Statistical Parsing for Type-Logical Grammars

Richard Moot

Richard.Moot@let.uu.nl

1 Overview

1 Overview

(i) The Spoken Dutch Corpus (CGN)

1 Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Type-Logical Proof Nets

1 Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Type-Logical Proof Nets

(iii) Extracting a Type-Logical Treebank

1 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) Type-Logical Proof Nets

(iii) Extracting a Type-Logical Treebank

(iv) Lexical Ambiguity

(v) Connecting Formulas

1 Overview

(i) The Spoken Dutch Corpus (CGN)

(ii) Type-Logical Proof Nets

(iii) Extracting a Type-Logical Treebank

(iv) Lexical Ambiguity

(v) Connecting Formulas

(vi) Conclusions

2 The Spoken Dutch Corpus (CGN)

2 The Spoken Dutch Corpus (CGN)

- 10 Million word corpus of contemporary spoken Dutch.

2 The Spoken Dutch Corpus (CGN)

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

2 The Spoken Dutch Corpus (CGN)

- 10 Million word corpus of contemporary spoken Dutch.
- POS tagging, phonetic transcription, ...
- 1 Million words syntactically annotated.

Syntactic Annotation

Syntactic Annotation

- 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.

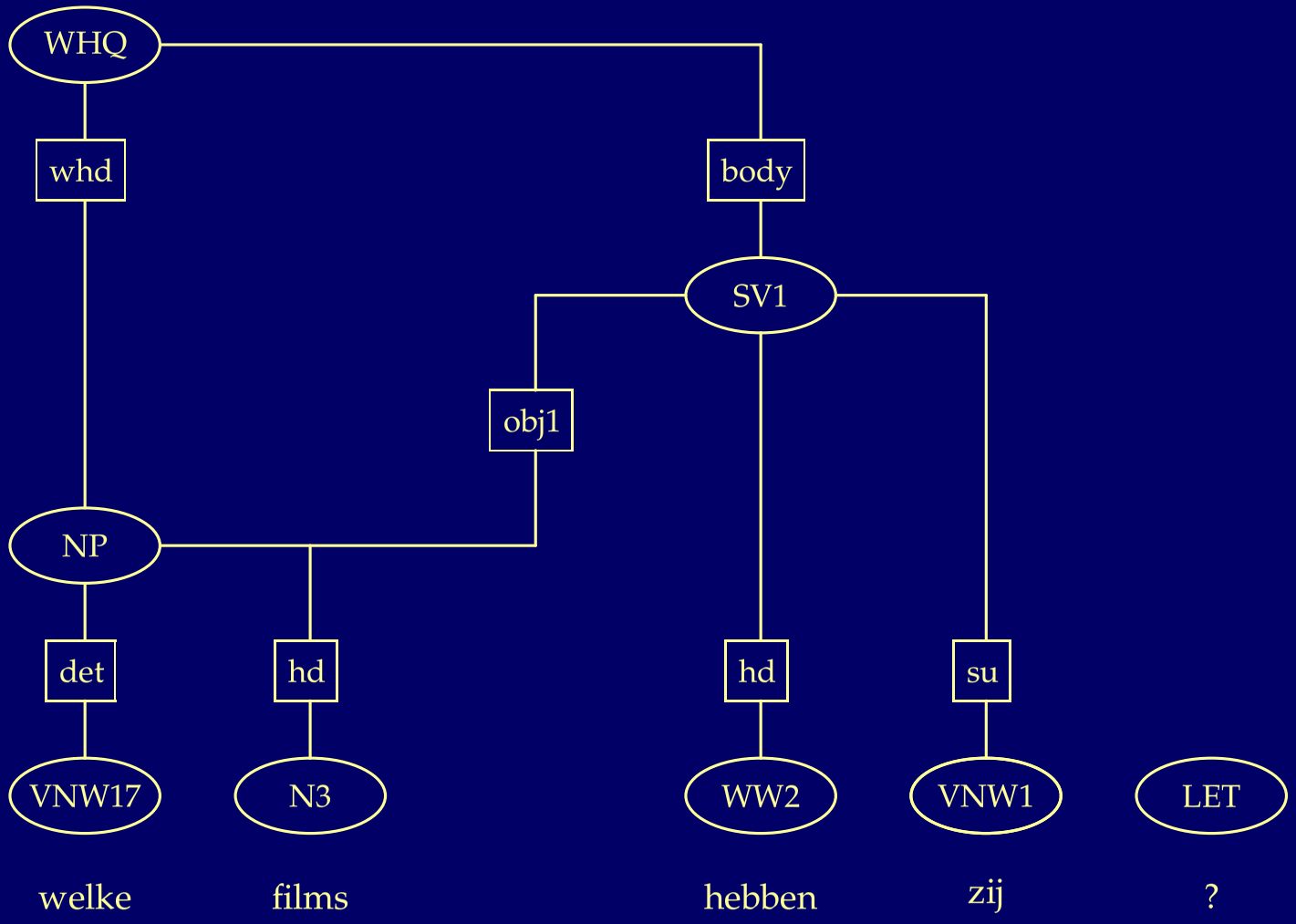
Syntactic Annotation

- 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.

