Extraction of Type-Logical Supertags from the Spoken Dutch Corpus*

Richard Moot

LaBRI-CNRS & INRIA Futurs
Domaine Universitaire
351, Cours de la Libération
33405 Talence, France
Richard.Moot@labri.fr

1 Introduction

The Spoken Dutch Corpus assigns 1 million of its 9 million total words a syntactic annotation in the form of dependency graphs. We will look at strategies for automatically extracting a lexicon of type-logical supertags from these dependency graphs and investigate how different levels of lexical detail affect the size of the resulting lexicon as well as the performance with respect to supertag disambiguation.

2 Type-Logical Grammars

Combinatory categorial grammars and type-logical grammars extend the simple but very limited **AB** grammars (Ajdukiewicz 1935, Bar-Hillel 1964) in different ways. Whereas CCGs (Steedman 2001) choose to add combinators allowing types to combine in more flexible ways, type-logical grammars, as pioneered by Lambek (1958) and further developed by — among many others — Morrill (1994), Moortgat & Oehrle (1994) choose to extend the system to a full logic. That is, they add to the **AB** calculus, which contains only rules telling us how to *use* complex formulas, the symmetric rules of *proof* which allow us to show we can derive a complex formula from an expression.

My introduction to type-logical grammars in this section will be necessarily brief, but I hope it will be enough for the reader to understand the applications in the following sections.

*This is an early version of the book chapter of the same title which has appeared in 'Supertagging: Using Complex Lexical Descriptions in Natural Language Processing' (2010), a volume edited by Srinivas Bangalore and Aravind Joshi and which is available with MIT press.

't $\vdash np/n$	the
eten $\vdash n$	food
is $\vdash (np\backslash s)/(n/n)$	is
koud $\vdash n/n$	cold

Table 1: Example lexicon

2.1 The non-associative Lambek calculus

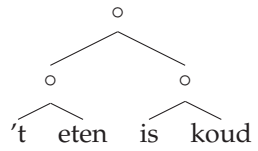
We will start by looking at the non-associative Lambek calculus, **NL** (Lambek 1961). Formulas in this calculus are either atomic formulas, from a fixed set determined by the grammar — for example, we typically find np for noun phrase, n for common noun and s for sentence — or complex formulas. If A and B are formulas, then

- $B\backslash A$ (B under A) is a formula looking to its left for a B formula to give an A as a result; an example would be $np\backslash s$ for an intransitive verb, which looks for a subject np to its left to form a sentence,
- A/B (A over B) is a formula looking to its right for a B formula to give an A as a result; an example would be np/n for a determiner, which looks for a common noun to its right to form a noun phrase or $(np\backslash s)/np$ for a transitive verb, which looks to the right for an object np to form an intransitive verb.
- finally, we have $A \bullet B$ (A and B) which is simply an expression of type A next to an expression of type B.

Note that we follow Lambek's notation where the result category is always above the slash and the argument below it.

A *statement* is a pair of the form $X \vdash A$, where X is a non-empty, binary branching tree with words as its leaves and where A is a formula. It states that we have shown X to be an expression of type A.

To save space, we will usually write the tree X in flat notation, using an infix operator '$\circ$'. In this notation, the tree



will be written as $(\textit{'t} \circ \textit{eten}) \circ (\textit{is} \circ \textit{koud})$.

A *lexicon* is a set of statements which assign formulas to single words. A Dutch example, with English translations for the words, is given is Table 1.

$$\frac{\begin{array}{c}\mathrm{x} \vdash A \quad \mathrm{y} \vdash B \\ \vdots \\ X \vdash A \bullet B \quad Z[\mathrm{x} \circ \mathrm{y}] \vdash C\end{array}}{Z[X] \vdash C}\ [\bullet E] \qquad \frac{X \vdash A \quad Y \vdash B}{X \circ Y \vdash A \bullet B}\ [\bullet I]$$

$$\frac{X \vdash A/B \quad Y \vdash B}{X \circ Y \vdash A}\ [/E] \qquad \frac{\begin{array}{c}\mathrm{x} \vdash B \\ \vdots \\ X \circ \mathrm{x} \vdash A\end{array}}{X \vdash A/B}\ [/I]$$

$$\frac{Y \vdash B \quad X \vdash B\backslash A}{Y \circ X \vdash A}\ [\backslash E] \qquad \frac{\begin{array}{c}\mathrm{x} \vdash B \\ \vdots \\ \mathrm{x} \circ X \vdash A\end{array}}{X \vdash B\backslash A}\ [\backslash I]$$

Table 2: Natural deduction rules for the non-associative Lambek calculus **NL**

$$\cfrac{\cfrac{X \vdash B \quad \mathrm{x} \vdash B\backslash A}{X \circ \mathrm{x} \vdash A}\ [\backslash E]}{X \vdash A/(B\backslash A)}\ [/I]$$

Figure 1: General derivability of lifting

The rules of **NL**, which allow us to combine these lexical statements into derivations, are given in Table 2.

A simple consequence of these rules is that type-lifting is now derivable: if X is an expression of type B it is necessarily also an expression of type $A/(B\backslash A)$. Figure 1 shows a proof of this.

2.2 Multimodal extensions

Because **NL** generates only context free languages, it is inadequate as a formalism for linguistic analysis. In this section we will look at some refinements of the calculus which allow it to expand its expressiveness without losing its essential logical nature. Moortgat (1997) gives a modern and more detailed description of these extensions.

Unary modalities

As a first extension, we no longer require the trees on the left-hand side of a statement to be binary branching. If we add the possibility of unary branching

$$\frac{\begin{array}{c}\mathrm{x} \vdash A \\ \vdots \\ X \vdash \Diamond_i A \quad Z[\langle \mathrm{x} \rangle^i] \vdash C\end{array}}{Z[X] \vdash C}\ [\Diamond E] \qquad \frac{X \vdash A}{\langle X \rangle^i \vdash \Diamond_i A}\ [\Diamond I]$$

$$\frac{X \vdash \Box_i^{\downarrow} A}{\langle X \rangle^i \vdash A}\ [\Box^{\downarrow} E] \qquad \frac{\langle X \rangle^i \vdash A}{X \vdash \Box_i^{\downarrow} A}\ [\Box^{\downarrow} I]$$

Table 3: Natural deduction rules for the unary modalities

as well is different *modes* i of unary branching, indicated on the flat term as $\langle X \rangle^i$, we are led — by analogy to the binary rules — to the rules shown in Table 3. Kurtonina & Moortgat (1997) provide a good introduction to these unary modalities and give several interesting applications.

