Richard Moot

Type-Logical Grammars

Statistical Parsing for
I Overview

(!) The Spoken Dutch Corpus (CGN)
Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Type-Logical Proof Nets
Overview

(iii) Extracting a Type-Logical Treebank

(ii) Type-Logical Proof Nets

(i) The Spoken Dutch Corpus (CGN)
I Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Type-Logical Proof Nets

(iii) Extracting a Type-Logical Treebank

(iv) Lexical Ambiguity
1 Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Extracting a Type-Logical Treebank

(iii) Type-Logical Proof Nets

(iv) Lexical Ambiguity

(v) Connecting Formulas

(vi) Type-Logical Formulas
1 Overview

The Spoken Dutch Corpus (CGN)

Type-Logical Proof Nets

Extracting a Type-Logical Treebank

Connecting Formulas

Conclusions

Lexical Ambiguity
2 The Spoken Dutch Corpus (CGN)
The Spoken Dutch Corpus (CGN)

10 Million word corpus of contemporary spoken Dutch.
The Spoken Dutch Corpus (CGN)

▶ 10 Million word corpus of contemporary spoken Dutch.
▶ POS tagging, phonetic transcription, ...

Ken Dutch.
The Spoken Dutch Corpus (CGN)

- 1 Million words syntactically annotated.
- POS tagging, phonetic transcription, ...
- 1 Million word corpus of contemporary spo-
- Ken Dutch.
- 2
A rich annotation format which is as theory neutral as possible.
Syntactic Annotation

A rich annotation format which is as theory neutral as possible.

Export format should allow users to view the annotation in a way which is most convenient for them.
A rich annotation format which is as theory neutral as possible.

Export format should allow users to view the annotation in a way which is most convenient for them.

That is, we want to derive theory specific notions from the theory neutral export format.

Syntactic Annotation
Welke films hebben zij in WW2?
CGN Annotation Graphs

- Direct Acyclic Graph
- Edges: Annotated With Dependency Relations
- Vertices: Annotated With Syntactic Categories

Welke films hebben zij uit WW2?
CGN Annotation Graphs

- Vertices Annotated With Syntactic Categories
- Edges Annotated With Dependency Relations
- Directed Acyclic Graph

welke filmshebbenzij?
welke N3 films hebben zij LET?

CGN Annotation Graphs

- Edges Annotated With Dependency Relations
- Vertices Annotated With Syntactic Categories
- Direct Acyclic Graph
3 Type-Logical Proof Nets

Basic Elements
3 Type-Logical Proofs

Basic Elements

Terminals
3 Type-Logical Proofs

Basic Elements

Terminals

Amerika
Amsterdam
Asthma
3 Type-Logical Proof Nets

Basic Elements

Terminals

Nonterminals

...
3 Type-Logical Proof Nets

Basic Elements

Terminals

Nonterminals
Constructors
Graph Contractions
Example Lexicon

Amerika

du
Example Lexicon
Example Lexicon

- badkamer: u
- afval: u
- asthama: du
- Amsterdam: du
- Amerika: du
Example Lexicon

- Amerika
- Amsterdam
- achterban
- afval
- badkamer
- Amerika
Example Lexicon
Example Lexicon
Example Lexical Entry

s

\text{demand}

\text{du}
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Example: Iemand verscheen
Algorithmic Aspects
Algorithmic Aspects

Automated deduction for the proof net calculus

This is divided into three stages:
Automated deduction for the proof net calculus is divided into three stages:

1. Look up a lexical proof structure for each word of the input.
Algorithmic Aspects

 Automated deduction for the proof net calculus is divided into three stages:

(i) Look up a lexical proof structure for each word of the input.

(ii) Connect atomic formulas.

(iii) Just is divided into three stages.
Algorithmic Aspects

Automated deduction for the proof nets calculus is divided into three stages.

(i) Look up a lexical proof structure for each word of the input.

(ii) Connect atomic formulas.

(iii) Contract the resulting proof structure to a tree.
Extracting a Lexicon

Moortgat & Moot (2002) present a parametric algorithm for extracting a type-logical lexicon from CGN annotation graphs.
Extracting a Lexicon

Moortgat & Moot (2002) present a parametric algorithm for extracting a type-logical lexicon from CGN annotation graphs.

Take a CGN annotation graphs.

CGN annotation graphs.
Extracting a Lexicon

Moortgat & Moot (2002)

Present a parametric algorithm for extracting a type-logical lexicon from CGN annotation graphs.

1. Take a CGN annotation graphs.
2. Identify the functor of every domain.
3. CGN annotation graphs.
Extracting a Lexicon

Moortgat & Moot (2002) present a parametric algorithm for extracting a type-logical lexicon from CGN annotation graphs.