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

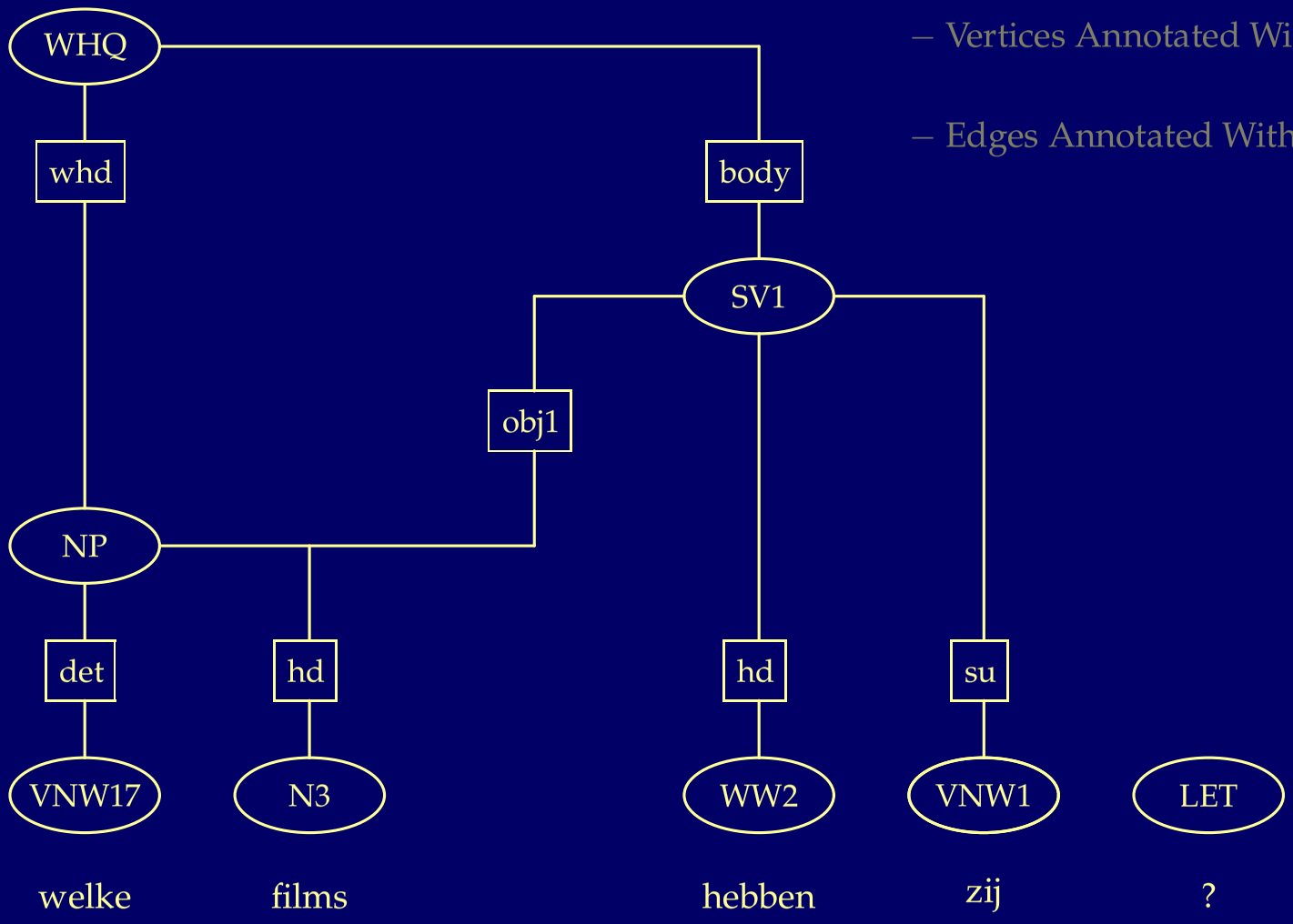
CGN Annotation Graphs

CGN Annotation Graphs



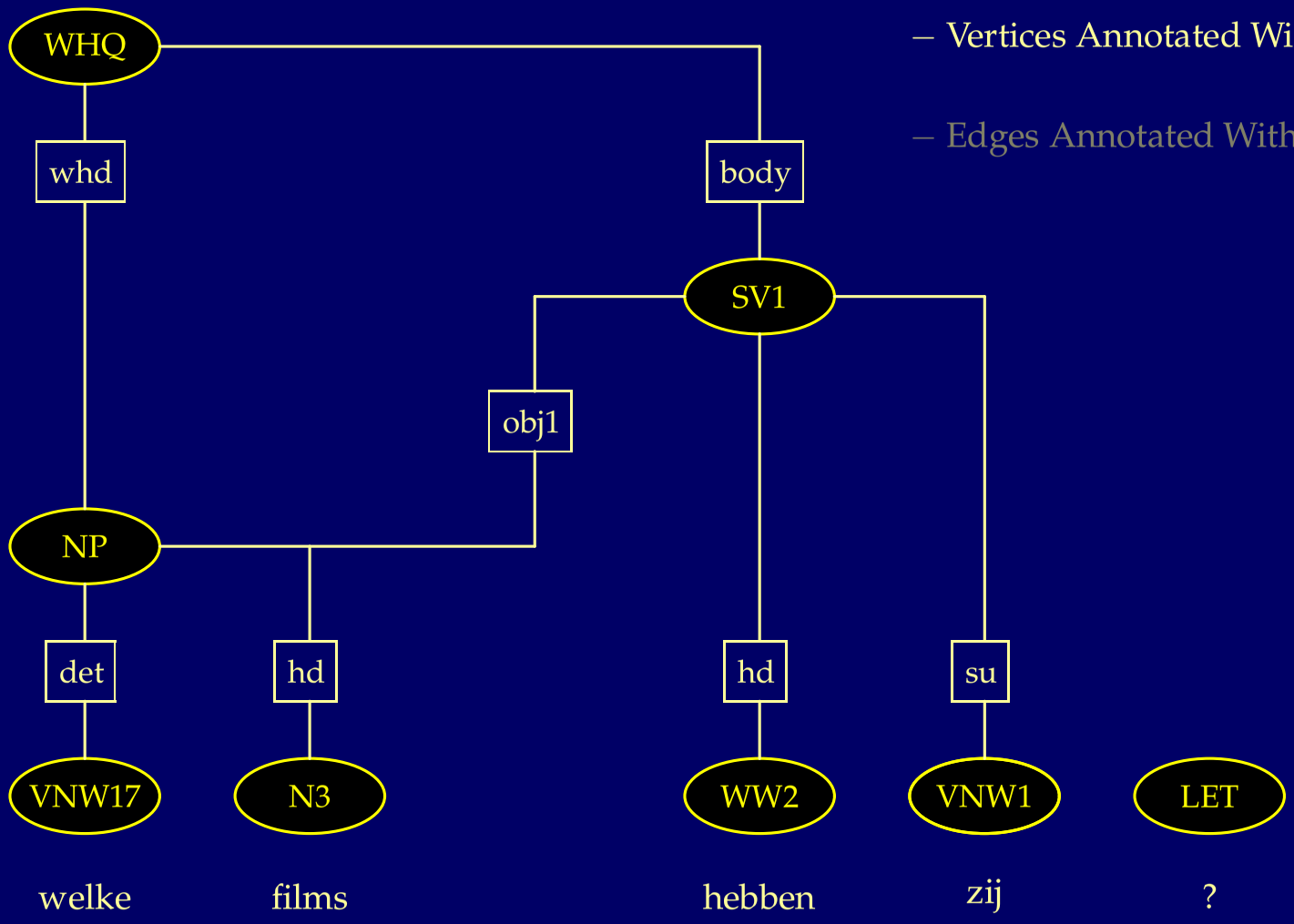
CGN Annotation Graphs

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



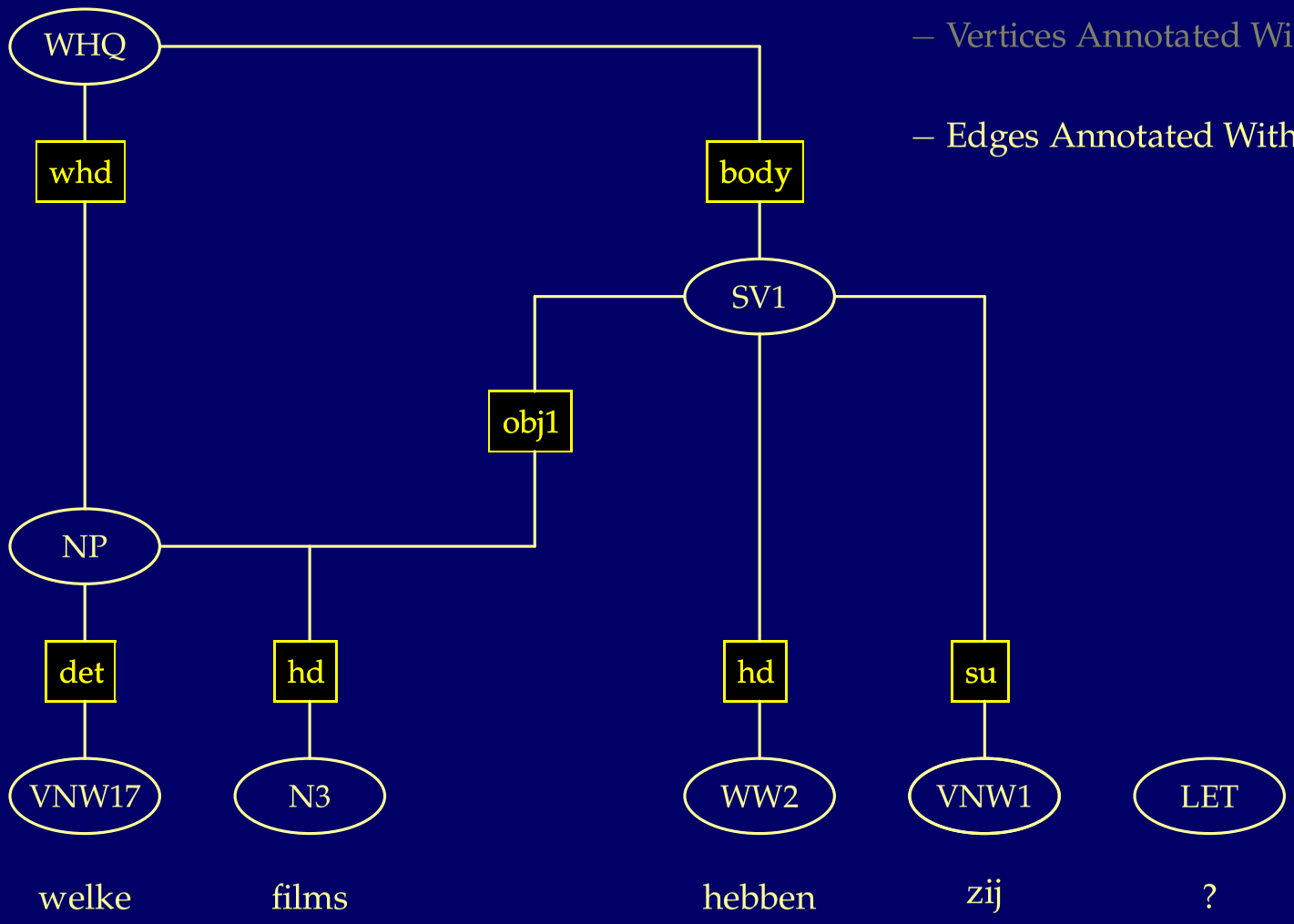
CGN Annotation Graphs

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



CGN Annotation Graphs

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



3 Type-Logical Proof Nets

Basic Elements

3 Type-Logical Proof Nets

Basic Elements

Terminals

3 Type-Logical Proof Nets

Basic Elements

Terminals

Amerika Amsterdam astma · · ·

3 Type-Logical Proof Nets

Basic Elements

Terminals

Amerika Amsterdam astma · · ·

Nonterminals

3 Type-Logical Proof Nets

Basic Elements

Terminals

Amerika Amsterdam astma . . .