One of the typical applications of the unary modalities is to use them to implement linguistic features. For example $\langle X \rangle^n$ could mean 'I'm an X constituent but I carry nominative case', whereas $\langle X \rangle^a$ could mean 'I'm an X constituent, but I carry accusative case'. Note that a and n are simply used as mnemonics and we might just as well have use more abstract modes like 1 and 2 two distinguish between the two cases.

Let's look at the rules in Table 3 with the intended implementation of features in mind. The $[\Box^{\downarrow} E]$ rule states that if structure X is of type $\Box_i^{\downarrow} A$ then $\langle X \rangle^i$ is of type A. The $[\Box^{\downarrow} E]$ rule *adds* feature information to a structure. Inversely, the $[\Box^{\downarrow} I]$ rule verifies if the antecedent structure has the proper form $\langle X \rangle^i$ and then *removes* this feature information.

The $[\Diamond I]$ rule states that if X is of type A then $\langle X \rangle^i$ is of type $\Diamond_i A$. Like the $[\Box^{\downarrow} E]$ rule, we add feature information to the previous structure. The final rule is perhaps the most difficult to explain; the $[\Diamond I]$ rule tell us what to do when we have derived a structure X to be of type $\Diamond_i A$. In order to remove this diamond formula, we start a subderivation using a structure x of type A demanding at the end of this subderivation that the variable x has the unary brackets $\langle\ \rangle^i$ around it. Finally, we continue the derivation replacing the structure $\langle \mathrm{x} \rangle^i$ by X, ie. the structure we had computed for $\Diamond_i A$ before. In other words, we verify that an A constituent has the correct feature information, then remove this feature information and continue with a $\Diamond_i A$ constituent. This 'check then remove' behavior is similar to the $[\Box^{\downarrow} I]$ rule.

Two useful derivability patterns are shown in Figure 2. On the left we show that for every mode i and structure X of type A, this same structure is also of type $\Diamond_i \Box_i^{\downarrow} A$. Conversely, as shown on the right of the figure, if X is of type $\Diamond_i \Box_i^{\downarrow} A$ then it is also of type A.

$$\frac{\dfrac{X \vdash A}{\langle X \rangle^i \vdash \Diamond_i A}\,[\Diamond I]}{X \vdash \Box_i^{\downarrow} \Diamond_i A}\,[\Box^{\downarrow} I] \qquad \frac{X \vdash \Diamond_i \Box_i^{\downarrow} A \quad \dfrac{\mathrm{x} \vdash \Box_i^{\downarrow} A}{\langle \mathrm{x} \rangle^i \vdash A}\,[\Box^{\downarrow} E]}{X \vdash A}\,[\Diamond E]$$

Figure 2: Derivability patterns for the unary modalities

hij $\vdash \Box_n^{\downarrow} \Diamond_n np$	he
hem $\vdash \Box_a^{\downarrow} \Diamond_a np$	him
Vincent $\vdash np$	Vincent
Peru $\vdash np$	Peru
slaapt $\vdash \Box_n^{\downarrow} \Diamond_n np \backslash s$	sleeps
bezoekt $\vdash (\Box_n^{\downarrow} \Diamond_n np \backslash s) / \Box_a^{\downarrow} \Diamond_a np$	visits

Table 4: Lexicon with basic case information

Table 4 shows how we can exploit these patterns when adding basic case information to the lexicon. Heylen (1999) gives a much more detailed treatment of feature information for categorial grammars. The personal pronouns, like 'hij' (he) and 'hem' (him) are specified in the lexicon for nominative and accusative case respectively.

Since Dutch proper nouns don't change their form to indicate their case, we would prefer to assign them only a single form which is underspecified for case. But this is exactly what the standard np type allows us to do, given that we have just see that any structure of type np is also a structure of type $\Box_n^{\downarrow} \Diamond_n np$ and a structure of type $\Box_a^{\downarrow} \Diamond_a np$. The resulting grammar correctly predicts that 'hij bezoekt hem' is a grammatical sentence but that '$*$hem bezoekt Peru' is not.

Structural Rules

By themselves, the unary modalities don't extend the generative capacity of type-logical grammars. A second extension, in the form of structural rules, allows us to do this. Adding structural rules to **NL** is not a new idea. For example, adding the structural rule of associativity to **NL** gives us an alternative formulation of the associative Lambek calculus **L**.

The advantage of a multimodal calculus is that rather than having structural rules apply *globally* we can 'anchor' them to specific modes, which in turn are obtained from the lexical types.

Consider for example a permutation mode p, which allows us to move constituents which have been lexically specified as being of type $\Diamond_p \Box_p^{\downarrow} A$. As we have seen in Figure 2 on the right, such a constituent can play the role of an A when needed. However, the structural rules shown in Figure 3 show us how we can move a constituent marked as $\langle X \rangle^p$ from one right branch in a

$$\frac{Z[X_1 \circ (X_2 \circ \langle X_3 \rangle^p) \vdash C}{Z[(X_1 \circ X_2) \circ \langle X_3 \rangle^p] \vdash C}\ [MA]$$

$$\frac{Z[(X_1 \circ \langle X_3 \rangle^p) \circ X_2] \vdash C}{Z[(X_1 \circ X_2) \circ \langle X_3 \rangle^p] \vdash C}\ [MC]$$

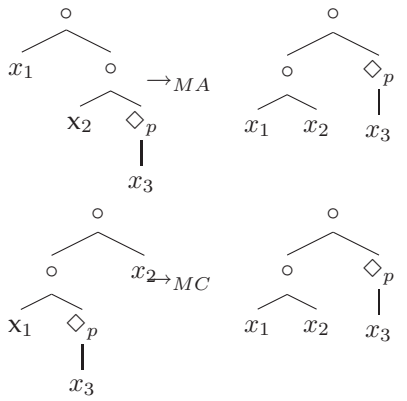


Figure 3: The structural rules of mixed associativity and mixed commutativity in flat representation and as tree rewrites

tree to another. The rule MA, for 'mixed associativity', takes an embedded x_3, which is a sister of the x_2 node and moves it upwards to be a sister of the node one level up, regrouping x_1 and x_2 together. It is an associativity rule since it only changes the grouping of the nodes but not their order. The rule MC, for 'mixed commutativity' *does* change the order. Before application of the rule, the x_3 node is between the x_1 and x_2 nodes. After the rule, x_1 and x_2 are adjacent.