1. Take a CGN annotation graph.
2. Identify the functor of every domain.
3. Recursively disconnect all daughters which are not the functor.
4. Extracting a Lexicon
Extracting a Lexicon

Moortgat & Moot (2002)
present a parametric algo-

4

rithm for extracting a type-logical lexicon from CGN annotation graphs.

Take a CGN annotation graphs.

Identify the functor of every domain.

Recursively disconnect all daughters which are not the functor.

Translate the obtained graphs into lexical proof structures.
Welke films hebben zij?
Welke films hebben zij WW2?
Identify Functors

Welke films hebben zij WW2?
Identify Functors
Welke films hebben zij WW2?
welke films hebben zij?
welke films hebben zij?
Welke фильмы WW2 hebben zij?
Welke films hebben zij in WW2?
welke films hebben zij?
Welke films hebben jullie zitten?
Welke films hebben zij?
Welke films hebben jullie gisteren gezien?
Welke films hebben jullie gezien?
Welke films hebben zij gisteren gezien?
welke films hebben zij let?
welke

films

n

n

hebben

zij

np

np

?

let

Translation

why

has

the

films

zij

have

welke films hebben zij?
welke films hebben zij?
Welke films hebben zij gemaakt?
welke films hebben zij?
Welke films hebben zij?
Welke films hebben zij laten zien?
Translation

Welke films hebben zij? du

¿Qué films tienen ellos? du
Welke films hebben zij laten?
Order of Constituents

Remembrances
Many verb entities will be generated for different word orders in different sentence types.
Many verb entries will be generated for different word orders in different sentence types. The formula assigned to ‘hebben’ has the constituents in OVS order. Many verb entries will be generated for different word orders in different sentence types.
Many verb entries will be generated for different word orders in different sentence types. The formula assigned to 'hebben' has the constituents in OVS order. Our solution is to treat all sentences as verb final and select the constituents from left to right using an obliqueness ordering.
Treatment of Multiple Dependencies
Multiple dependencies are translated using auxiliary constructors.
Multiple dependencies are translated using auxiliary constructors. However, the formulas produced by the translation look a bit unfamiliar.
Multiple dependencies are translated using auxiliary constructors.

However, the formulas produced by the transformations look a bit unfamiliar.

For example, when we compute the formula assigned to 'welke', it is

\[ \frac{u}{(du \bullet (\tau s/\eta m))} \]
Treatment of Multiple Dependencies

- Multiple dependencies are translated using auxiliary constructors.

- However, the formulas produced by the translation look a bit unfamiliar.

- For example, when we compute the formula assigned to ‘welke’ it is \(((whq/sv1) \cdot np)/n\).

- A solution is to attach these dependencies at the argument category of which the second parent is a child.
Welke films hebben zij?
Welke films hebben zij?

Translation: Revised
welke films hebben zij?
Welke films hebben jullie gezien?
Welke films hebben zij?
Translation

I have seen films.
What happens when we parse sentences with the generated lexicon?
The algorithm produces a very large lexicon.

Generated lexicon?

What happens when we parse sentences with the
Evaluation

What happens when we parse sentences with the generated lexicon?

- The algorithm produces a very large lexicon.
- A test run over 46,229 words produced
Evaluation

What happens when we parse sentences with the generated lexicon?

The algorithm produces a very large lexicon.

A test run over 46,229 words produced 2,849 distinct trees.
Evaluation

What happens when we parse sentences with the generated lexicon?

- The algorithm produces a very large lexicon.
- A test run over 46,229 words produced many common words with over 38 lex-
  ical entries.
- 2,849 distinct trees.
Evaluation

What happens when we parse sentences with the generated lexicon?

- The algorithm produces a very large lexicon.
- A test run over 46,229 words produced
  - 2,849 distinct trees,
  - many common words with over 38 lexical entries.
- Brute force lexical search is not an option.
What happens when we parse sentences with the generated lexicon?

For large sentences, we have to consider many possible connections.

Brute force lexical search is not an option.

- The algorithm produces a very large lexicon.

A test run over 46,229 words produced 2,849 distinct trees.

Many common words with over 38 lexical entries.
5
Lexical Ambiguity
5 Lexical Ambiguity

Collapsing Lexical Entries
Reduce the size of the lexicon by collapsing similar entries.
5 Lexical Ambiguity

Collapsing Lexical Entries

When are two lexical entries for a word similar?

Reduce the size of the lexicon by collapsing similar entries.

![Image of a pie chart with two slices: one labeled 'Lexical Entries' and another labeled 'Collapsing Lexical Entries.']
Suppose word $w$ has two lexical entries of the following form.
Suppose word \( w \) has two lexical entries of the following form.
Suppose word $w$ has two lexical entries of the following form:

\[ \emptyset \neq d \]

\[ \begin{array}{c}
  & w \\
B & d \\
\hline
A & w \\
\end{array} \]
Suppose word \( w \) has two lexical entries of the following form.