Nonterminals

$s, n, np, \ldots$

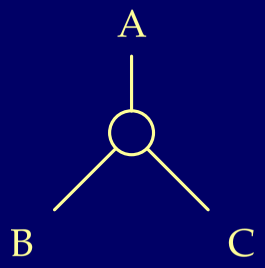
Constructors

Constructors

Main

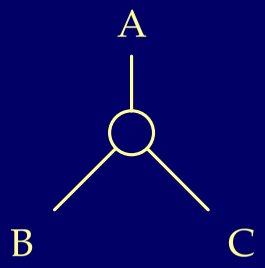
Constructors

Main



Constructors

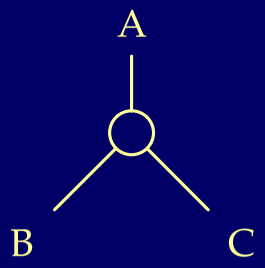
Main



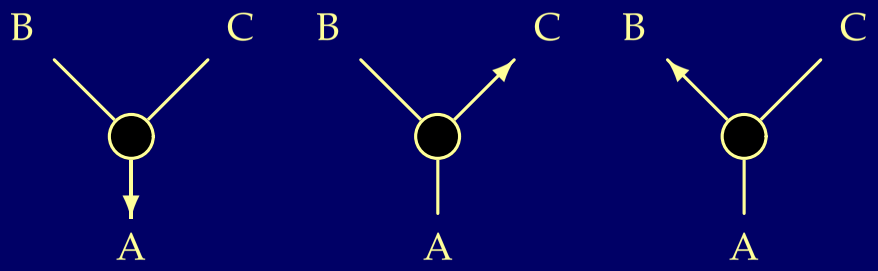
Auxiliary

Constructors

Main

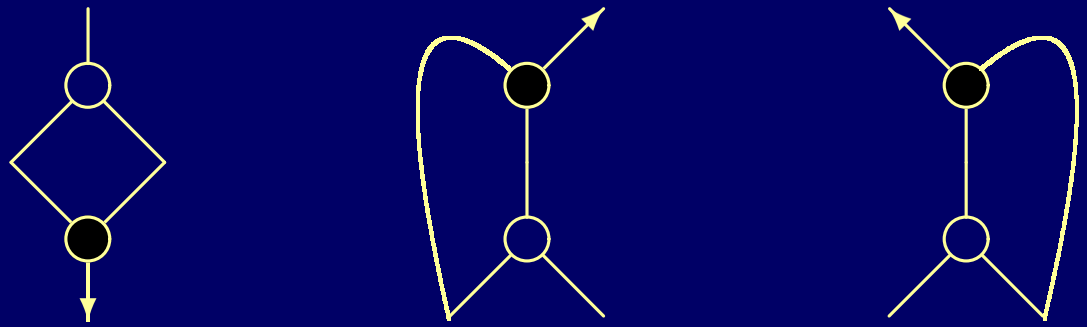


Auxiliary



Graph Contractions

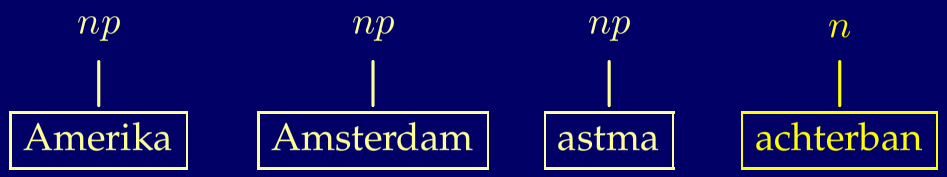
Graph Contractions



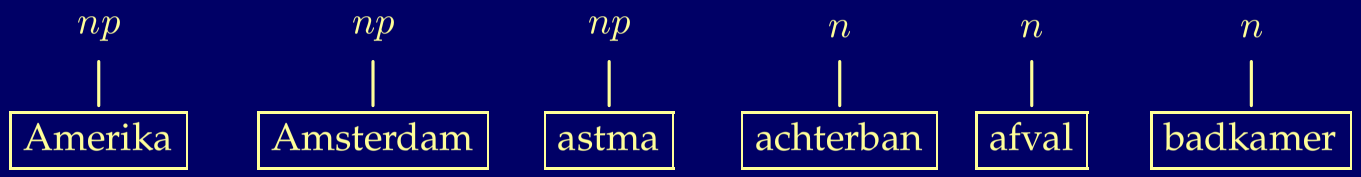
Example Lexicon

Example Lexicon

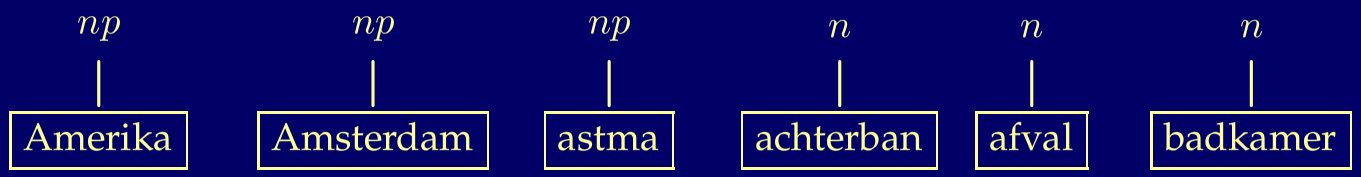
Example Lexicon

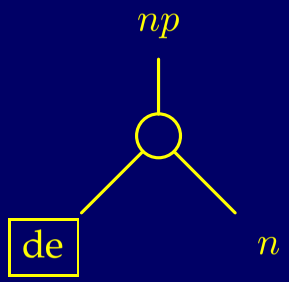


Example Lexicon

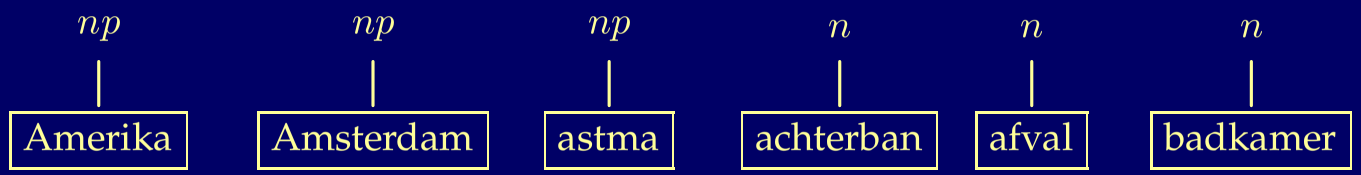


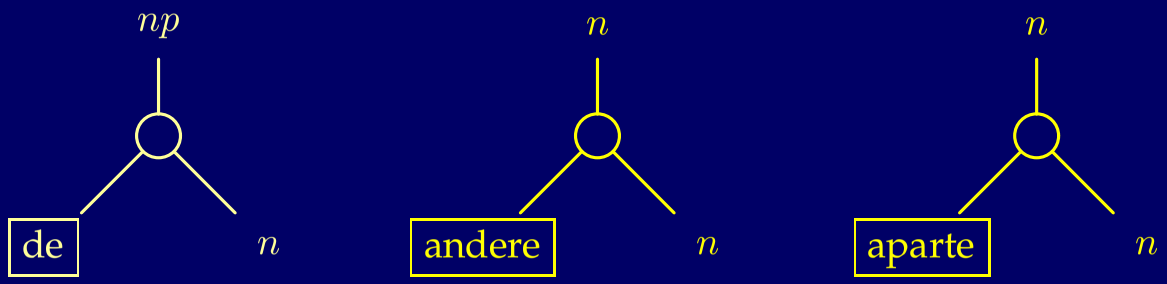
Example Lexicon



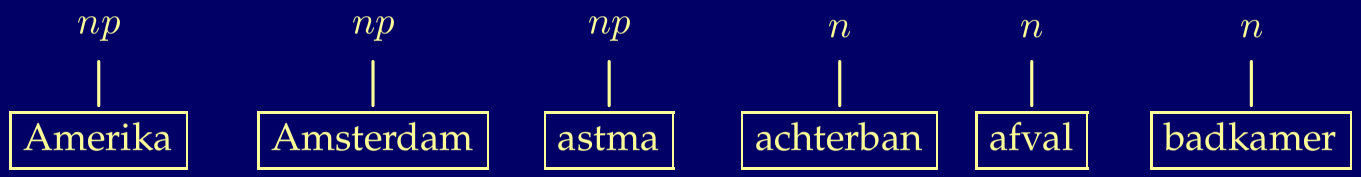


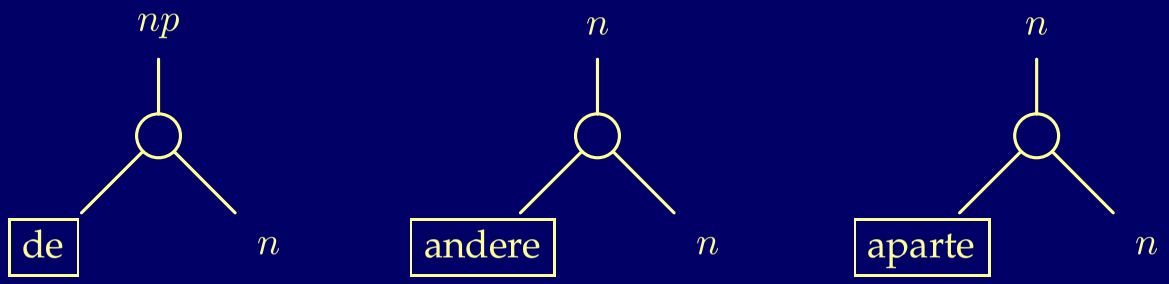
Example Lexicon

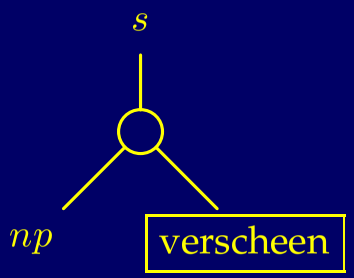




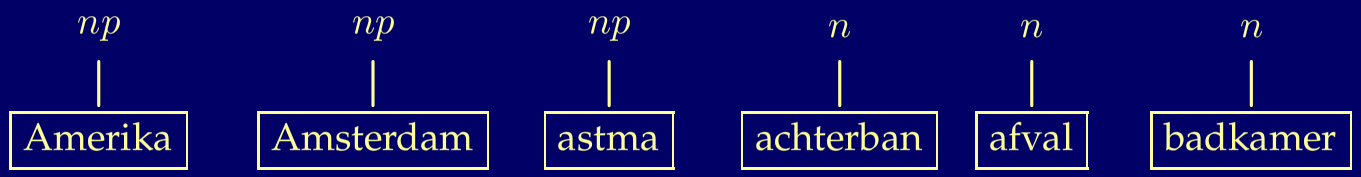
Example Lexicon

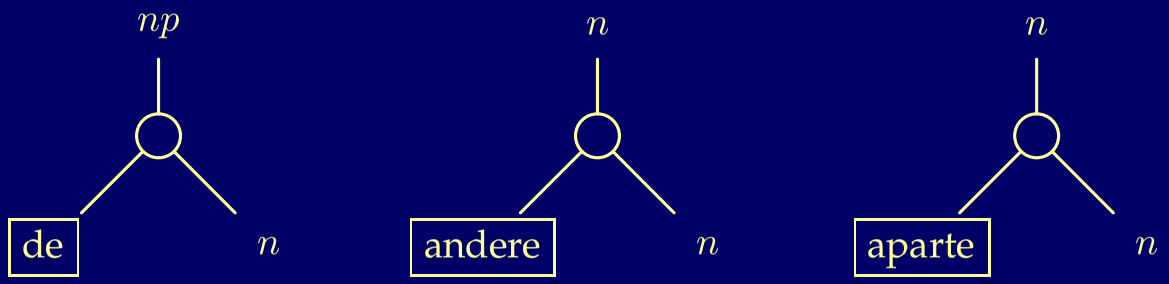


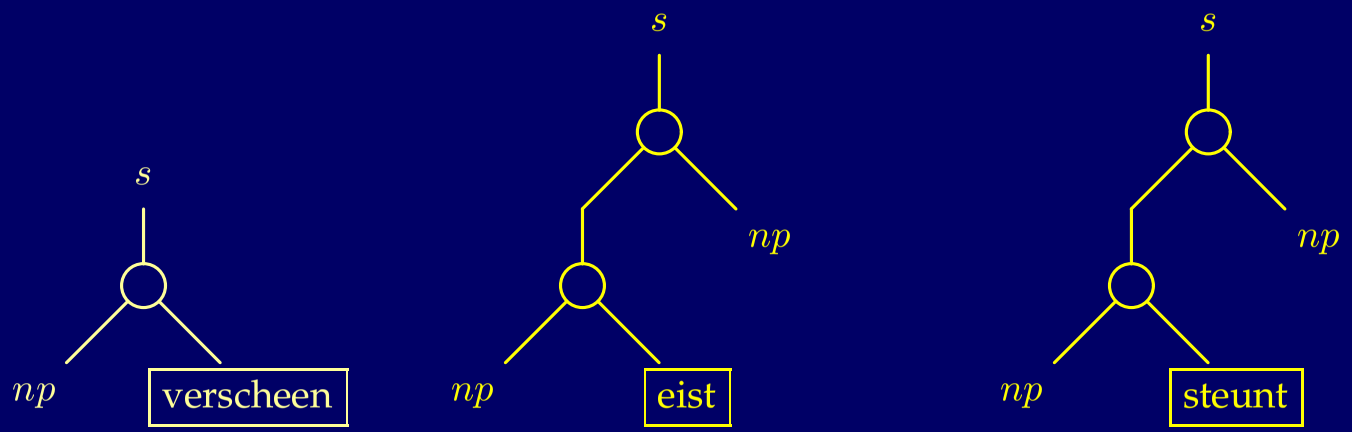




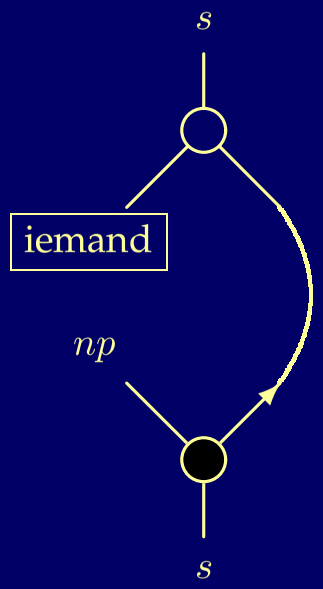
Example Lexicon



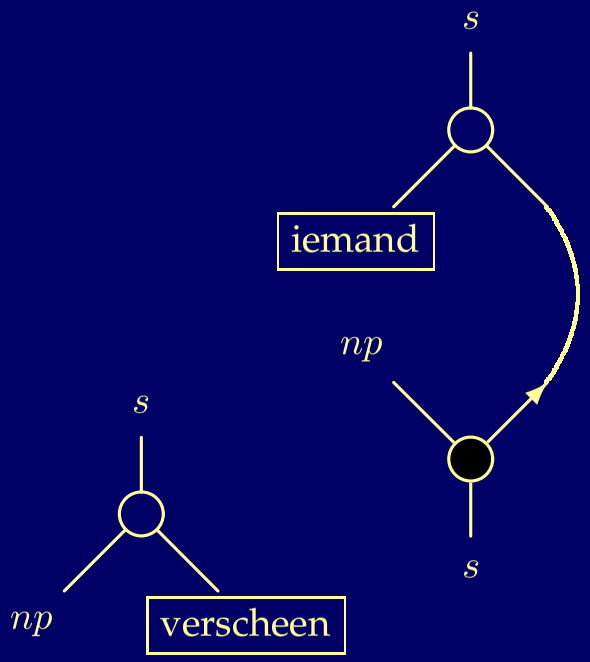




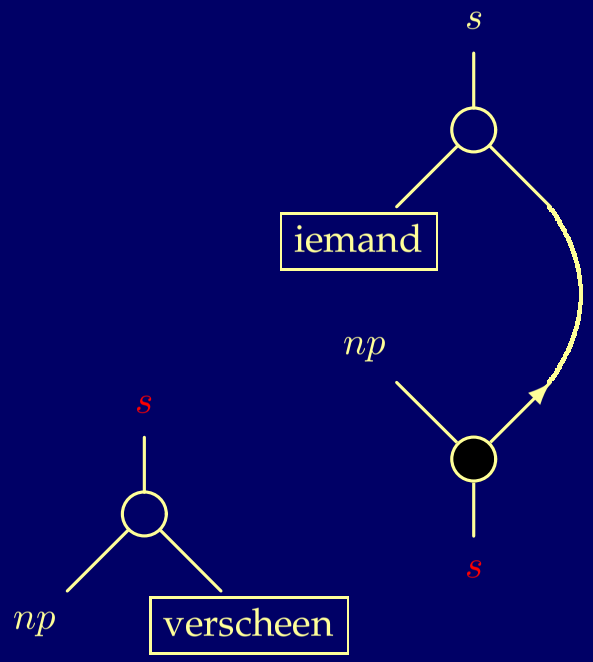
Example Lexical Entry



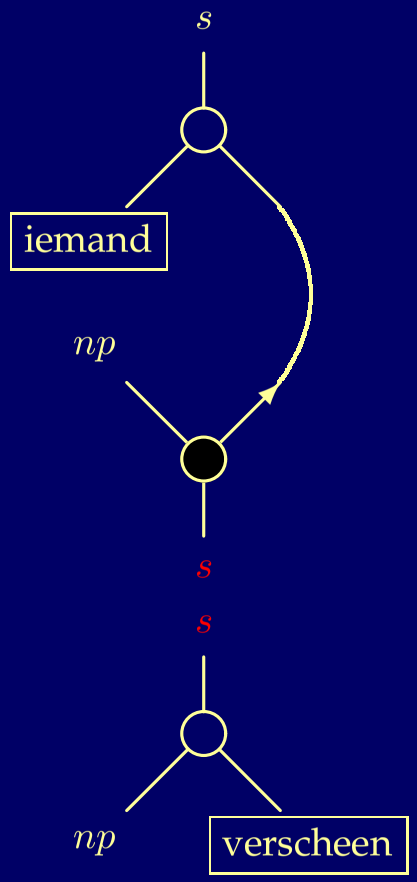
Example: Iemand verscheen



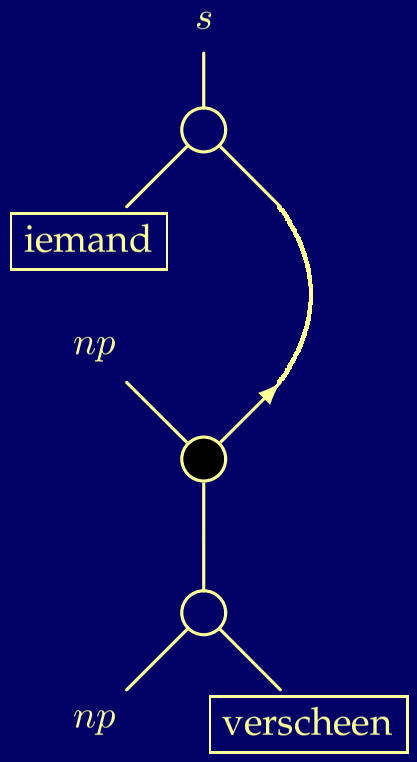
Example: Iemand verscheen



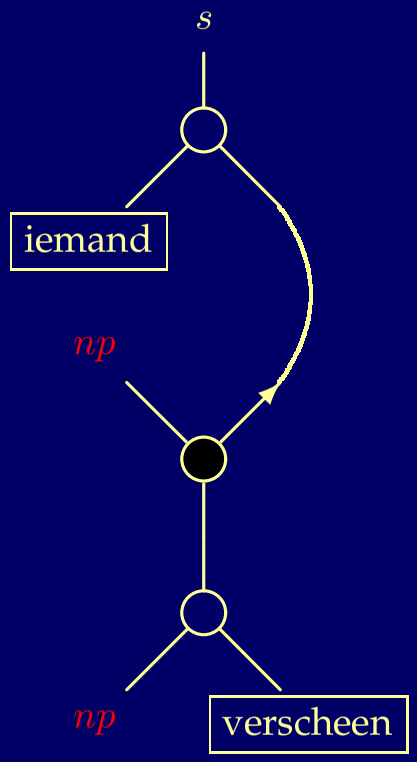
Example: Iemand verscheen



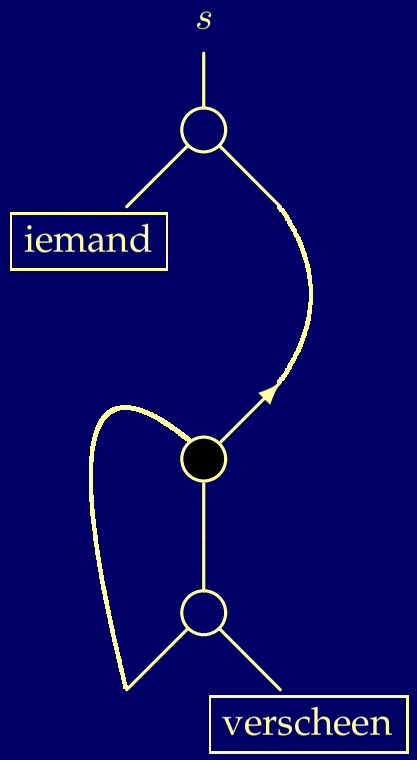