Observe that as long as the top of the binary tree hasn't been reached, ex-

wie $\vdash wh/(s/\Diamond_p \Box_p^{\downarrow} np)$	who
leest $\vdash (s/np)/np$	reads (interrogative)
Proust $\vdash np$	Proust
Tsjechov $\vdash np$	Chekov

Table 5: Lexicon for the treatment of *wh* questions in Dutch

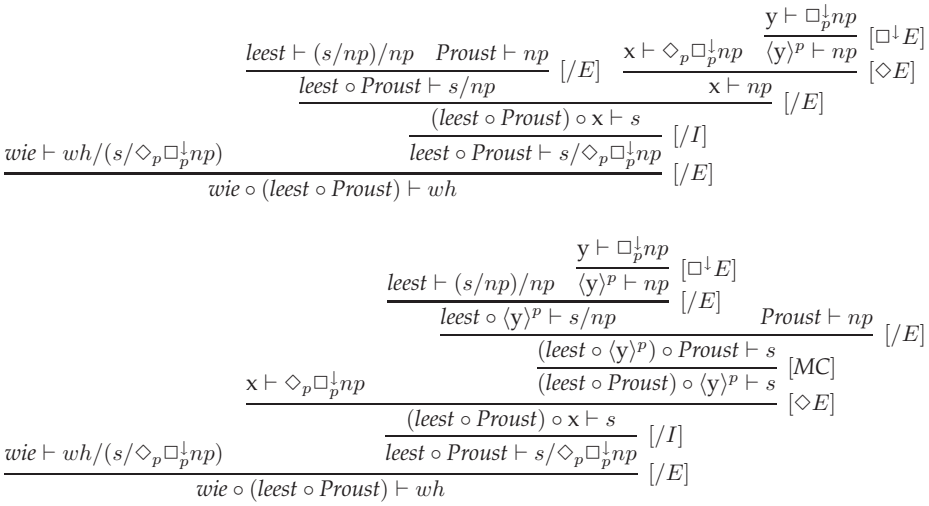


Figure 4: Two derivations for the ambiguous sentence 'wie leest Proust'

actly one of these two rules will apply, depending on whether the parent node of $\langle x_3 \rangle^p$ is on a left or a right branch.

We can use these structural rules to given an account of extraction in type-logical grammar. Let's look at the following examples.

(1) *leest* *Proust* *Tsjechov* *?*
reads Proust Tsjechov ?
'does Proust read Chekov?'

(2) *wie* *leest* *Proust* *?*
who reads Proust ?
ambiguous between 'who reads Proust?' and 'who does Proust read?'

If we were to assign 'wie' the formula $wh/(s/np)$ in our lexicon, only the second reading would be available to us. However, if we would assign it the formula $wh/(s/\Diamond\Box^{\downarrow}np)$ it would say: I'm looking to my right for a sentence which is missing an np anywhere on a right branch to give a wh question. Figure 4 shows how this allows us to find both readings for this sentence.

The first reading simply uses the fact that a structure of type $\Diamond_p\Box^{\downarrow}_p np$ can be used as an np, after which we simply have an **NL** derivation. The second reading is more interesting, as it requires us to use the structural rule of mixed commutativity. The $[\Diamond E]$ and $[\Box^{\downarrow}E]$ rules introduce a constituent $\langle \mathrm{y} \rangle^p$, our moving np. Combining this np with *leest* and *Proust* gives a configuration to which the mixed commutativity rule can be applied. We have the bottom left

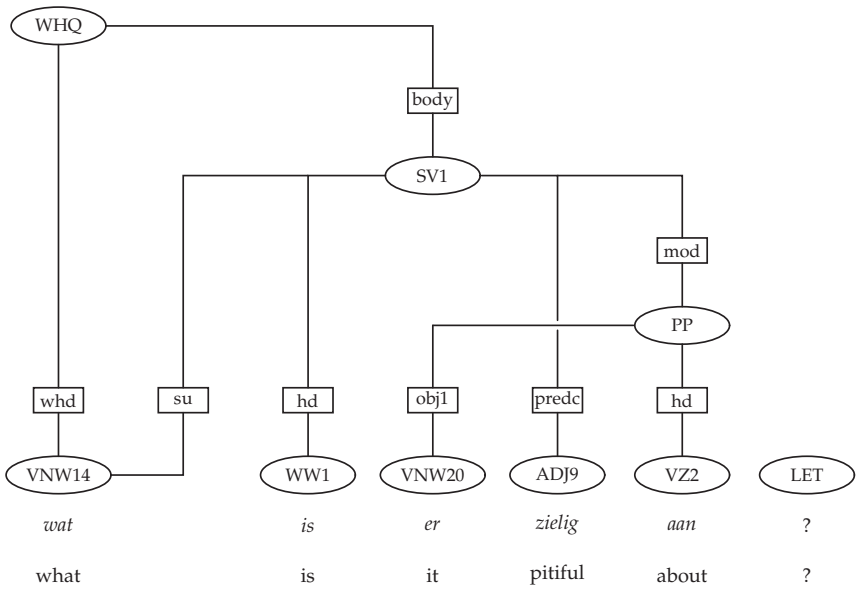


Figure 5: What is pitiful about it?

tree of Figure 3 with $x_1 = \textit{leest}$, $x_3 = \text{y}$ and $x_2 = \textit{Proust}$. After this structural rule we can finish the $[\Diamond E]$ rule, replacing $\langle \text{y} \rangle^p$ by x, then conclude our derivation as we would in **NL**.

Adding unary modalities and structural rules gives us a flexible logic, able to handle the harsh reality of linguistic phenomena, to which we will now turn.

3 Treebank Extraction

In this section, I will show how to generate a type-logical treebank from the syntactic annotation files of the Spoken Dutch Corpus and some methods for reducing the size of the generated lexicon.

