Using the Spoken Dutch Corpus for type-logical grammar induction[1]

Michael Moortgat[†], Richard Moot[†]

[†]Utrecht Institute of Linguistics – OTS
Trans 10, 3512 JK Utrecht, The Netherlands
{moortgat,moot}@let.uu.nl

Abstract

The dependency-based annotation format employed within the Spoken Dutch Corpus (CGN) project (van der Wouden et al., 2002) has been 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 use the computational facilities of the Grail system (see Moot, 2002) to extract type logical grammars from the CGN annotation graphs. Grail is a general grammar development environment for type-logical categorial grammars (TLG). The Grail parsing engine combines proof net technology with structural rewriting.

1. Type logical grammar

Type logical grammar, as described in (Moortgat, 1997), is a generalization of Lambek categorial grammar. TLG has a two-component architecture. A universal *base component* captures grammatical invariants and provides the input for meaning assembly, along the lines of the Curry-Howard interpretation of derivations. This base component is combined with a *structural module*, which relates the semantically interpreted structures to surface structure configurations. The structural module accommodates cross-linguistic variation; its rules have the status of *non-logical axioms* or postulates. Structural rules do not operate in a global fashion; they have to be explicitly licensed by control features, and are thus lexically anchored.

The TLG type language is obtained by taking a small set of basic types (say, s, n, np, ...), and closing it under n-ary type-forming operations. The type-forming operations come in families consisting of an n-ary structure-building operation, together with its n residuals for disassembling complex structures. The type-forming operations are indexed with a *composition mode* index. The different composition modes all share the same logical rules, given by the residuation inferences below for the unary and binary case, but they can differ in their structural possibilities.

$$\begin{array}{lrll} (R0) & \Diamond_i A \vdash B & \text{if and only if} & A \vdash \Box_i B \\ (R1) & A \bullet_j B \vdash C & \text{if and only if} & A \vdash C/_j B \\ (R2) & A \bullet_j B \vdash C & \text{if and only if} & B \vdash A\backslash_j C \end{array}$$

An illustration of a structural rule is given below. Rather than attributing global associativity to the $\bullet_0$ operation, this rule allows a form of controlled rebracketing, keyed to the $\Diamond_1$ feature.

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

The strict separation of logical and structural components creates subtle tools for probing the mildly context-sensitive territory. The TLG base component is of polynomial complexity. The complexity of the combination with the structural module depends on the constraints one imposes on allowable structural inferences. In (Moot, 2002) it is shown that a linearity constraint (no duplication or waste of grammatical material) corresponds to context-sensitive expressivity (PSPACE complexity), but that with appropriate further restrictions on structural rules and lexical entries polynomial fragments of LTAG expressivity can be identified.

[1]An earlier version of this paper appeared as (Moortgat and Moot, 2001)

2. Proof nets for TLG

The parsing theory for TLG — a refinement of the proof nets originally developed for Linear Logic — directly exploits the resource-sensitivity of the grammar logic. We refer the reader to (Moot and Puite, 2002) for formal results, showing soundness and completeness of these proof nets with respect to the sequent calculus for multimodal categorial grammar of (Moortgat, 1997). Below one finds an informal introduction.

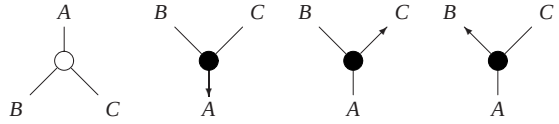
Definition 1 *A* categorial proof net system *consists of the following:*

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

alcohol Apeldoorn reclame ...

[Nonterminals] *The proof net nonterminals are the basic type formulas. For example:* s, np, n, ...