Example: Iemand verscheen



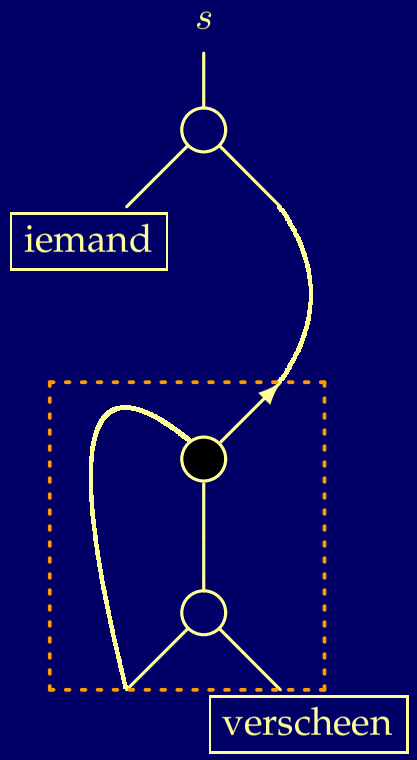
Example: Iemand verscheen



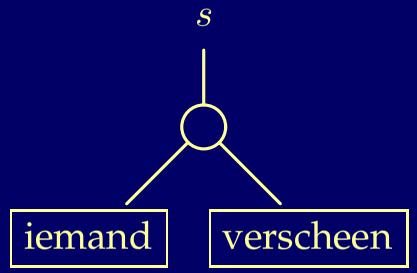
Example: Iemand verscheen



Example: Iemand verscheen



Example: Iemand verscheen



Algorithmic Aspects

Algorithmic Aspects

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

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.

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.

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) contract the resulting proof structure to a tree.

4 Extracting a Lexicon

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

4 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.

4 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.
- Identify the functor of every domain.

4 Extracting a Lexicon

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

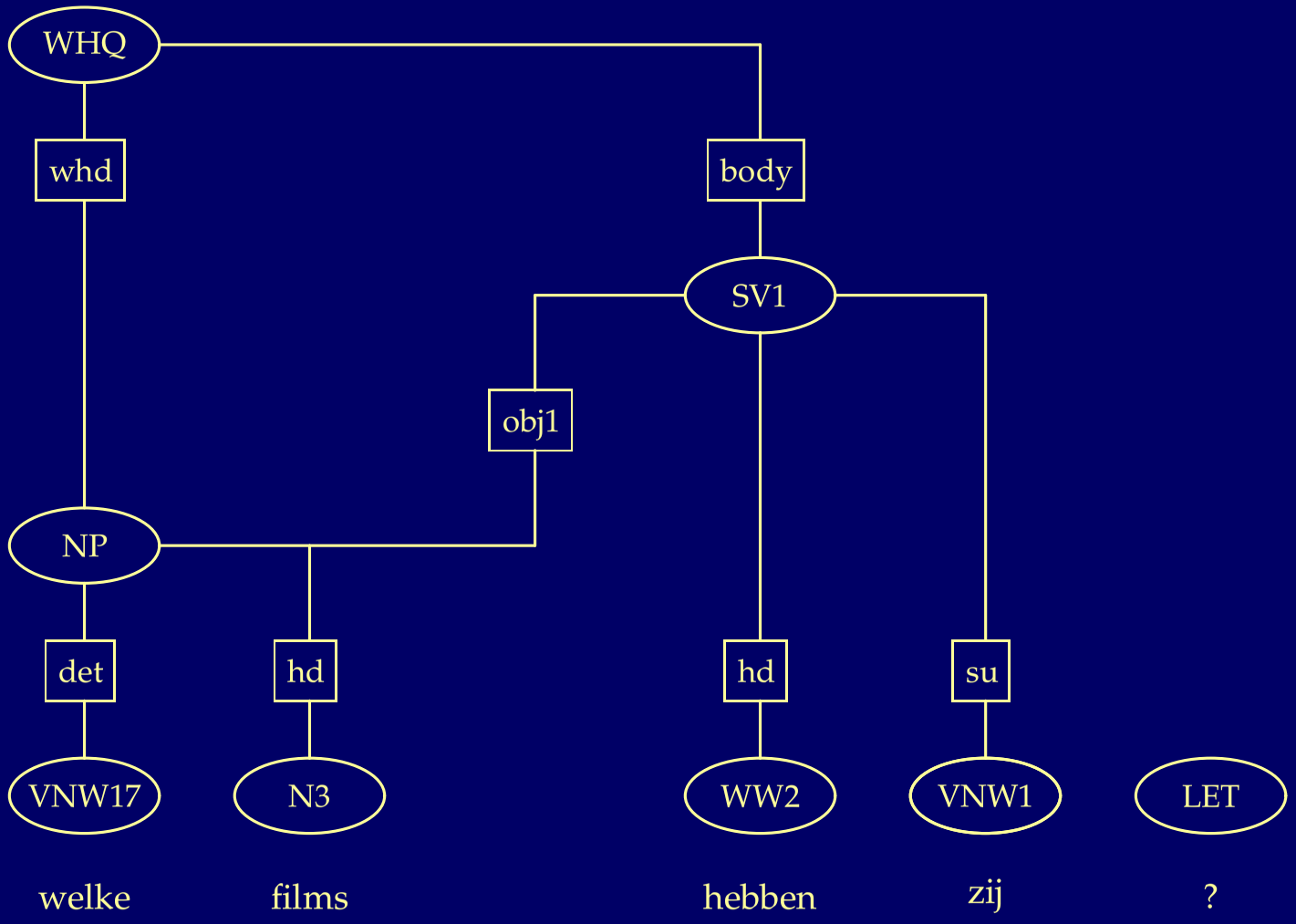
- Take a CGN annotation graphs.
- Identify the functor of every domain.
- Recursively disconnect all daughters which are not the functor.

4 Extracting a Lexicon

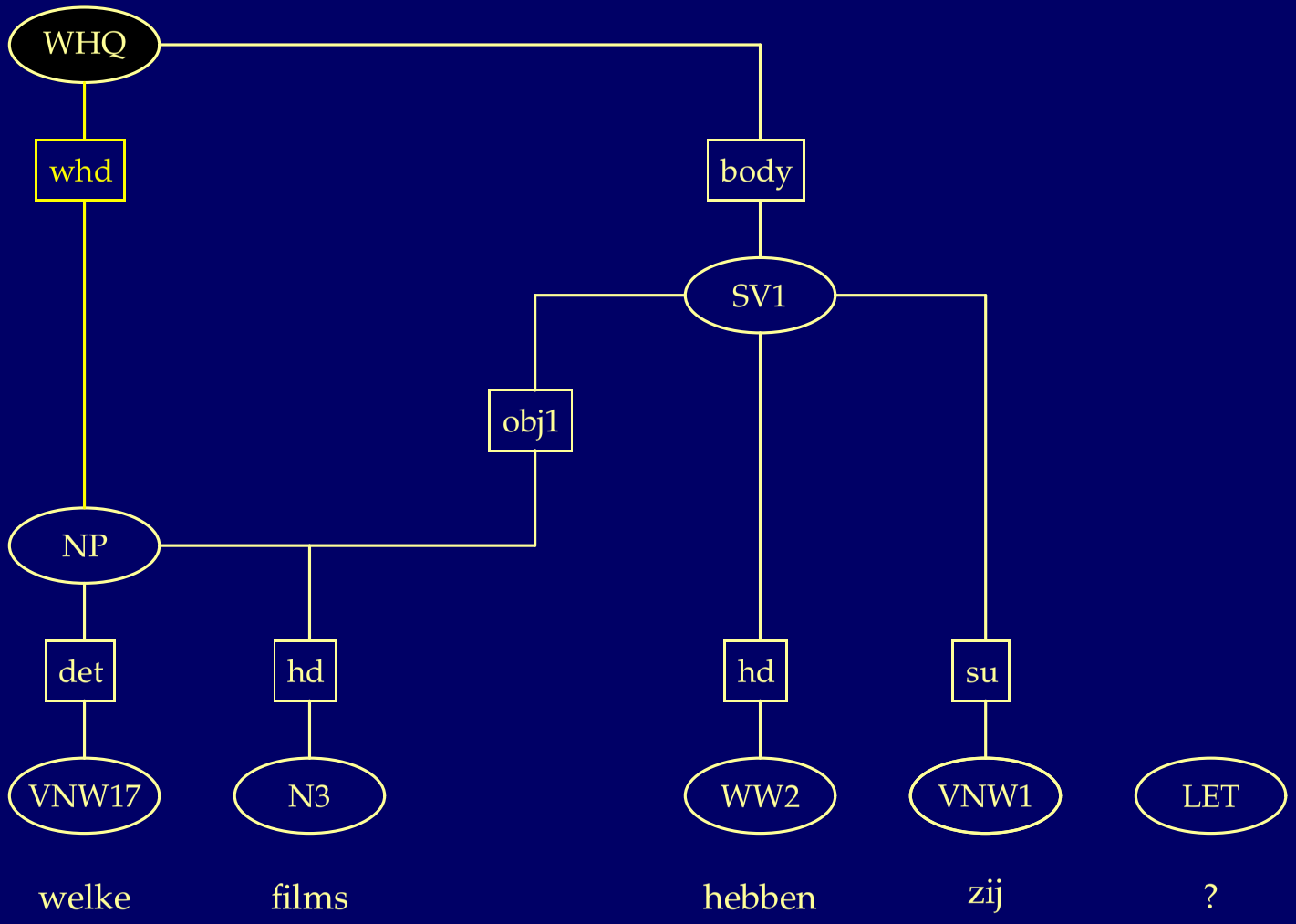
Moortgat & Moot (2002) present a parametric algorithm for extracting a type-logical lexicon from CGN annotation graphs.

- Take a CGN annotation graphs.
- Identify the functor of every domain.
- Recursively disconnect all daughters which are not the functor.
- Translate the obtained graphs into lexical proof structures.