3.1 Syntactic Annotation for the Spoken Dutch Corpus

The Spoken Dutch Corpus ('Corpus Gesproken Nederlands' or CGN) contains 9 million words of contemporary spoken Dutch with various forms of linguistic annotation. Orthographic transcription and part-of-speech tagging have been provided for all 9 million words. A core corpus of 1 million word has, in addition, been provided with a syntactic annotation in the form of dependency graphs.

For the CGN syntactic annotation, the annotation tools developed for the German NEGRA Corpus (Brants 1999) have been used to semi-automatically produce syntactic annotation graphs of the form shown in Figure 5 on the preceding page.

I will briefly discuss some properties of these annotation graphs. More details on the annotation format and philosophy can be found in (Hoekstra, Moortgat, Renmans, Schuurman & van der Wouden 2002).

The annotation graphs are directed, acyclic graphs, where every leaf is labeled with a part-of-speech tag (like WW1 for a singular, inflected verb) and all other vertices are labeled with a grammatical constituent (like SV1 for a verb-initial sentence). Edges are labeled with a dependency relations (like *hd* for head and *obj1* for a direct object).

The POS tags used for the syntactic annotation are a simplification of the morphologically richer tags which have been used for POS tagging the Spoken Dutch Corpus. The WW1 tag for the verb, for example is a simplification of the T301 tag, which also indicates the verb is in present tense and does not end with the 't' suffix for verb inflection.

Some other properties are:

- Annotation structures are flat. A new node is only introduced when necessitated by a lexical head. This means, for example, that we have no separate *vp* nodes.
- We can have multiple dependencies, sometimes called 'secondary edges'. These are used both for the treatment of ellipsis and for long-distance dependencies. In the example, we have a long-distance dependency, where VNW14 (wat) is both the head of the *wh* phrase and the subject of the verb-initial sentence. The annotation
- Constituents can be discontinuous. For example 'er aan' is a PP in the example sentence, even though the ADJ9 (zielig) is positioned between these two words.
- Annotation graphs are allowed to be disconnected. In our example the LET constituent is an isolated vertex.

3.2 Preprocessing

Given that we are using a spoken corpus, we have to deal with a number of artifacts which would not normally be present in written language. Words indicating speaker hesitation like 'uh', for example, occur frequently and are almost never assigned a grammatical function. They appear as isolated vertices in the annotation graph. Some annotators choose to give them a role in incomplete phrases, but this is relatively rare (192 out of 23,289 occurrences). The same can be said for interpunction marks, which, as shown in Figure 5 on the previous page, are generally annotated as an isolated LET vertex. Only

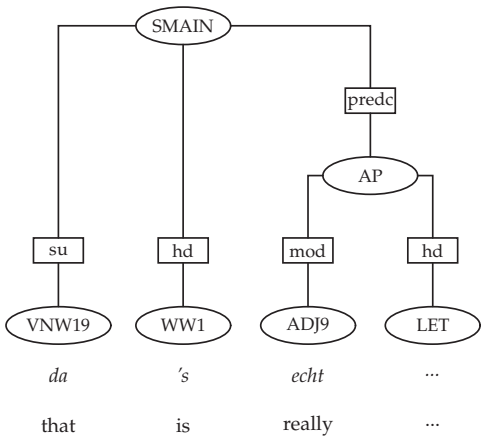


Figure 6: Incomplete phrase with grammatical role for an interpunction symbol

634 out of 114,221 occurrences are assigned a grammatical function. Figure 6 shows an example.

To prevent these 'easy' word categories from giving a too flattering impression of the performance of the supertagger, I cleaned up the data, filtering out all isolated vertices, including hesitation marks and interpunction symbols but also partial repeats (see 3, where 'het/'t is' occurs twice) and self-corrections, as in 4 where the speaker corrects the gender of the article he is using.

(3) *ja het is 't is nog wel erger geweest ...*
yes it is it is still even worse been ...
'yes, still it has been even worse ...'

(4) *... dat 't de huidige regelgeving ...*
... that the-NEU the-MSC/FEM current legislation ...
... 'that the current legislation'

In cases like these, the first occurrences of the repeated or corrected words are usually not assigned a grammatical role.

Removing all these isolated vertices gives a filtered corpus contains 87,404 sentences (out of 114,801, most of the removed sentences being single-word utterances like 'yes') and 794,872 words (out of 1,002,098).

We will look a way of filtering the corpus without looking at the syntactic annotation in Section 4.4.

3.3 Extraction Algorithm

The algorithm used for extracting a treebank is essentially the one proposed in (Moortgat & Moot 2002), which is parametric for three functions:

1. a function from vertex labels (or $\langle$*vertex label*, *edge label*$\rangle$ pairs) to formulas,
2. a function identifying the head of every grammatical constituent,
3. a function identifying the modifiers of every grammatical constituent.

We will discuss different functions assigning formulas to vertex labels, since this has a great influence on the number of different lexical entries we extract. The other two functions have been kept constant.

The current implementation identifies the functor from a set of daughters by their edge labels using a list in order of preference. For most vertex labels this will just be *hd*, but, because the algorithm requires finding a head for every syntactic category, there are typically some strange items further down the list to make sure we can find a head even for incomplete utterances.

The function identifying modifiers marks only constituents with edge label *mod* as modifier for most syntactic categories. Siblings of the head which are not modifiers will be its arguments.

The algorithm operates by performing a basic depth-first search from the different root nodes of a corpus graph assigning a *primary* formula f to every node in the graph as well as a list $g_1, \ldots, g_n$ of *secondary* formulas. The list of secondary formulas is necessary for nodes with multiple parents and will contain a formula for each additional parent of the node, ie. every node will have one formula for every role if fulfills in the syntactic structure. The corresponding formula is $(\ldots(f \bullet \Diamond_p \Box_p^{\downarrow} g_1) \ldots \bullet \Diamond_p \Box_p^{\downarrow} g_n)$. For most nodes, however, this list will be empty, and the primary formula f will directly correspond to the assigned type.