[Constructors] *Finally, we have constructors which allow us to make complex expressions out of terminals and nonterminals. For the binary type-forming operations, our proof nets have four basic binary constructors. The upward branching constructors, drawn with a grey center, which we will call* auxiliary*, denote* constraints *on the use of their lexical entry, in a sense to be made precise later. The downward branching constructor, the* main *constructor, has no constraints associated with its use.*

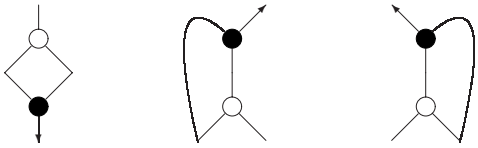


Definition 2 *A* lexical entry *for a categorial proof net system is a tree made from constructor nodes such that*

1. *every root node (there can be multiple root nodes because 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 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. Whenever we find one of the following three configurations of constructors, we can contract this configuration to a single point.*



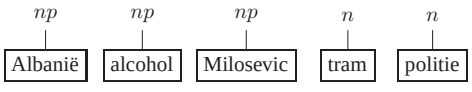
Note that all contractions are of the same general form: they combine an auxiliary constructor with a main constructor on both ends not marked by the arrow and in a way which respects the left-right ordering of the nodes.

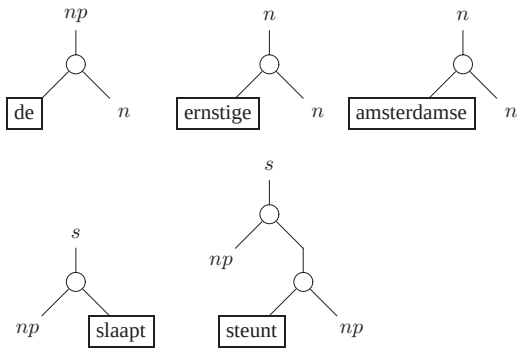
Also note that drawing these graphs on a plane sometimes requires us to bend one of the connections.

Definition 4 *A* grammatical expression *of type t in a proof net system is graph which contracts to a 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 looks as follows.*

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.

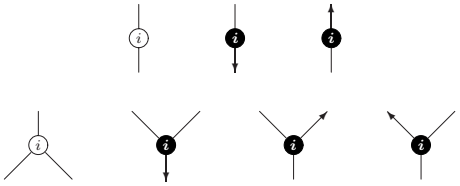




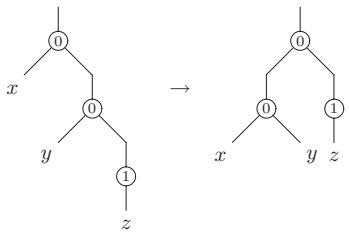
To obtain a direct correspondence to the multimodal sequent calculus, the full system described in (Moot and Puite, 2002) is more extensive than what we have described so far in a number of ways. We will see later that we need some of these extensions for our proposed translation.

First of all, we allow our lexical graphs to have unary branches, corresponding to the $\diamondsuit, \square$ connectives. The unary constructors look as follows.

Secondly, we allow our constructors to have different modes of composition by writing and 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 is given below. It corresponds to the postulate $(A \bullet_0 B) \bullet_0 \diamondsuit_1 C \vdash A \bullet_0 (B \bullet_0 \diamondsuit_1 C)$ we saw in the previous section. 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.



3. From CGN to TLG

In the following sections we discuss the extraction of a type logical grammar from the CGN dependency annotations. The logic+structure architecture of TLG suggests

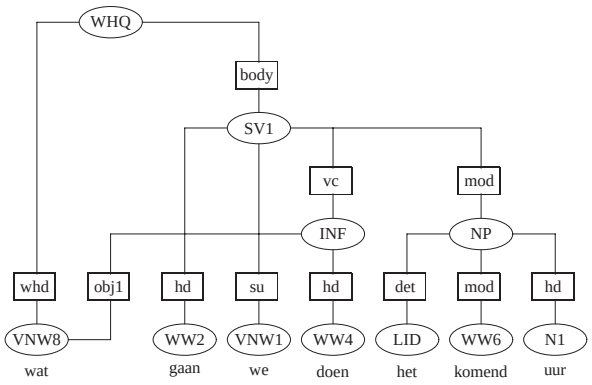


Figure 1: A CGN annotation graph

that the 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 multiple and discontinuous dependencies, the lexical type assignments will not always be directly compatible with surface constituency and word order. The second subtask then consists in calibrating the lexical type assignments in such a way that one has controlled access to the structural reasoning component of the grammar.

Our TLG grammar extraction algorithm for the CGN tree bank is parameterized in a number of ways.