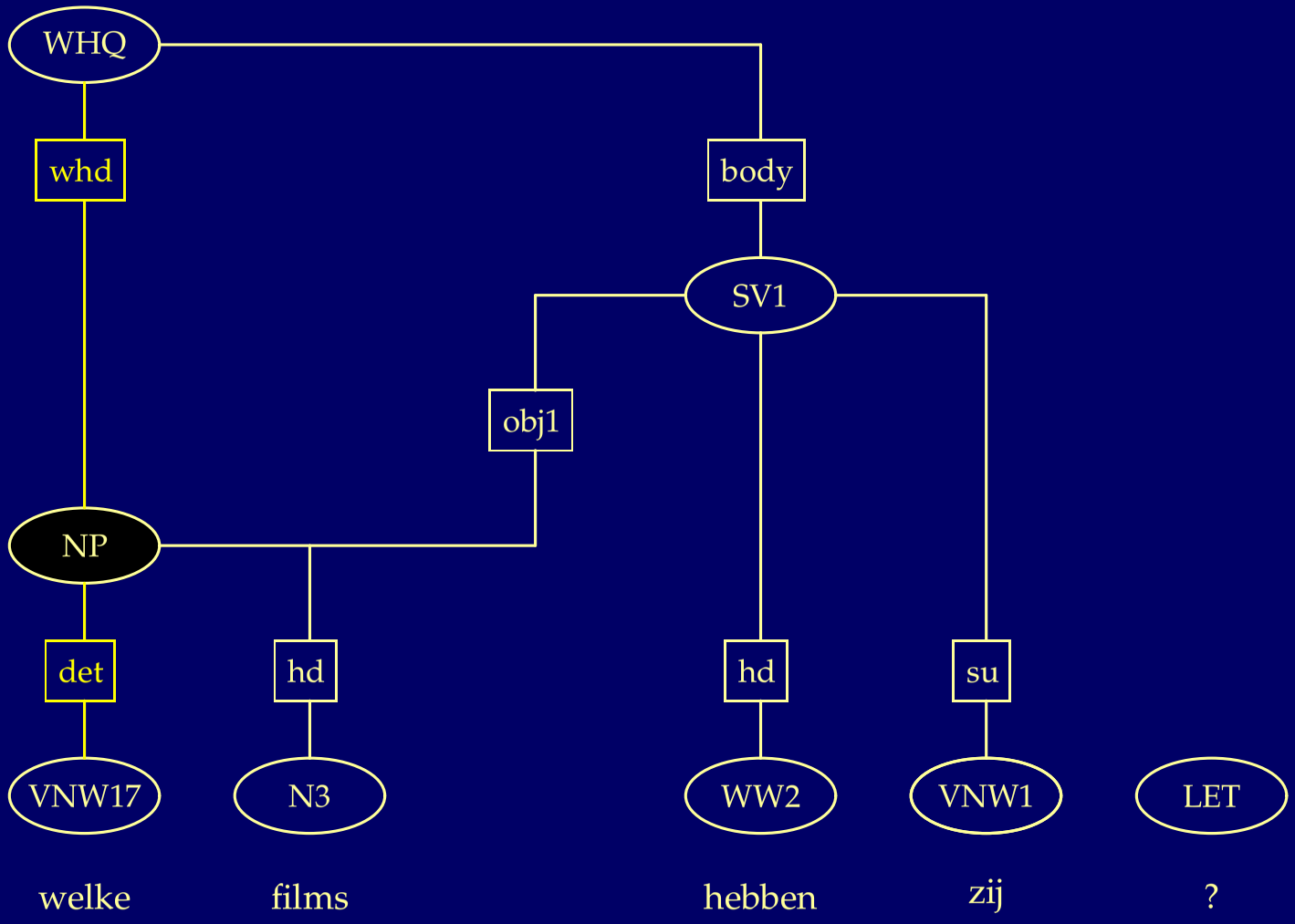
Identify Functors



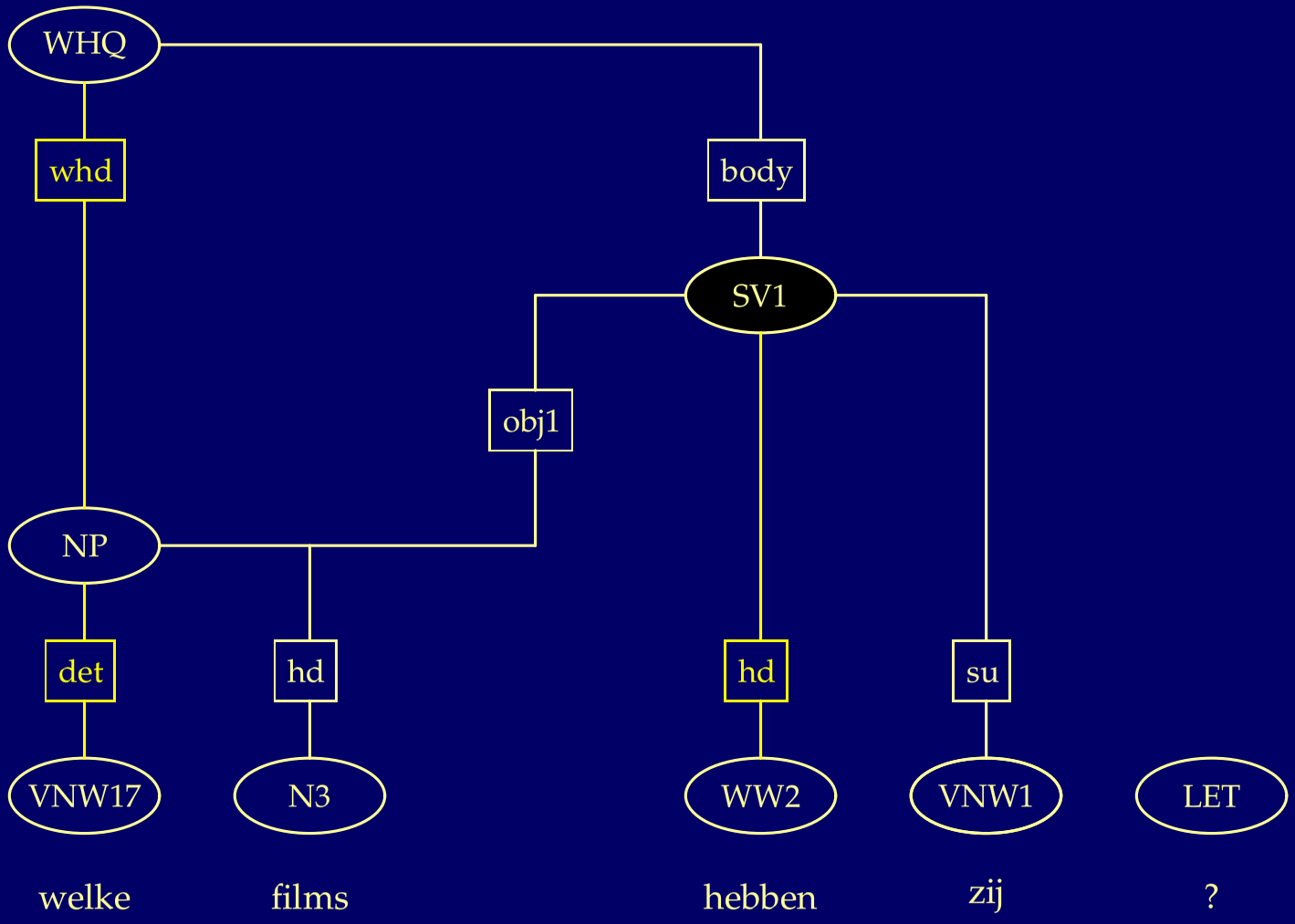
Identify Functors



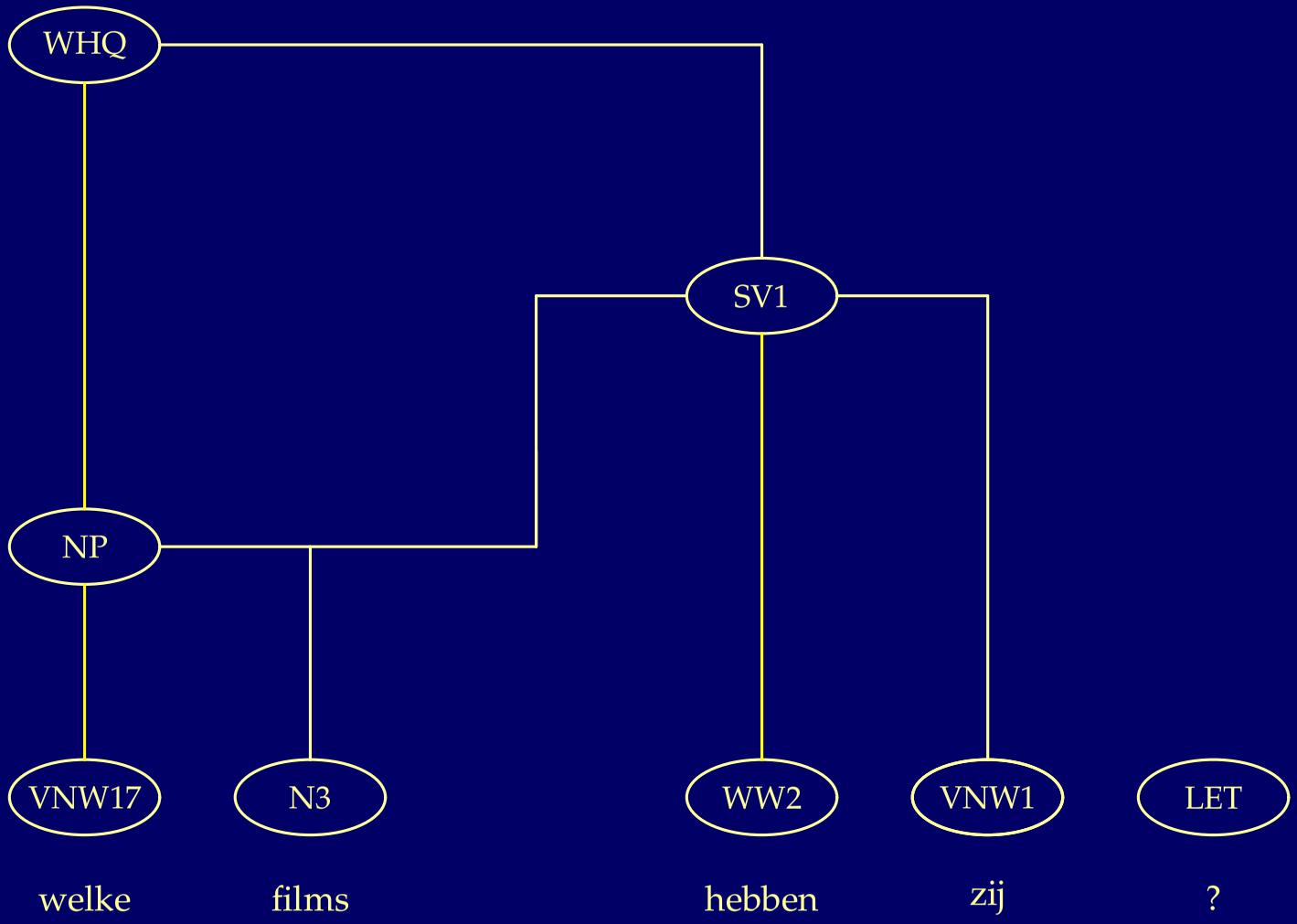
Identify Functors