To divide an annotation graph into lexical formulas for all words in it, we begin by finding all root nodes in the annotation graph and looking up the corresponding formula f in the table. We determine for each of the children if they are the unique head, a modifier or an argument. We descend the modifiers assigning their vertex f/f or $f\backslash f$, depending on their position with respect to the head. For the arguments, we look up the corresponding formulas $a_1, \ldots, a_n$ in the table and assign it to their vertex. Finally, the head child will be assigned the complex formula

$$(a_i\backslash \ldots (a_1\backslash((f/a_n)\ldots/a_{i+1})))$$

which first selects the arguments $a_{i+1}, \ldots, a_n$ to the right of it then the arguments $a_1, \ldots, a_i$, to its left. When we arrive at a node via a secondary edge we append the formula to the list of secondary formulas.

In every case, after assigning a type to a vertex we will descend recursively until we reach the leaves, in which case the formula will be added to the lexicon for that word.

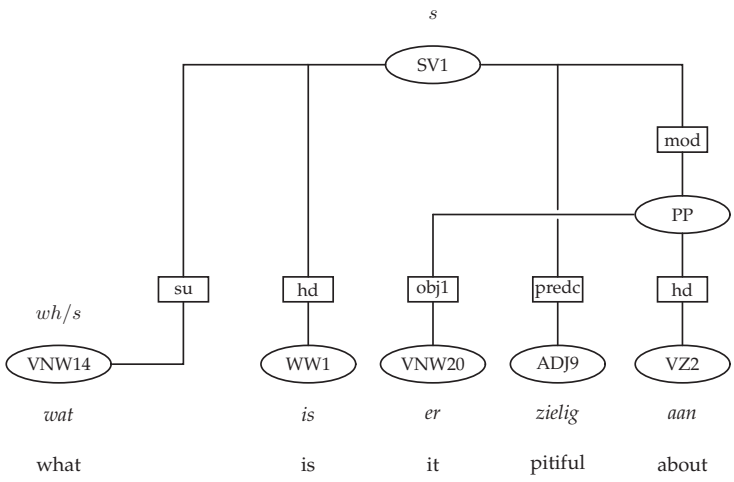


Figure 7: Lexicon extraction: removing the WHQ node

The operation of the algorithm is perhaps most clearly demonstrated by an example. If our vertex label to formula mapping assigns wh to WHQ and s to SV1 and we make, like the CGN annotation, the choice of *whd* as head of a *wh* question then, returning to Figure 5 on page 8, we are compelled to assign the formula s to the SV1 and wh/s to the VNW14 node, as shown in Figure 7.

Now it's the turn of the SV1 node to be split. It has four children, the WW1, reached by the *hd* label, being the head and the PP, reached by the *mod* label, a modifier. Because it is to the right of the head, it will be assigned the modifier category $s\backslash s$. The other two children are arguments and we look up the formulas which correspond to them: np for VNW14 and ap for ADJ9. Therefore, the formula assigned to the verb will be $(s/ap)/np$.

Because we now arrive at the relative pronoun via a secondary edge, we add an np formula to the list of secondary formulas. As shown in the figure, this corresponds to assigning it the formula $(wh/s) \bullet \Diamond\Box^{\downarrow} np$.

There is only one nonterminal left to be split, which is the PP node. We have the preposition VZ2 as its head and the pronoun VNW20 as its argument. Again, the type for the argument is np, so the type for the head will be $np\backslash(s\backslash s)$.

The final extracted lexicon is shown in Figure 9 on the next page. We note that if we specialize the product type for 'wat', we will produce the type $wh/(s/\Diamond\Box^{\downarrow} np)$ which is identical to the one proposed for the manually generated lexicon of Table 5 on page 6.

The algorithm for corpus extraction only exploits the fact that we have a dependency-based annotation. Given the three functions we discussed at the beginning of this section, we could directly use the current algorithm to extract

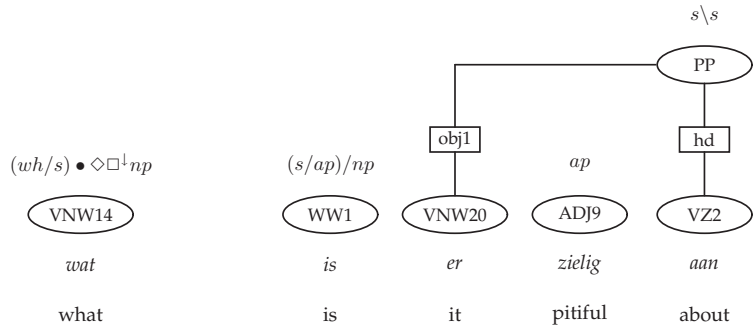


Figure 8: Lexicon extraction: removing the SV1 node

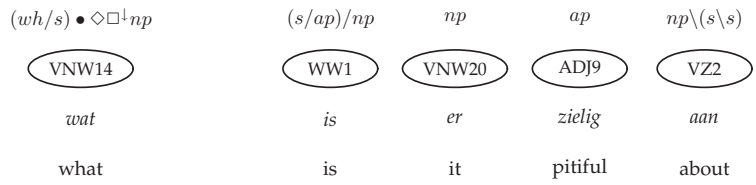


Figure 9: Final lexicon

a type-logical grammar from other corpora employing dependency structures for their annotation format, such as the French TALANA treebank (Abeillé, Clément & Kinyon 2000), the German TIGER treebank (Brants, Dipper, Hansen, Lezius & Smith 2002) and the Prague dependency treebank (Hajic 1998).

3.4 Refinement

Some of the vertex labels of the Spoken Dutch Corpus, like DU (discourse unit), CONJ (conjunction), LIST and MWU (merged word unit, a category assigned to multi-word names and fixed expressions) are not really grammatical categories.

An example of a discourse unit is shown in Figure 10 on the next page. Often it is just an ordinary sentence introduced by a tag like 'yes' or (as in the example) 'no' or an element linking it to the previous sentence like 'and'. Examples of the other categories are shown below.