Node labels Which syntactic category labels does one retain as atomic types for the resulting TLG?

Edge labels How does one translate the dependency information into the TLG function/argument/modifier articulation?

Thematic hierarchy How does one fix the canonical order of complements within a dependency domain in terms of the degree of coherence with the head?

Head position For the various clausal types, one can determine the directional orientation of the head with respect to its complements.

Licensing features In TLG (as in Minimalist Grammars), structural inferences have to be explicitly licensed by control features. Depending on their position in the type structure, one obtains (the deductive counterpart of) head movement or phrasal movement.

Below we report on the options for setting these parameters. In §4., we focus on the proof nets corresponding to lexical type assignments; in §5., we discuss the options for the structural module.

4. Obtaining the lexical proof nets

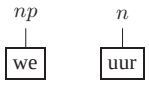
Our running example for the sections that follow is given in Figure 1. Its word-by-word translation would be 'What shall we do the next hour?' The annotation graph, with its crossing branches and multiple dependencies in the case of the question word 'wat', exhibits the typical features our TLG will have to deal with.

4.1. Basic Entries

When translating the basic entries, the issue of granularity surfaces. The tags for some words differ only in their morphological information. For example, the syntactic category VNW for 'voornaamwoord' (pronoun) has 19, in the CGN annotation standard, 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 or do we want to keep them mostly the same? For the moment, we take this last option, and we translate the syntactic categories of our example into nonterminal categories as follows.

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

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

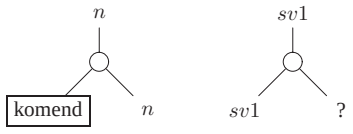


If a more fine-grained approach turns out to be desirable, we can use the built-in TLG facilities for expressing subtyping patterns. A direct implementation would be to use the part-of-speech tags as mode indices on unary connectives. One could then type 'we' as $\diamondsuit_{vnw1}\square_{vnw1}np$, from which one can *derive* type np.

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.

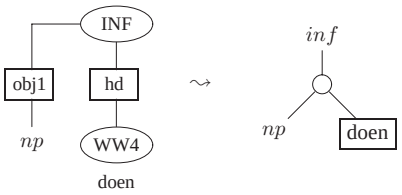
The n modifier is lexically anchored and is in fact the same lexical graph used for noun modifiers in the example lexicon in Example 1. The translation for the $sv1$ modifier is still partial, since it depends on the translation for the functor of the NP domain.



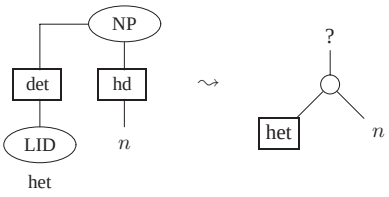
4.3. Functors

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. Below we illustrate the first option.

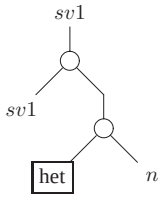
The functor 'doen' (to do) selects a direct object np to its left to produce an inf category. This is coded by the following lexical graph.



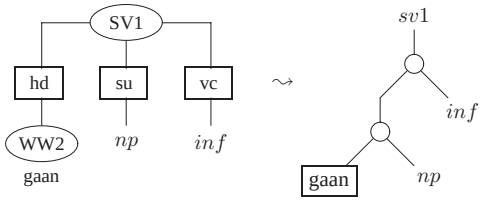
The functor 'het' (the) selects a noun to its right to produce the translation of the np category.



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.



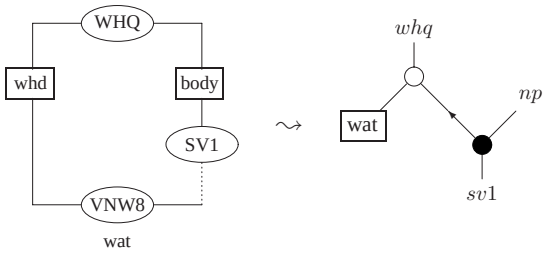
The auxiliary verb 'gaan' (go) selects for a subject np and an infinitival complement inf, which is translated as follows.