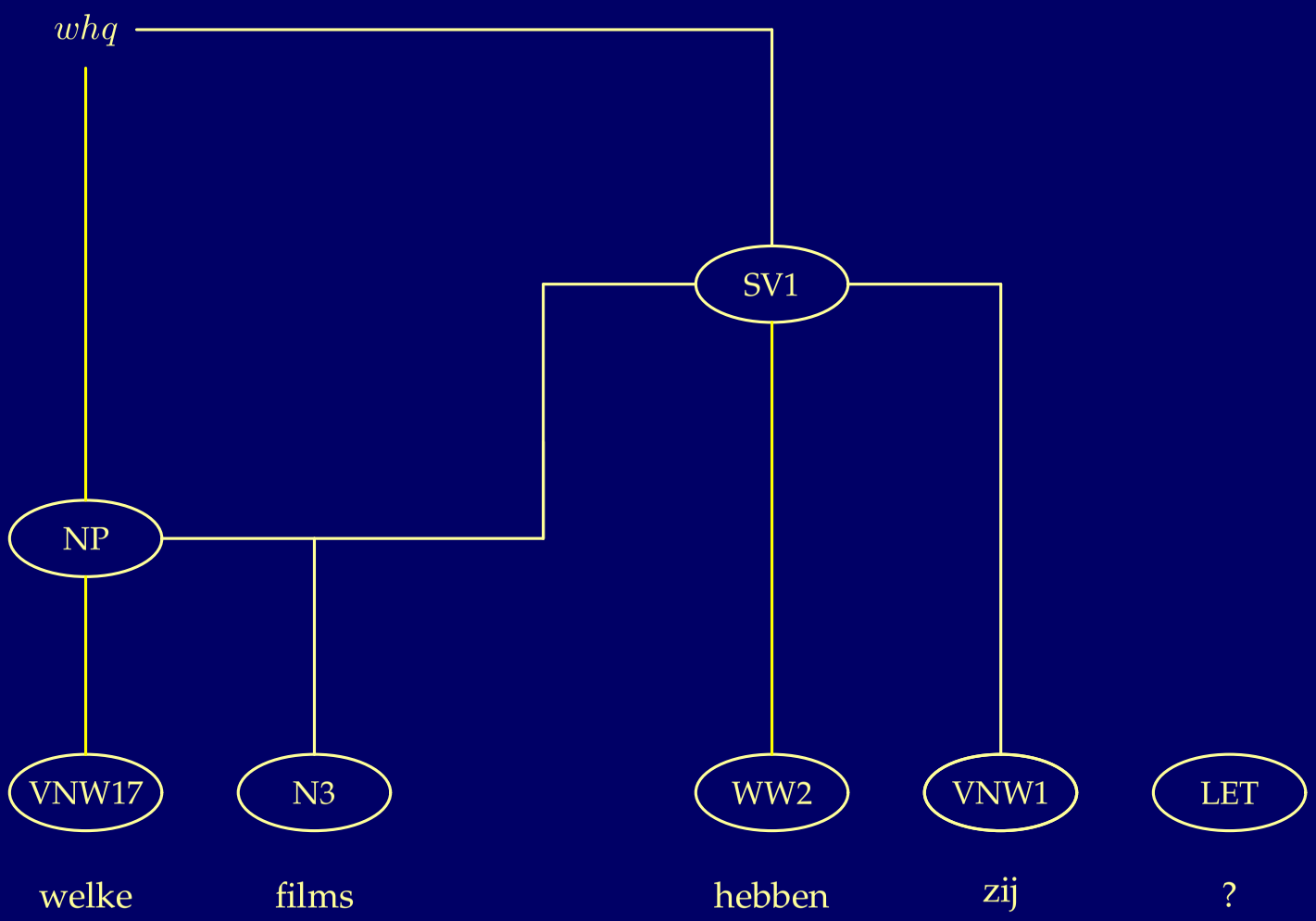
Identify Functors



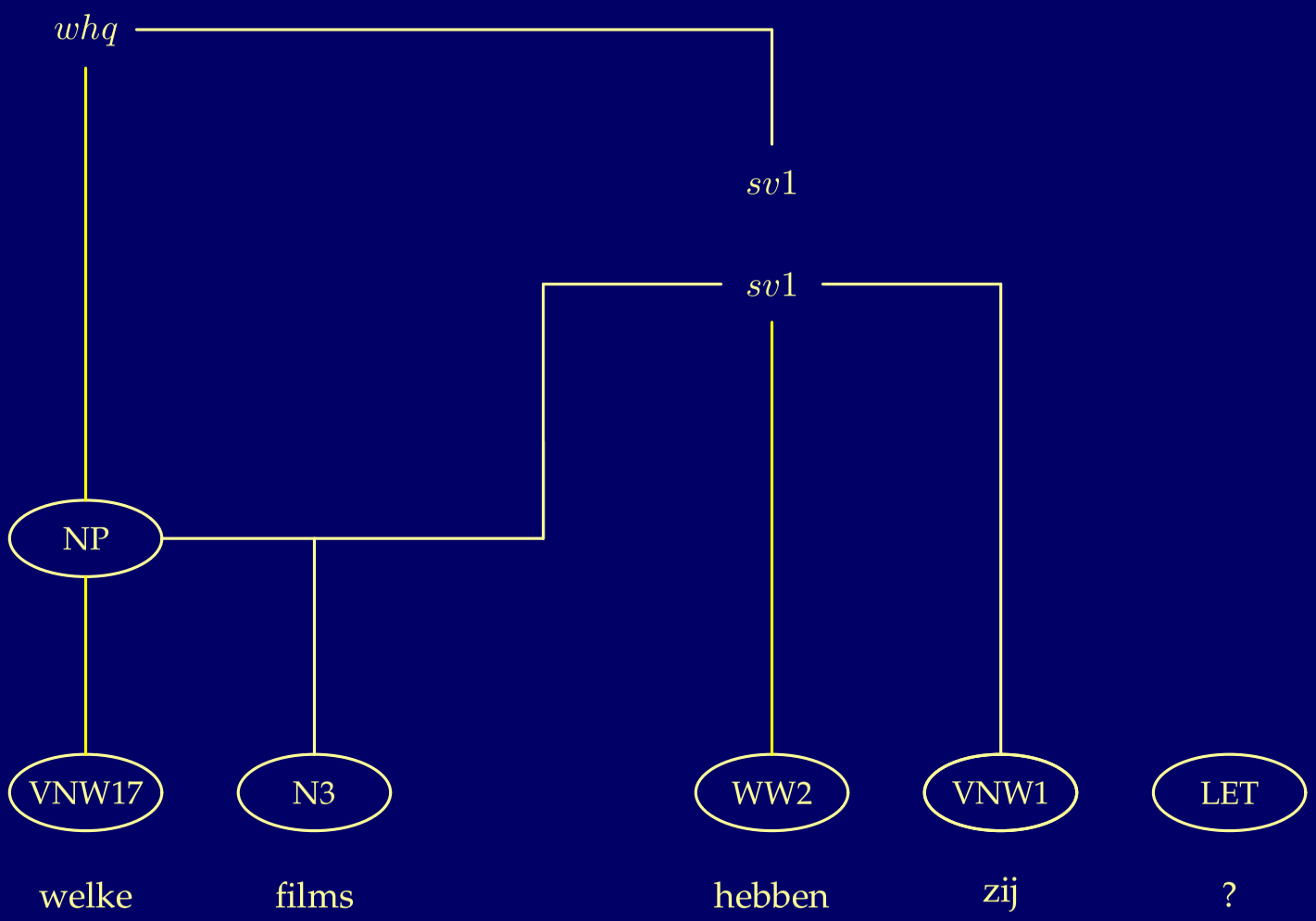
Remove Edge Labels



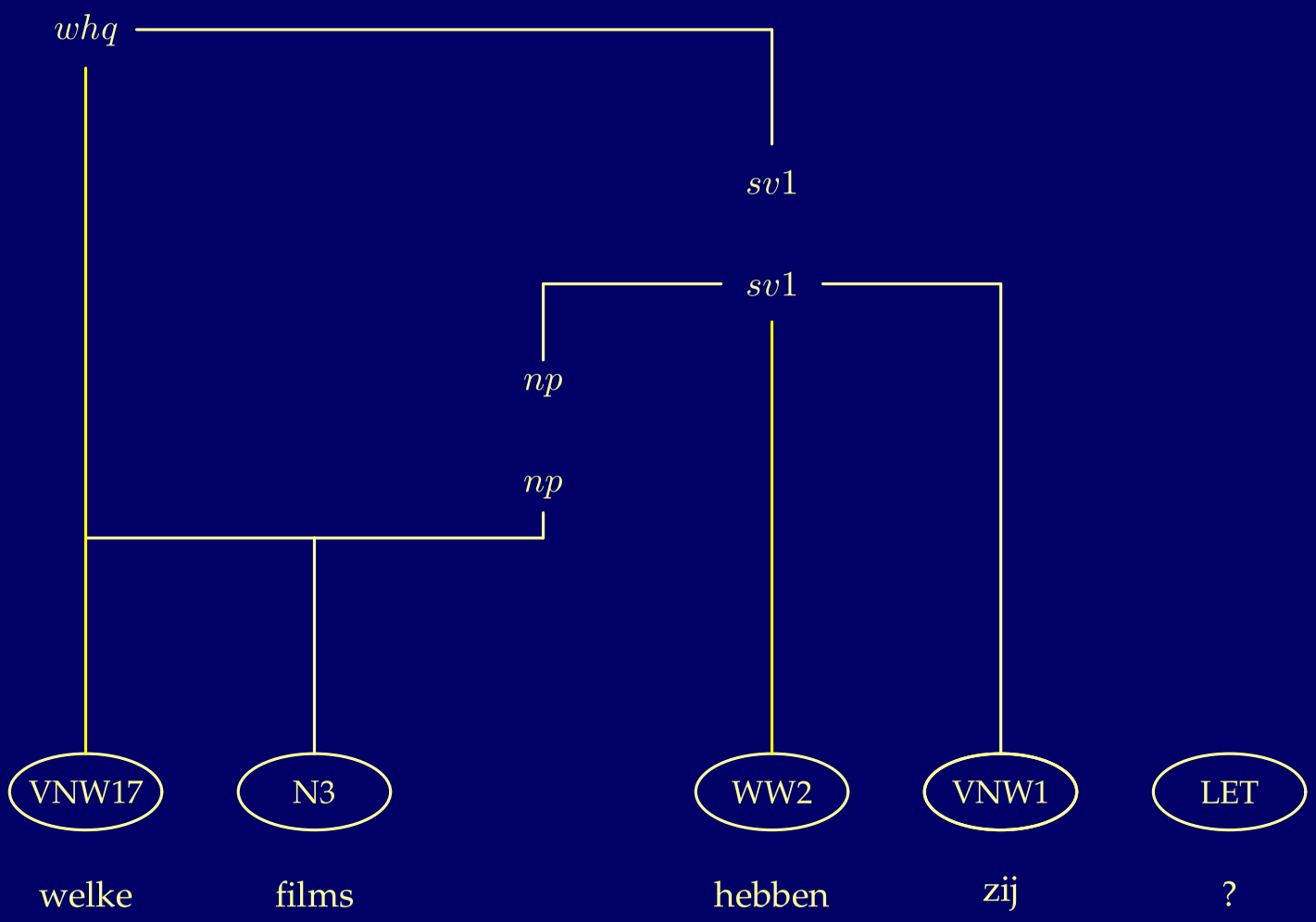
Translation



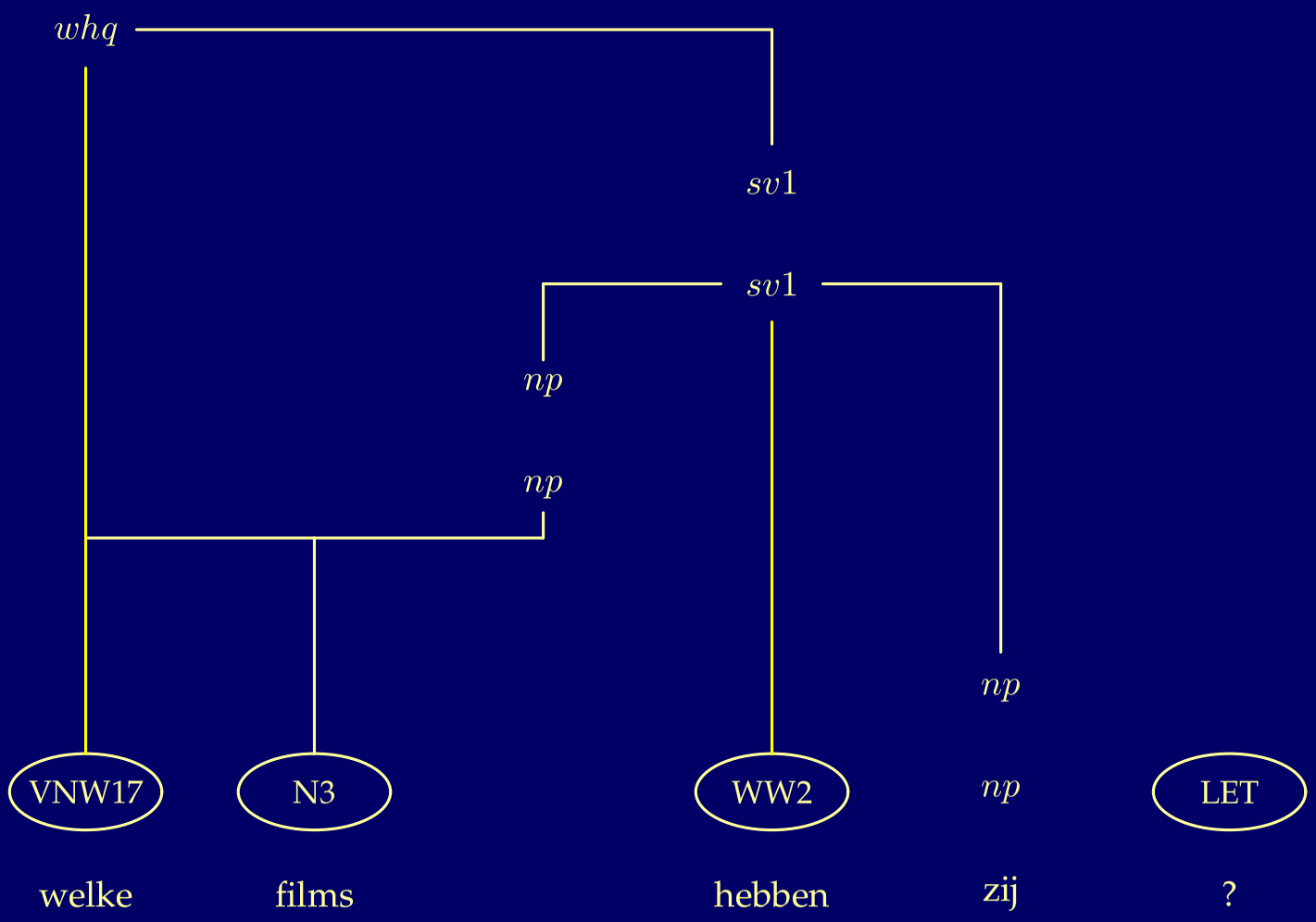
