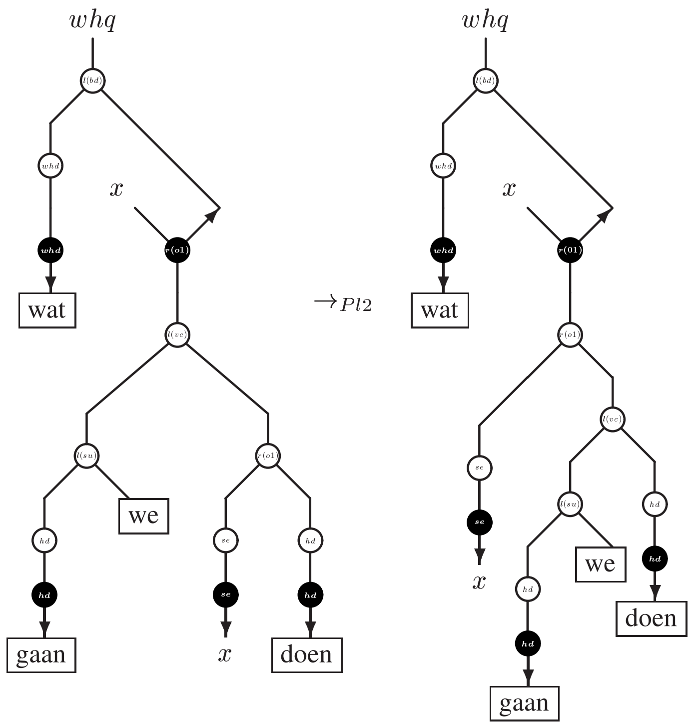


Figure 8: The application of the $Pl2$ conversion

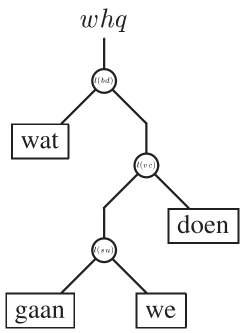


Figure 9: The resulting tree

6 Concluding Remarks

We have shown that the theory neutral annotation format used by CGN contains enough information to construct a type logical lexicon from it. The translation we have proposed is parametric in a number of respects. Our general strategy now is to use the growing CGN tree bank to determine which dependency modes should have access to which structural postulates, and to find out what the proper balance is between storage (the tolerated amount of lexical ambiguity) and computation (the complementary amount of on-line structural reasoning needed).

References

Carpenter, B.(1998), *Type-logical Semantics*, MIT Press.

Hoekstra, H., Moortgat, M., Schuurman, I. and van der Wouden, T.(2001), Syntactic annotation for the spoken Dutch corpus project (CGN), *in* W. Daelemans (ed.), *Proceedings of CLIN2000*.

Kurtonina, N. and Moortgat, M.(1997), Structural control, *in* P. Blackburn and M. de Rijke (eds), *Specifying Syntactic Structures*, CSLI, Stanford, pp. 75–113.

Lambek, J.(1993), From categorial grammar to bilinear logic, *in* K. Došen and P. Schröder-Heister (eds), *Substructural Logics*, Oxford University Press, Oxford, pp. 207–237.

Moortgat, M.(1997), Categorial type logics, *in* J. van Benthem and A. ter Meulen (eds), *Handbook of Logic and Language*, Elsevier/MIT Press, chapter 2.

Moortgat, M.(1999), Constants of grammatical reasoning, *in* G. Bouma, E. Hinrichs, G.-J. Kruijff and R. Oehrle (eds), *Constraints and Resources in Natural Language Syntax and Semantics*, CSLI, Stanford, pp. 195–219.

Moot, R.(1999), Grail: an interactive parser for categorial grammars, *in* R. Delmonte (ed.), *Proceedings of VEXTAL'99*, University Cá Foscari, Venice, pp. 255–261.

Moot, R.(2000), Proof nets and their relation to LTAGs, Manuscript.

Moot, R.(2001), Grail. `http://www.let.uu.nl/~Richard.Moot/personal/grail.html`.

Moot, R. and Puite, Q.(2001), Proof nets for the multimodal Lambek calculus, *in* W. Buszkowski and M. Moortgat (eds), *Studia Logica*, Kluwer. Special Issue Dedicated to Joachim Lambek.

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

Plähn, O.(2000), Annotate. `http://www.coli.uni-sb.de/sfb378/negra-corpus/annotate.html`.

Sarkar, A.(2001), Xtag. `http://www.cis.upenn.edu/~xtag`.

Stabler, E.(1997), Derivational minimalism, *in* A. Lecomte (ed.), *LACL97*, Vol. 1582 of *Lecture Notes in Computer Science*, Springer.

Vermaat, W.(1999), *Controlling movement. minimalism in a deductive perspective*, Master's thesis, Utrecht University.