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 below for convenience, will be converted as follows, indicating 'wat' (what) produces an expression of type whq if it finds an expression of type $sv1$, in which 'wat' will function as an np, to its right.

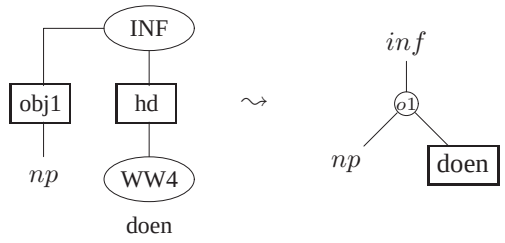


The introduction of an auxiliary constructor in the lexical graph commits us to contract this constructor and doing this may require the use of structural conversions. We will return to the topic of structural conversions §5., where they allow us to derive discontinuous constituents.

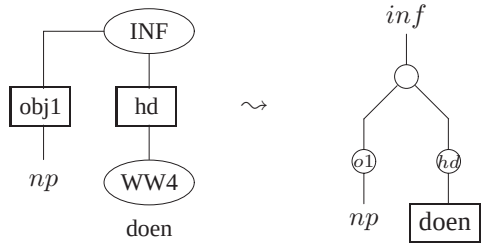
4.5. Edge Labels

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 $np\backslash_{\text{obj1}}inf$, 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 $np\backslash_{\text{r(obj1)}}inf$.