Translation



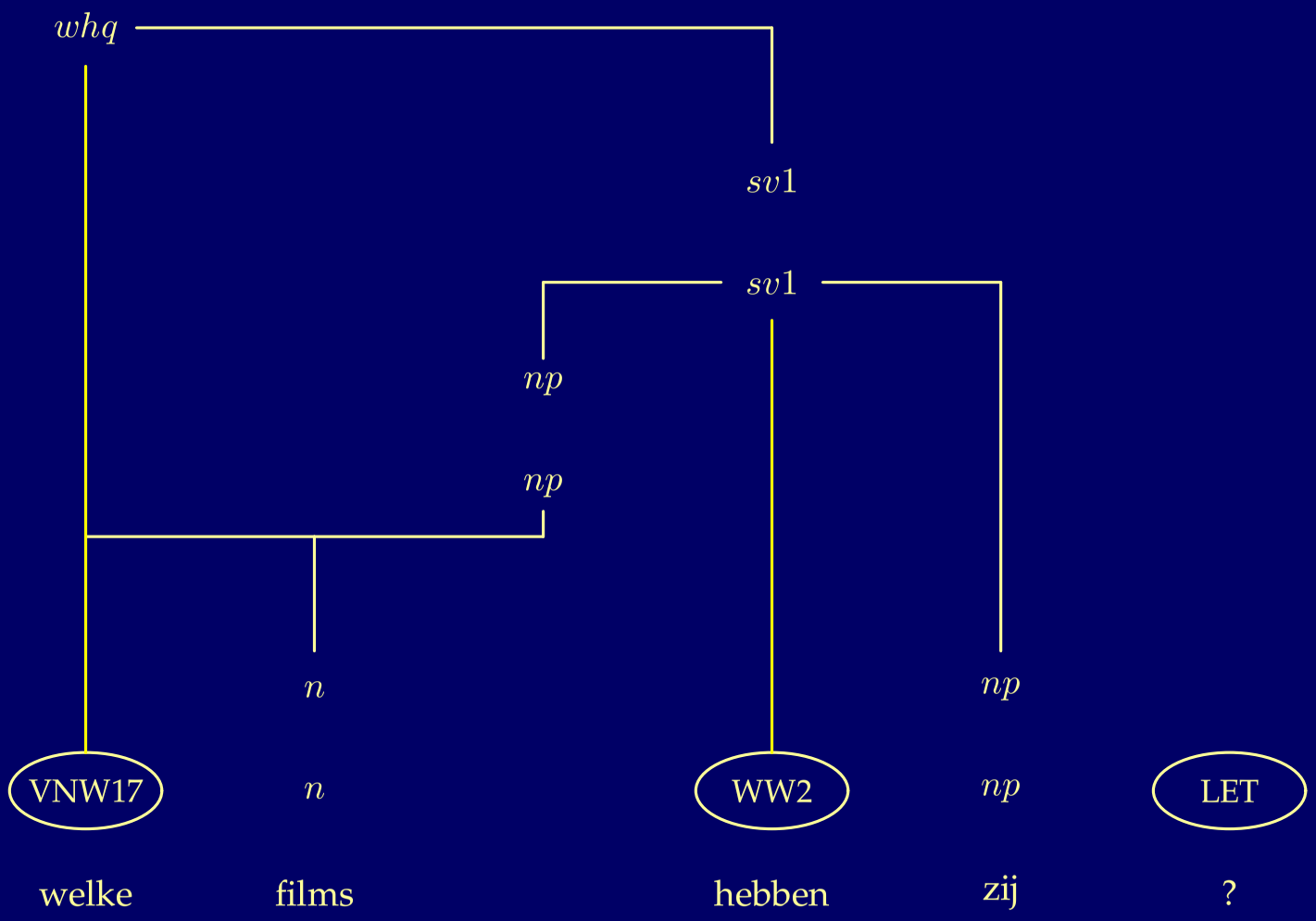
Translation



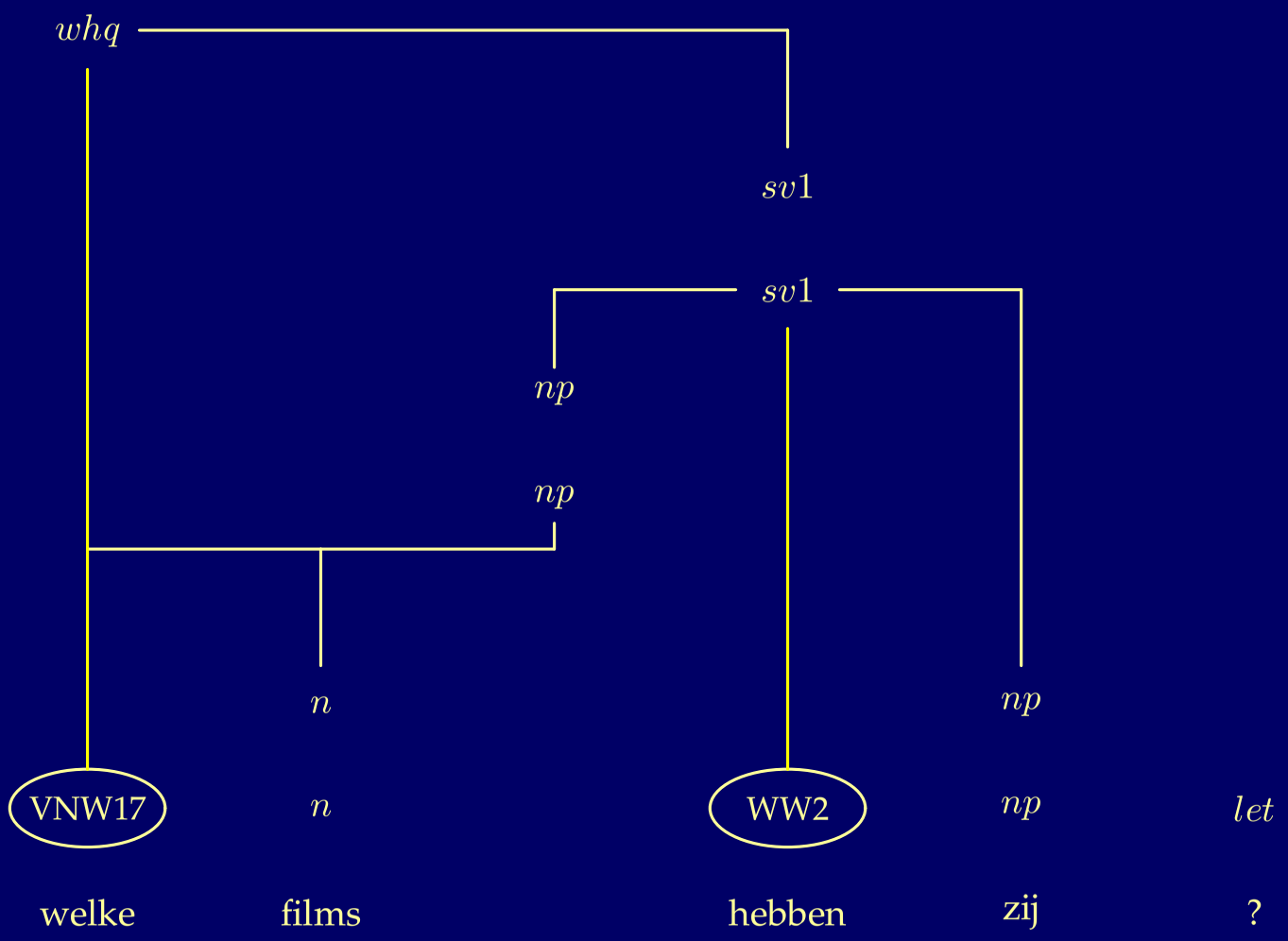
Translation



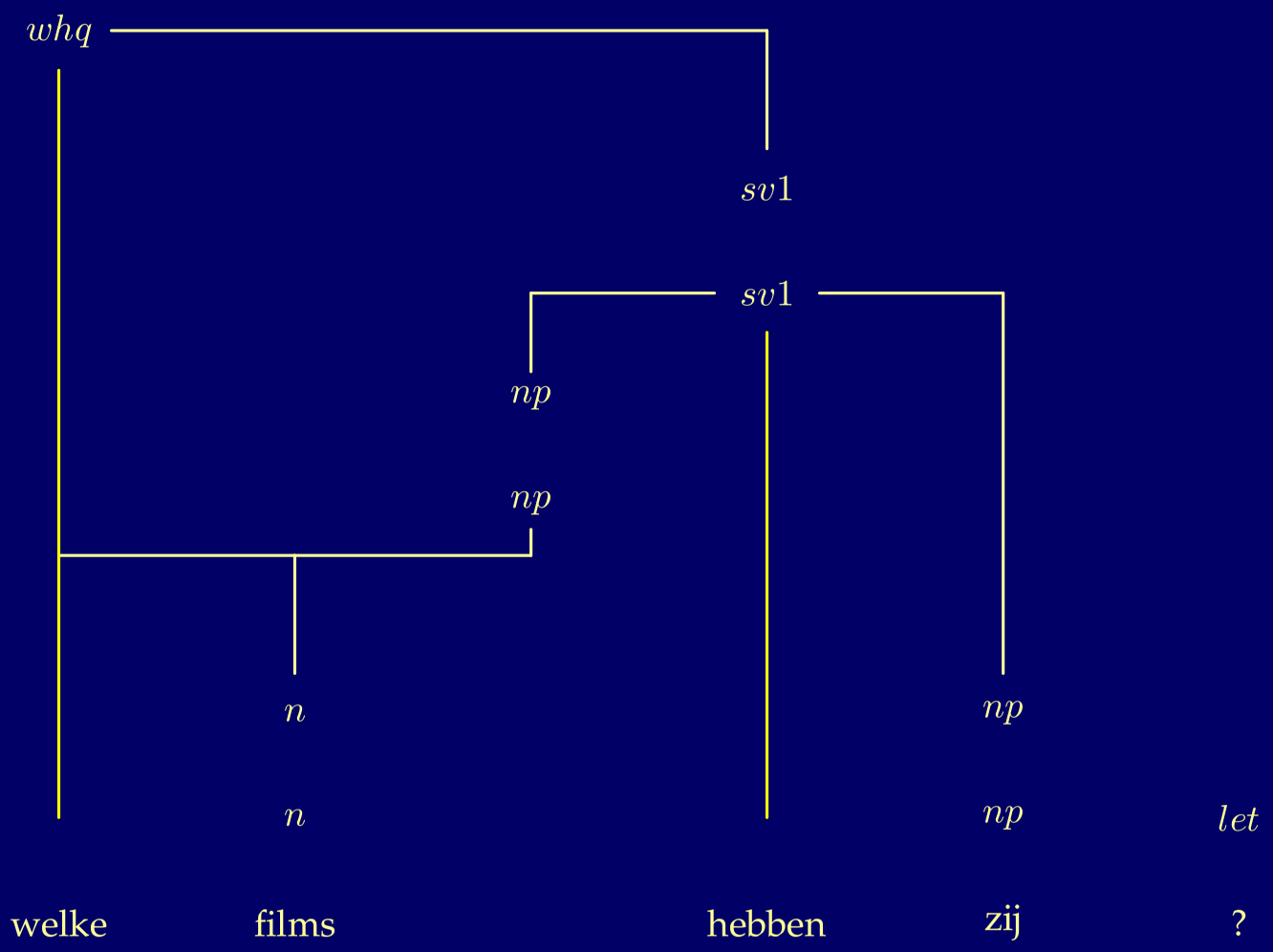
Translation