(5) *wij hadden nog* [*meetkunde en algebra*]$_{CONJ}$
we had still [geometry and algebra]$_{CONJ}$
'we still had geometry and algebra'

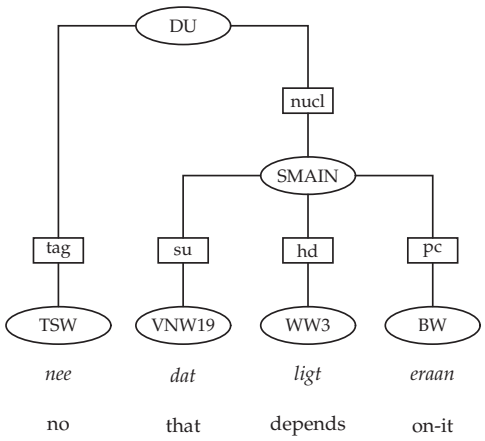


Figure 10: Discourse unit

(6) *koffie* [*geen melk geen suiker*]$_{LIST}$
coffee [no milk no sugar]$_{LIST}$
'coffee, no milk, no sugar'

(7) [*boulimia nervosa*]$_{MWU}$ *heet 't*
[boulimia nervosa]$_{MWU}$ called it
'it is called boulimia nervosa'

Rather than using a type *conj* or *list* in the lexicon, we give a conjunction the type of it first conjoint and a list the type of its first list item. A discourse unit is assigned the type of its nucleus, if there is one, and the type of its first discourse element otherwise. We simply assign np to merged-word units.

4 Supertagging

Now that we have a generic method of extracting supertags from the Spoken Dutch Corpus, it is time to extract different grammars and evaluate the size of the lexicon as well as the performance of supertag disambiguation for these lexicons.

4.1 The Extracted Grammars

The most basic way of applying the algorithm of the previous section is simply to keep syntactic categories of the CGN annotation (22 in total when we exclude DU, CONJ, LIST and MWU) and map the part-of-speech tags of the

leaves to these categories (adjectives to AP, pronouns to NP, etcetera). This leaves us with a rather large lexicon, the *basic* treebank, containing 6,817 different formulas.

To reduce the size of the lexicon, we have merged a number of these formulas: no longer distinguishing between the different sentence types SMAIN (declarative sentence, which is verb second), SV1 (interrogative sentence, with the verb first) and SSUB (subordinate clause, with the verb last) and mapping them all simply to s. Though we can still determine the sentence type from the directionality of the implications, this will make a big difference for the sentence modifiers, which now no longer need to distinguish between the different sentence types. Other changes include mapping AP to noun modifier. This reduces the size of the lexicon to 3,539, giving us the *compact* treebank.

Another simplification is obtained by removing external discourse unit nodes. This removes the need to analyze a discourse unit containing several different parts, which are often elliptical or only loosely related like 8, 9 or 10 below, where any decision as to what would be modifiers, arguments or head would be rather artificial.

(8) *deze* *onderaan* *hier*
this at the bottom here
'this one at the bottom here'

(9) *mama* *dronken*
mother drunk
'mother (is) drunk'

(10) *positief* *tenzij*
positive unless
'I am of positive opinion, unless ...'

The resulting *split* treebank contains 2,201 lexical entries.

For comparison, I added three other strategies to reduce the size of this treebank. While these grammars will provide much coarser linguistic descriptions, it is perhaps useful to look at the effect of the reduced lexicon size on the performance of the supertagger.

- The first collapses even more lexical categories than before, by for example replacing the *pp* category by $s\backslash s$, which causes a preposition which modifies a verb to have the same type as a preposition which is the argument of the verb, with only the verb type differing in the two cases. This is the *very compact* treebank, containing 1,962 entries.

- The second removes all multiple dependencies from the grammar. This makes the grammar a simple **AB** grammar, needing only elimination rules to find a derivation. This is useful if we are interested in parsing the supertagged sentence afterwards, given that we can use chart parsing techniques for these grammars. The *AB* treebank contains 1,761 lexical formulas.

- The third removes all directionality from the implications, making it a grammar for the Lambek-van Benthem calculus **LP**. For example, it no longer distinguishes between verbs in initial, second or final position. If we are only interested in the semantic type, however, this information would suffice. The *LP* treebank is the smallest we will consider, containing 1,137 formulas.

Other strategies for extracting type-logical grammars from structured data and reducing the size of the lexicon exist. Buszkowski & Penn (1990) propose a strategy for learning **AB** grammars from structures and using unification of type variables to reduce the lexical ambiguity. An advantage of this method is that it can be used to find the optimal way of mapping vertex labels to formulas. Unfortunately, the algorithm for finding this mapping is exponential and therefore the current choice of requiring the extraction algorithm to know the formulas corresponding to the different labels seems more practical, giving us an algorithm for extracting our grammars in time linear to the size of the corpus.

Moortgat (2001) proposes a strategy for reducing the lexical ambiguity for automatically extracted grammars by introducing modal operators and structural reasoning to replace different lexical type assignments by a single one which is at least as general the types it replaces. If we can avoid overgeneralisation, this would allow us to reduce our treebank to a corpus of a size which would ideally be comparable to the LP treebank without collapsing to full commutativity. This seems challenging to achieve, however.

4.2 Experiments and Results

All experiments have used the 72 part-of-speech tags instead of the 324 morphology tags, which contain too detailed morphology information, actually hindering performance because of the resulting data sparseness. The single exception is the SPEC (special) tag, which we split into the seven different versions used by the morphology tagger, giving us a total of 78 part-of-speech tags for our experiments. The choice to split up the special tags was made because it comprises words in different groups like: inaudible, broken off, foreign, part-of-whole (as in multi-word proper names) and background noise.

We have trained our models using a supertagger based on the Edinburgh maximum entropy tools, making a decision for the lexical formula of the words based on:

- the current word and POS tags, as well the two previous/following words and POS tags
- the previous two supertags.

Only features which were seen 5 times or more in the training data have been selected for training.

	Experiment	Formulas	Result
1	Basic	6,817	70.61%
2	Compact	3,539	72.06%
3	Split	2,201	77.13%
4	Very compact	1,962	77.83%
5	AB	1,761	77.52%
6	LP	1,137	80.50%

Table 6: Combined results of the different extraction procedures

All models were trained using 100 iterations on 69,923 sentences of the filtered corpus (635,764 words), whereas the other 17,480 sentences (159,086 words, every fifth sentence of the corpus) were kept for testing the performance.

Table 6 summarizes the results for the different corpora.

The Split corpus assigns lexical formulas which are closest to those provided by a manually generated lexicon and offers a good compromise between performance and descriptive adequacy. Compared to the Compact corpus, which differs only in that it does not split up root discourse unit nodes, there is a big improvement both in lexicon size and performance.