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 2 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.

```
BEGIN
FOR EVERY component c of C
  LOOK UP the formula F corresponding
    to 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\_{l(e)}F)
      ELSE IF d is a complement
        LOOK UP the formula D corresponding
          to the vertex label d
        TRANSL(d,f1\_{l(e1)} ... fi\_{l(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\_{l(e)} ... fi\_{l(e)} F/_{r(e)} fj ... /_{r(e)} 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 2: The translation algorithm

5. Discontinuity 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 saw 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 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.

5.1. The structural package

We are currently experimenting with the structural package below (from (Moortgat, 2001)) 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.

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

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

The postulates can be read in two directions. In the *Output* $\dashv$ *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* $\vdash$ *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, 2001), 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 : $\Diamond_{m}\Box_{m}(\Diamond_{m}\Box_{m}(s1\backslash s1)/\Diamond_{m}\Box_{m}(s1\backslash s1))$
uur : np
wat : $\Diamond_{whd}\Box_{whd}(whq/_{l(body)}(\Diamond_{se}\Box_{se}np\backslash_{r(obj1)}s1))$
we : np

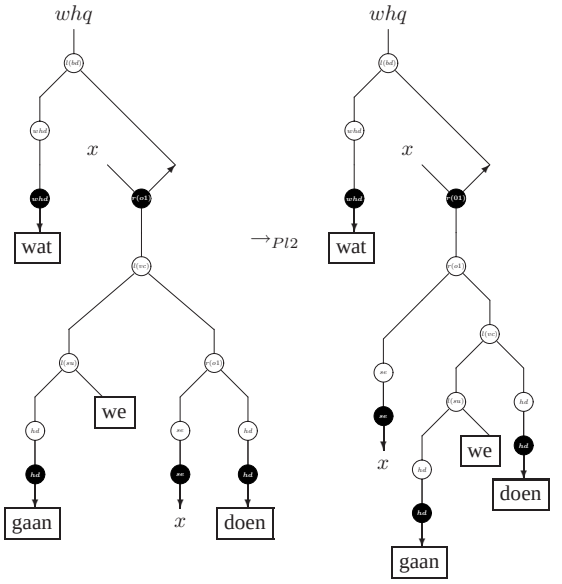


Figure 3: The application of the $Pl2$ conversion

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 3 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 $Pl2$.

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

After the $Pl2$ structural rewriting, the unary and binary redexes are all in the right configuration for contraction, as shown in Figure 3 on the right. The resulting tree is displayed in Figure 4.

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 *m* feature for the sentential modifier 'het komend uur' ('the next hour') will be active too, if we want to derive our running example

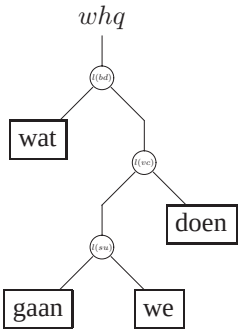


Figure 4: The resulting tree

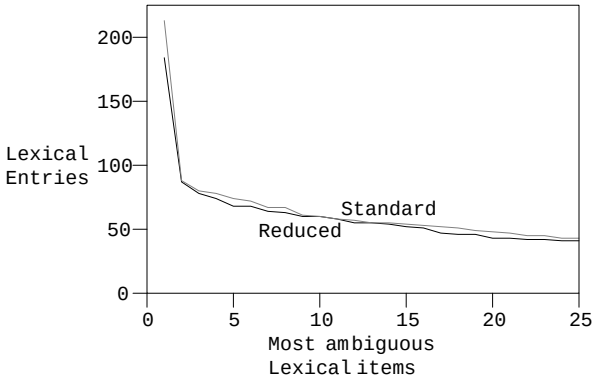


Figure 5: Reduction of the size of the lexicon for the most ambiguous lexical entries

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).

5.3. Lexical Reduction

As some preliminary test data we have extended the translation function of Figure 4.5. to produce modally decorated lexical entries and eliminated the entries which have become derivable from other lexical entries for a cross-section of 50.000 words of CGN Release 5. Figure 5 plots the data for the 25 words with the most lexical graphs assigned to them.

Table 1 lists the individual number of lexical entries in the inital and the reduced lexicon. As can be seen from these figures, lexical ambiguity is still a serious problem even in the reduced case. We are currently investigating ways of reducing the size of the lexicon even further and ways akin to suppertagging (Joshi and Srinivas, 1994) for reducing the complexity of lexical lookup.

6. Concluding Remarks

We have shown that the theory neutral annotation format used by CGN contains enough information to extract a type logical grammar from it. The translation we have proposed is parametric in a number of ways, which makes it possible to study the trade-offs between storage and computation, optimization of the TLG lexicon (reduction of lex-

Initial Lexicon		Reduced Lexicon	
is	213	is	184
en	88	en	85
van	80	van	74
dat	78	dat	71
zijn	74	in	68
in	72	zijn	67
niet	67	niet	63
maar	67	maar	63
als	61	als	60
de	60	de	59
wat	58	wat	58
een	57	met	55
om	55	om	54
met	55	een	54
op	54	op	52
voor	53	voor	47
ook	52	ook	46
het	51	nog	46
nog	49	het	46
dan	48	zo	43
je	47	of	43
zo	45	dan	42
wel	45	wel	41
was	43	die	38
of	43	je	37

Table 1: Initial and Reduced Lexicon

ical ambiguity, formula complexity) and parsing complexity (constraints on structural rules). At the time of writing, the fifth CGN release has become available, with a total of 210.000 words of syntactically annotated spontaneous speech. On the basis of this material, we are currently evaluating different structural modules and parameter settings for the extraction of a type logical lexicon. The reader is refered to the Utrecht CGN page (http://cgn.let.uu.nl) for the computational tools described in this paper, and for some numerical data obtained by running these algorithms on the Release 5 material.

7. References

Aravind Joshi and Bangalore Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 17th International Conference on Computational Linguistics*, Kyoto.

Michael Moortgat and Richard Moot. 2001. CGN to Grail: Extracting a type-logical lexicon from the CGN annotation. In Walter Daelemans, editor, *Proceedings of CLIN 2000*.

Michael Moortgat. 1997. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier/MIT Press.

Michael Moortgat. 2001. Structural equations in language learning. In Philippe de Groote, Glyn Morrill, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer.

Richard Moot and Quintijn Puite. 2002. Proof nets for the multimodal Lambek calculus. In Wojciech Buszkowski and Michael Moortgat, editors, *Studia Logica*. Kluwer Academic Publishers. To appear.

Richard Moot. 2002. *Proof Nets for Linguistic Analysis*. Ph.D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Ton van der Wouden, Heleen Hoekstra, Michael Moortgat, Bram Renmans, and Ineke Schuurman. 2002. Syntactic Analysis in the Spoken Dutch Corpus (CGN). Proceedings LREC 2002.