Two atomic formulas \( A, B \) •

\( \emptyset \neq d \) •

\[ w \]

\[ d \]
Suppose word \( w \) has two lexical entries of the following form:

\[
\begin{align*}
A, B, &\text{ atomic formulas} \\
\emptyset &\neq d
\end{align*}
\]

We want to reduce these entries to a single entity of the form:

\[
\begin{align*}
\emptyset &\neq d
\end{align*}
\]
Suppose word $w$ has two lexical entries of the following form:

\[ P \wedge \neg \emptyset \]

\[ A, B \text{ atomic formulas} \]

\[ A \land B \neq \emptyset \]

We want to reduce these entries to a single entry of the form $C$ for some formula $C$.

The following form.

Entity of the form $d$.
Radical
Replace all occurrences of $\mathbf{A}$ and $\mathbf{B}$ in the text by occurrences of a new atomic formula $\mathbf{C}$. 
Replace all occurrences of \( A \) and \( B \) in the text by occurrences of a new atomic formula \( C \), and generate many identifications.
Radical

Replace all occurrences of A and B in the lexicon by occurrences of a new atomic formula C.

Many identifications shift part of the lexical complexity to the connection stage.
Conservative
that \( \mathcal{C} \vdash A \) and \( \mathcal{C} \vdash B \). Replace all occurrences of \( A \) and \( B \) in the lexicon by occurrences of \( A' \) and \( B' \), such that \( \mathcal{C} \vdash A' \) and \( \mathcal{C} \vdash B' \).
Conservative

Replace all occurrences of \( A \) and \( B \) in the lexicon by occurrences of formulas \( A' \) and \( B' \) such that \( C \vdash A' \) and \( C \vdash B' \).

Whenever a word has both a \([A]_d\) and a \([B]_d\) entry, we can replace both by a single \([C]_d\). Replace all occurrences of a formulas \( A' \) and \( B' \) in the lexicon.
Conservative

Replace all occurrences of $A$ and $B$ in the lexicon by occurrences of formulas $A'$ and $B'$ such that $C \vdash A'$ and $C \vdash B'$.

Whenever a word has both a $[A]$ and a $[B]$ entry, we can replace both by a single $[C]$ entry. No accidental identifications.

Replace all occurrences of $A$ and $B$ in the lexicon.
Replace all occurrences of $A$ and $B$ in the lexicon by occurrences of formulas $A'$ and $B'$ such that $C \vdash A'$ and $C \vdash B'$.

Whenever a word has both an $A$ and a $B$ entry, we can replace both by a single $C$ entry. Whenever a word has both an $A$ and a $B$, that $C \vdash A$ and $C \vdash B$.

Replace all occurrences of $A$ and $B$ in the lexicon.

Conservative translation stage:

- shift part of the lexical complexity to the con...

+ no accidental identifications;

$[C]d$. Whenever a word has both a $A$ and a $B$ entry, we can replace both by a single $C$ entry.

$[B]d$. Whenever a word has both an $A$ and a $B$, that $C \vdash A$ and $C \vdash B$.

Conservative
Conservative

Replacing all occurrences of $A$ and $B$ in the lexicon by occurrences of formulas $A'$ and $B'$ such that $C \vdash A'$ and $C \vdash B'$.

Whenver a word has both an $A$ and a $B$ entry, we can replace both by a single $C$. That is, if $A$ and $B$ are connected to $C$ in the lexicon by occurrences of $A$ and $B$ formulas, then replace all occurrences of $A$ and $B$ in the lexicon by $C$.

Identical influence on the complexity of the contraction stage.

No accidental identifications.
Supertagging
During the development of the XTAG project, TAG parsers suffered from massive lexical ambiguity.
During the development of the XTAG project, TAG parsers suffered from massive lexical ambiguity. A solution to this was extending the methodology used in Part-of-Speech tagging to assign lexical trees to a given sequence of words (Joshi & Srinivas 1994).
During the development of the XTAG project, TAG parsers suffered from massive lexical ambiguity. A solution to this was extending the methodology used in Part-of-Speech tagging to assign lexical trees to a given sequence of words. This approach was referred to as Supertagging (Joshi & Srinivas 1994).

Results: faster parsing, reasonable accuracy (92.2%).
During the development of the XTAG project, TAG parsers suffered from massive lexical ambiguity.

A solution to this was extending the methodology used in part-of-speech tagging to assign lexical trees to a given sequence of words. This led to a given sequence of words, which was extended to assign dependency information to the sequence.

A solution to take into account dependency information was also to improve parsing accuracy and extendibility.

Results: Faster parsing, reasonable accuracy (92.2%).

Supertagging during the XTAG project.
Supertagging

During the development of the XTAG project, TAC parsers suffered from massive lexical ambiguity. TAG parsers suffered from massive lexical ambiguity.