The AB corpus, in spite of its reduced lexicon size, actually performs worse than the Very Compact corpus, presumably because not treating the secondary edges forces us to assign unusual types to verbs.

The LP corpus has the best performance, both in lexicon size and in correct supertag assignments, though at the price of a significant loss in the information the supertags provide.

4.3 Analysis of the Extracted Lexicon

To give a better idea of the form of the extracted lexicon, I will now discuss some of the frequently occurring verb forms as well as the most frequent product types which are extracted in the case of multiple dependencies. All discussion in this section will be about the Split treebank.

Table 7 list the supertags which have been extracted over 1,000 times for finite, singular verbs, together with the percentage of the total extracted verb forms these formulas account for. So, the nine formulas shown in the table, when taken together, account for 72.09% of the total verbs forms.

The type assigned to auxiliaries $(np\backslash s)/(np\backslash s)$ is the most frequent, closely followed by the type for transitive verb with the verb occurring in second position, as it will in a non-topicalized declarative sentence. In total there are 724 different supertags assigned to these verbs, though only 396 are assigned more than once.

	Formula	Occurrences	Total Coverage
1	$(np\backslash s)/(np\backslash s)$	9,487	14.66%
2	$(np\backslash s)/np$	9,401	29.18%
3	$np\backslash s$	7,187	40.28%
4	$(s/(np\backslash s))/np$	6,019	49.58%
5	s/np	4,698	56.84%
6	$(np\backslash s)/(np/np)$	3,893	62.85%
7	$(s/np)/np$	3,050	67.57%
8	$np\backslash(np\backslash s)$	1,660	70.07%
9	$(s/(np/np))/np$	1,268	72.09%

Table 7: All supertags which have been extracted over 1,000 times for finite verbs

As can be seen from the formulas in the table, the types assigned are quite construction-specific, as they would be in an extracted LTAG grammar. For example, several frequently occurring type differ only in the order of the arguments. For intransitive verbs, we see both s/np and $np\backslash s$, and for transitive verbs we see all three possibilities: verb-initial $(s/np)/np$, verb-second $(np\backslash s)/np$ and verb-final $np\backslash(np\backslash s)$. Even the type $(s/(np\backslash s))/np$ is just the sentence-initial version of the standard auxiliary verb type. While it seems we can gain something using a unique type in these cases and derive the other possibilities using structural rules, the improvement in supertag performance is relatively small, with less than one percent improvement in the number of supertags which is correctly assigned to verbs. Apparently, the trigrams give us enough context to decide on the proper direction of the arguments of verbs and, given that this information is useful for parsing, I have decided not to collapse types which differ only in the direction of the arguments.

Table 8 lists all singular, declarative verb forms which occur more than 1,000 times, together with the number of supertags with which they have been found as well as indicating the most common one. Unsurprisingly, forms of 'be', 'have' and other auxiliaries dominate here, both in number of occurrences and in the number of assigned supertags. Some remarks on the most frequently assigned supertags: 'vind', as indicated by the type $((np\backslash s)/(np/np))/np$ is most often seen in constructions of the type 'find movies interesting', whereas the s type for 'zeg' is due to the frequently occurring interjection 'zeg maar', which corresponds roughly to 'so to say'.

Product types for long-distance dependencies are a frequent occurrence as well. Table 9 shows the three most frequent ones, with their number of occurrences, formula and (where possible) the corresponding product-less formula. Unsurprisingly, the two most frequently occurring formulas are the type of the relativizer, as extracted for words like 'die' (that) and the question type for

Word	Transl.	Freq.	# st	Most Frequent
's	is	1,919	42	$(np\backslash s)/(np/np)$
ben	am	1,474	90	$(np\backslash s)/(np\backslash s)$
denk	think	1,851	61	$(np\backslash s)/s$
had	had	2,104	112	$(np\backslash s)/(np\backslash s)$
heb	have	4,049	126	$(np\backslash s)/(np\backslash s)$
is	is	14,212	313	$(np\backslash s)/np$
kan	can	2,731	93	$(np\backslash s)/(np\backslash s)$
moet	has to	3,340	91	$(np\backslash s)/(np\backslash s)$
vind	find	1,353	82	$((np\backslash s)/(np/np))/np$
was	was	4,467	176	$(np\backslash s)/np$
weet	know	2,362	78	$(np\backslash s)/np$
wil	want	1,137	65	$(np\backslash s)/(np\backslash s)$
zeg	say	1,095	61	s
zit	sit	1,018	62	$np\backslash s$
zou	would	1,688	57	$(np\backslash s)/(np\backslash s)$

Table 8: All verb forms occurring more than 1,000 times

	Occurrences	Type	Simplified Type
1	3,672	$((np\backslash np)/s) \bullet \diamond_p \square_p^{\downarrow} np$	$(np\backslash np)/(s/\diamond_p \square_p^{\downarrow} np)$
2	3,174	$(wh/s) \bullet \diamond_p \square_p^{\downarrow} np$	$wh/(s/\diamond_p \square_p^{\downarrow} np)$
3	1,796	$np \bullet \diamond_p \square_p^{\downarrow} np$	

Table 9: Frequent long-distance dependencies

words like 'wat' (what) as we have already seen it in Section 3.3. The final type is perhaps more surprising. It is a result of the treatment of ellipsis in the Spoken Dutch Corpus.

For example, in the sentence

(11) *die ouwe krant of de nieuwe*
that old newspaper or the new
'that old newspaper or the new one'

the word 'krant' will be assigned the type $np \bullet \diamond_p \square_p^{\downarrow} np$ since it is considered to be the head noun of both conjuncts. From a type-logical point of view, this is not an entirely satisfactory solution, we would like to keep a simple np type in these cases and analyze the construction differently, for example along the lines proposed by Hendriks (1995). However, it is unclear if we can keep the lexicon extraction fully automatic when reanalyzing these constructions in the treebank.

Experiment	Result
Non-filtered, with interpunction	81.26%
Non-filtered, without interpunction	78.85%
Combined supertagger, with interpunction	81.50%
Combined supertagger, without interpunction	79.11%

Table 10: Comparison between the combined and the non-filtered supertaggers

4.4 Detecting Isolated Vertices Automatically

While we have cleaned up the training data using information from the syntactic annotation itself, it is useful to see how well we do if we perform this step using only the information we would have for the 8 million which have only part-of-speech tag information.

To verify if training with the filtered dataset has actually gained us something we performed the following final experiment. We repeated the Split extraction procedure, this time using the non-filtered corpus and divided this into a set of training data and test data. As before, every fifth sentence has been reserved for test data.

We trained three models using these data: one assigning supertags directly, one indicating only if a word is isolated or not and a final model using the filtered version of the training data. Our goal is to compare the performance of the first model against the two others applied in series. That is to say: a word correctly tagged as isolated is considered to have the correct tag, a word incorrectly tagged as isolated is considered to have an incorrect tag and a word tagged as non-isolated is considered to have the correct tag if the model trained on the filtered corpus produces the correct result for it.

For deciding whether a word is an isolated vertex in the graph or not, a simple experiment was performed, using only word and part-of-speech tag information of the current word and the two words preceding and following it. Additionally, we used information about words which are labeled as being broken off or a repeat of the current word or part-of-speech tag in the previous five words. This simply strategy received a 98.35% success rate on the test data.

Table 10 shows a comparison between the two strategies. We note that the performance on both tasks is better than the supertagging performance on the filtered corpus, even when we don't count interpunction symbols. This is because, as already noted in Section 3.2, word categories like hesitation marks 'uh' are almost always correctly tagged as isolated (in the combined experiment) or assigned the correct category based on their unigram information (in the non-filtered experiment).

We also note that the combined experiment performs slightly better than the direct method: filtering out words which do not contribute to the syntactic structure of the phrase means we get somewhat cleaner trigrams to train our

models.

Finally, these experiments show it is possible to remove the isolated nodes from the corpus automatically, without negatively affecting the performance.

4.5 Comparison

Chen & Vijay-Sjanker (2000) propose a method of automatically extracting LTAGs from the Penn treebank and investigate the effect of different extraction strategies. Their results appear to be a bit better than the results presented here, between 77.79% and 78.90% depending on the extraction strategy, even though we obtain a more compact lexicon. It stands to reason, however, that these differences are due to spoken corpora being inherently more noisy.

Semiautomatically extracted supertag sets, as used by Clark (2002), and manually crafted supertag sets, as used by Srinivas (1997), appear to fare significantly better, producing both a more compact lexicon and better performance for supertag disambiguation, which suggests there is a trade-off to be made between manual effort and performance.

5 Conclusions

We have seen how we can automatically extract a type-logical treebank from the CGN syntactic annotation. Depending on the level of detail we choose to maintain in our lexicon, the number of different formulas varies between 6.817 and 1.137, whereas the correctness of supertag disambiguation varies between 72.06 and 80.50%.

References

Abeillé, A., Clément, L. & Kinyon, A. (2000), Building a treebank for French, *in* 'Proceedings of the Second International Language Resources and Evaluation Conference', Athens, Greece.

Ajdukiewicz, K. (1935), 'Die syntaktische Konnexität', *Studies in Philosophy* **1**, 1–27.

Bar-Hillel, Y. (1964), *Language and Information. Selected Essays on their Theory and Application*, Addison-Wesley, New York.

Brants, S., Dipper, S., Hansen, S., Lezius, W. & Smith, G. (2002), The TIGER treebank, *in* 'Proceedings of the Workshop on Treebanks and Linguistic Theories', Sozopol.

Brants, T. (1999), Tagging and Parsing with Cascaded Markov Models - Automation of Corpus Annotation, PhD thesis, German Research Center for Artificial Intelligence and Saarland University, Saarbrücken, Germany.

Buszkowski, W. & Penn, G. (1990), 'Categorial grammars determined from linguistic data by unification', *Studia Logica* **49**, 431–454.

Chen, J. & Vijay-Sjanker, K. (2000), Automated extraction of TAGs from the Penn treebank, *in* 'Proceedings of the 6th International Workshop on Parsing Technologies', Trento, Italy.

Clark, S. (2002), Supertagging for combinatory categorial grammar, *in* 'Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Formalisms', Venice, pp. 19–24.

Hajic, J. (1998), Building a syntactically annotated corpus: The Prague dependency treebank, *in* 'Issues of Valency and Meaning', Karolinum, Prague, Czech Republic, pp. 106–132.

Hendriks, P. (1995), Ellipsis and multimodal categorial type logic, *in* G. Morril & R. T. Oehrle, eds, 'Proceedings of Formal Grammar 1995', Barcelona, Spain, pp. 107–122.

Heylen, D. (1999), Types and Sorts: Resource Logic for Feature Checking, PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Hoekstra, H., Moortgat, M., Renmans, B., Schuurman, I. & van der Wouden, T. (2002), Syntactic analysis in the Spoken Dutch Corpus (CGN), *in* 'Proceedings of the Third International Language Resources and Evaluation Conference', Las Palmas.

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

Lambek, J. (1958), 'The mathematics of sentence structure', *American Mathematical Monthly* **65**, 154–170.

Lambek, J. (1961), On the calculus of syntactic types, *in* R. Jacobson, ed., 'Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics', Vol. XII, American Mathematical Society, pp. 166–178.

Moortgat, M. (1997), Categorial type logics, *in* J. van Benthem & A. ter Meulen, eds, 'Handbook of Logic and Language', Elsevier/MIT Press, chapter 2, pp. 93–177.

Moortgat, M. (2001), Structural equations in language learning, *in* P. de Groote, G. Morrill & C. Retoré, eds, 'Logical Aspects of Computational Linguistics', Vol. 2099 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 1–16.

Moortgat, M. & Moot, R. (2002), Using the Spoken Dutch Corpus for type-logical grammar induction, *in* 'Proceedings of the Third International Language Resources and Evaluation Conference', Las Palmas.

Moortgat, M. & Oehrle, R. T. (1994), Adjacency, dependency and order, *in* 'Proceedings 9th Amsterdam Colloquium', pp. 447–466.

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

Srinivas, B. (1997), Performance evaluation of supertagging for partial parsing, *in* 'Proceedings of Fifth International Workshop on Parsing Technology', Boston.

Steedman, M. (2001), *The Syntactic Process*, MIT Press, Cambridge, Massachusetts.