A solution to this was extending the methodology used in Part-of-Speech tagging to as-assign lexical trees to a given sequence of words. A solution to this was extending the method.

Joshi & Srinivas (1994) extended the Part-of-Speech tagging method to also take dependency information into account.

Results: Faster parsing, reasonable accuracy (92.2%).

Can be extended to other lexicalized grammars.
Some Differences

Joshi & Srinivas use only a limited set (300-400) of lexical trees.
Joshi & Srinivas use only a limited set (300-400) of lexical trees.

The training data, consisting of issues of the Wall Street Journal and IBM technical man-

Joshi & Srinivas use only a limited set (300-400) of lexical trees.

Some Differences
Joshi & Srinivas use only a limited set (300-400) of lexical trees, while the full CGN annotation will be upon completion. The training data, consisting of issues of the Wall Street Journal and IBM technical manuals, was relatively clean. The reported size of the training data which gave the best performance is the same size as the full CGN annotation will be upon completion.

Some Differences
Some Differences

Joshi & Srinivas use only a limited set (300-400) of lexical trees.

The training data, consisting of issues of the Wall Street Journal and IBM technical manual, was relatively clean.

The reported size of the training data which gave the best performance is the same size as the full CGN annotation will be upon completion.

Some Differences
6
Connecting Formulas
6 Connecting Formulas

The Problem
6 Connecting Formulas

The Problem
6 Connecting Formulas

The Problem

$m!$ different ways of performing the connections.
The Problem

Connecting Formulas

\[ n \cdot w \]

\[ \cdots \]

Again, enumerating all possible solutions doesn’t seem like a good strategy.

11 different ways of performing the connect-

ions.
Suppose, however, that we assign a weight to every possible connection.
Suppose, however, that we assign a weight to every possible connection. A weight function could be based on our tree bank data:

Weighted Links
Suppose, however, that we assign a weight to every possible connection.

A weight function could be based on our tree bank data:

Given an atomic formula in a lexicon en-
Suppose, however, that we assign a weight to every possible connection. We keep track of the distance to the forest when we disconnect it. If it is linked to when we disconnect a formula in a lexical entry, we keep track of the distance to the forest.

A weight function could be based on our tree bank data. Given an atomic formula in a lexical entry, to every possible connection, we assign a weight.
Suppose, however, that we assign a weight to every possible connection. A weight function could be based on our tree.

Other weight functions are also possible:

- Given an atomic formula in a lexicon:
  - If a formula is linked to when we disconnect it, we keep track of the distance to the root.

Weights Links
Weighted Bipartite Graphs
Weighted Bipartite Graphs

Positive formulas

Negative formulas
We have polynomial algorithms to:

- positive formulas
- negative formulas
We have polynomial algorithms to:

compute a perfect matching with minimum/maximum cost (Kuhn 1955),

weighted bipartite graphs
We have polynomial algorithms to:

- compute a perfect matching with minimum/maximum cost (Kuhn 1955),
- compute the k-best perfect matchings (Chegireddy & Hamacher 1987).

Weighted Bipartite Graphs

Positive formulas

Negative formulas
7 Conclusions
Conclusions

We can automatically extract a large type-normalized logical treebank from the CGN syntactic notation.
Conclusions

We can automatically extract a large type-logical treebank from the CGN syntactic an-
notation. This produces rather unwieldy type-logical grammars.
Conclusions

We can automatically extract a large type-logical treebank from the CGN syntactic notation.

This produces rather unwieldy type-logical grammars:

- massive lexical ambiguity
- grammars
- notation

We can automatically extract a large type-logical treebank from the CGN syntactic notation.
Conclusions

- We can automatically extract a large type-logical treebank from the CGN syntactic annotation.
- This produces rather unwieldy type-logical type-logical grammars:
  - massive lexical ambiguity
  - a prohibitive number of possible connections
  - grammatical notation

Eventually, we can automatically extract a large type-logical treebank from the CGN syntactic annotation.
Conclusions

We can automatically extract a large type-logical treebank from the CGN syntactic notation. This produces rather unwieldy type-logical grammars:

- massive lexical ambiguity
- prohibitively large number of possible connections

We have sketched how statistical methods may help us solve both problems.


Algorithmic Aspects

Identify Functors

Example: Iemand verscheen

Example: Iemand verscheen

Remove Edge Labels

Translation

Extracting a Lexicon

Example: Lexical Entry

Example: Example

Example: Lexicon

Example: Lexicon

Example: Lexicon

Example: Lexicon
Bibliography

5 Lexical Ambiguity

6 Connecting Formulas

7 Conclusions

5 Evaluating the Weighted Bipartite Graphs

4 The Problem

3 Supertags and Supertagging

2 Conservative and Radical

1 Collapsing Lexical Entries

0-112

0-144

0-150

0-155

0-161

0-162

0-166