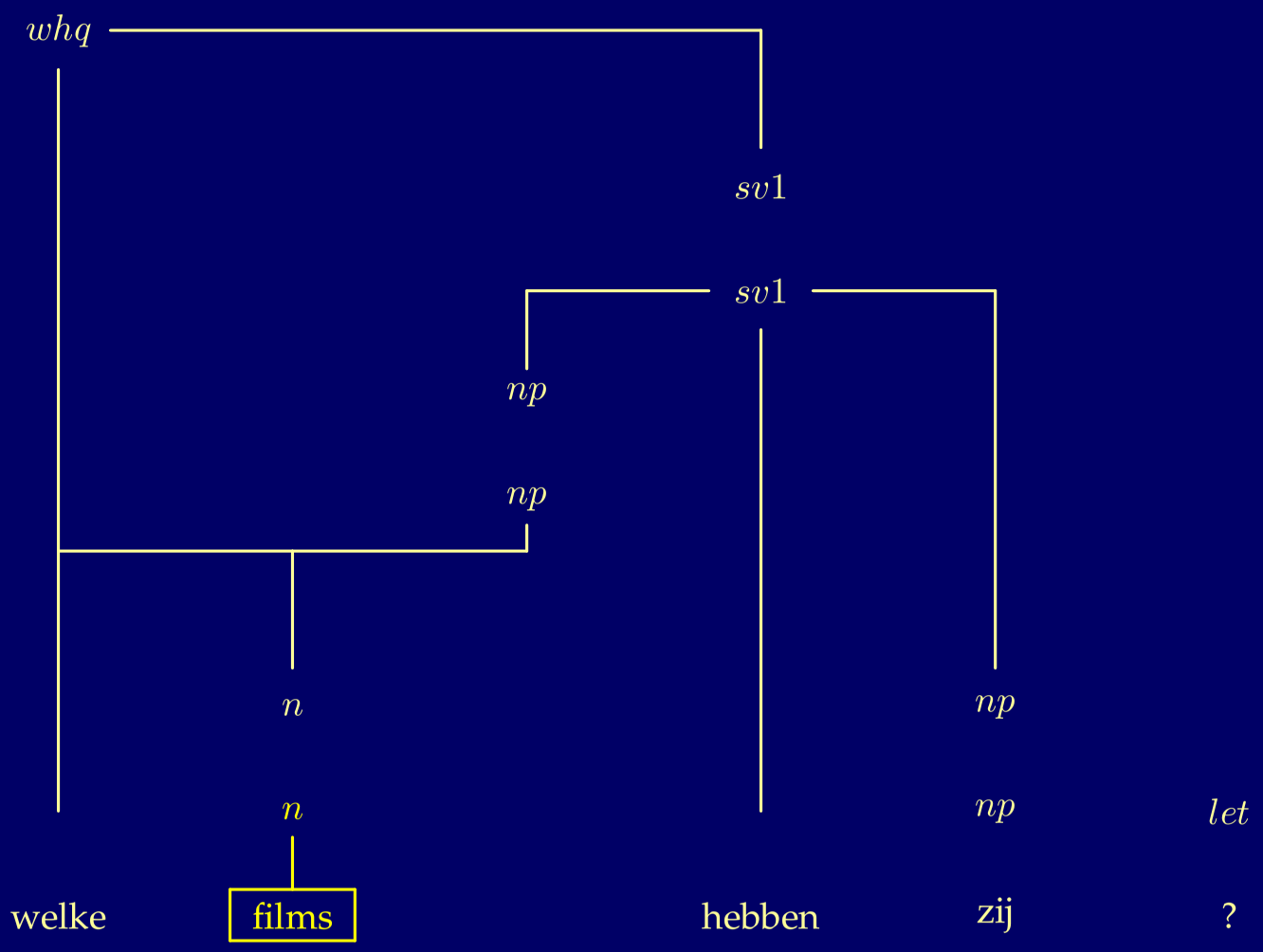
Translation



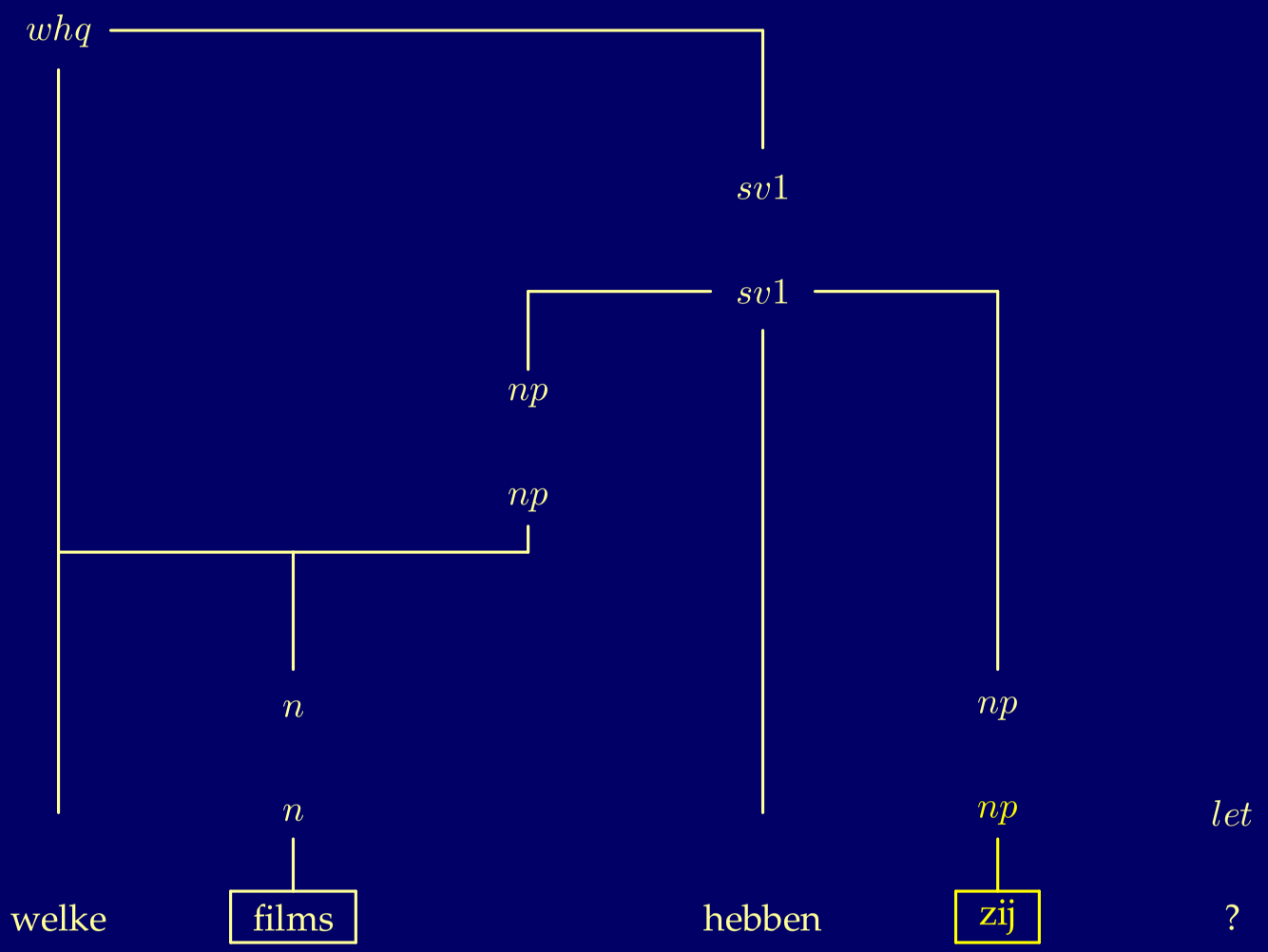
Translation



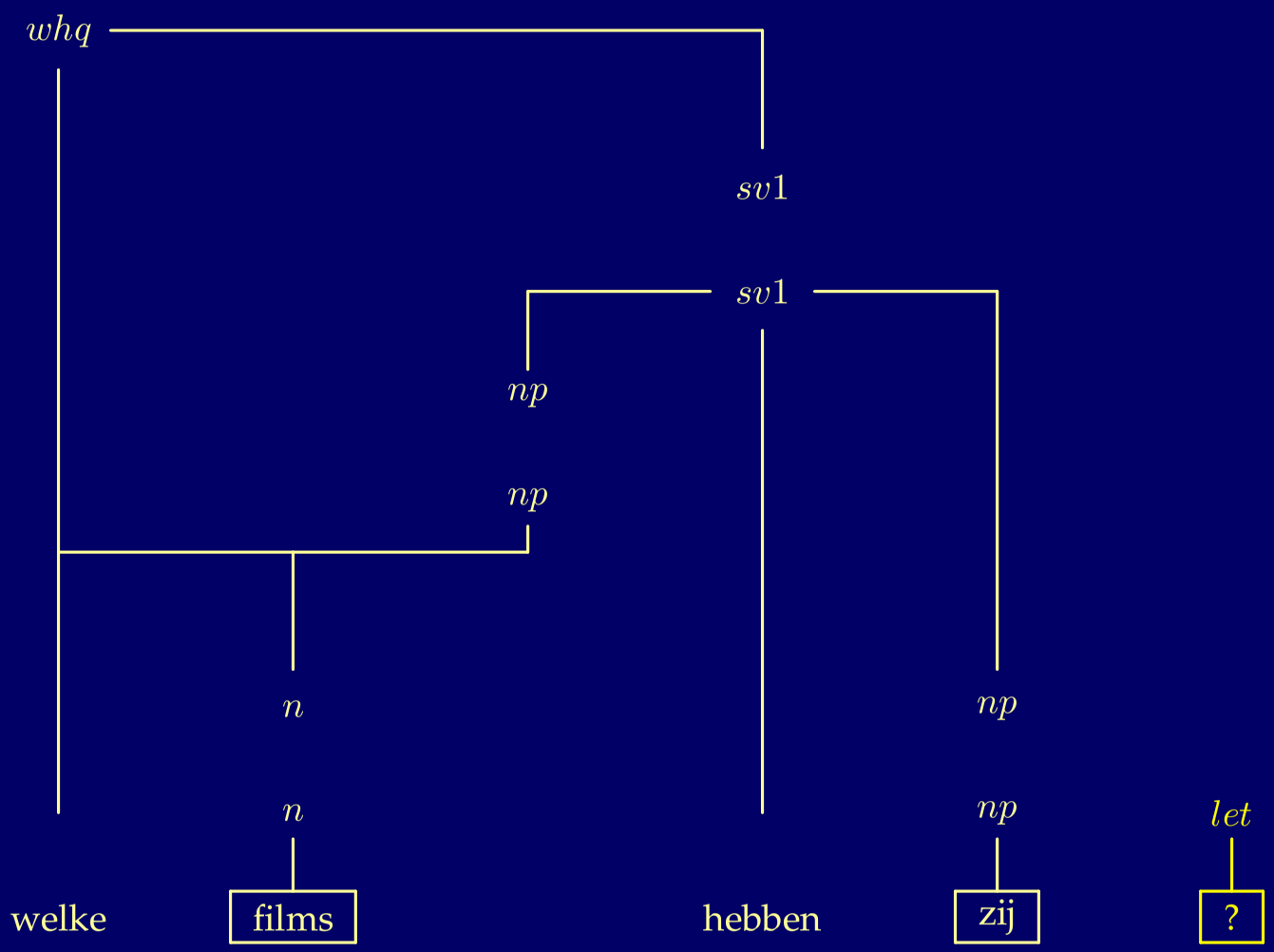
Translation



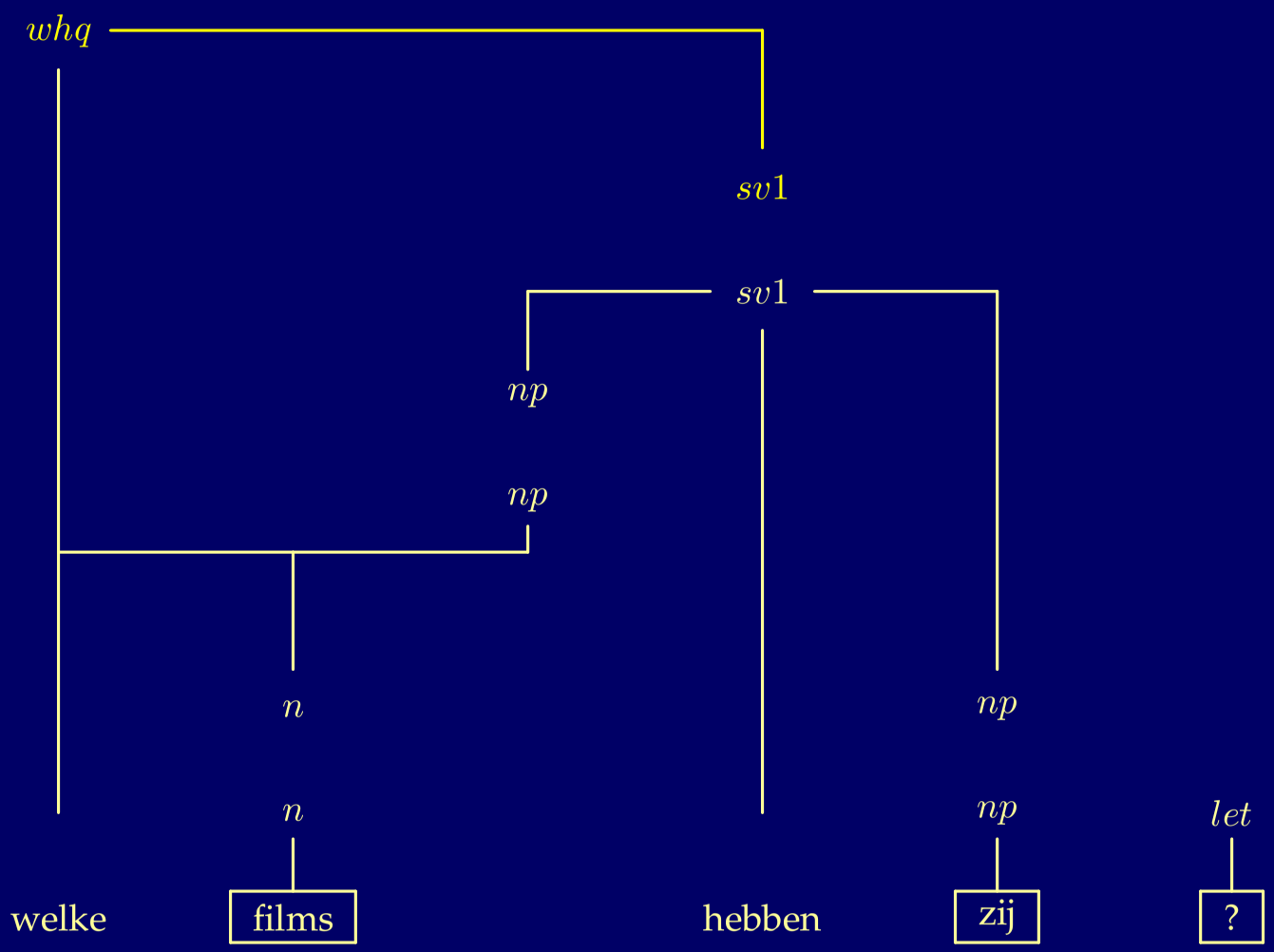
Translation



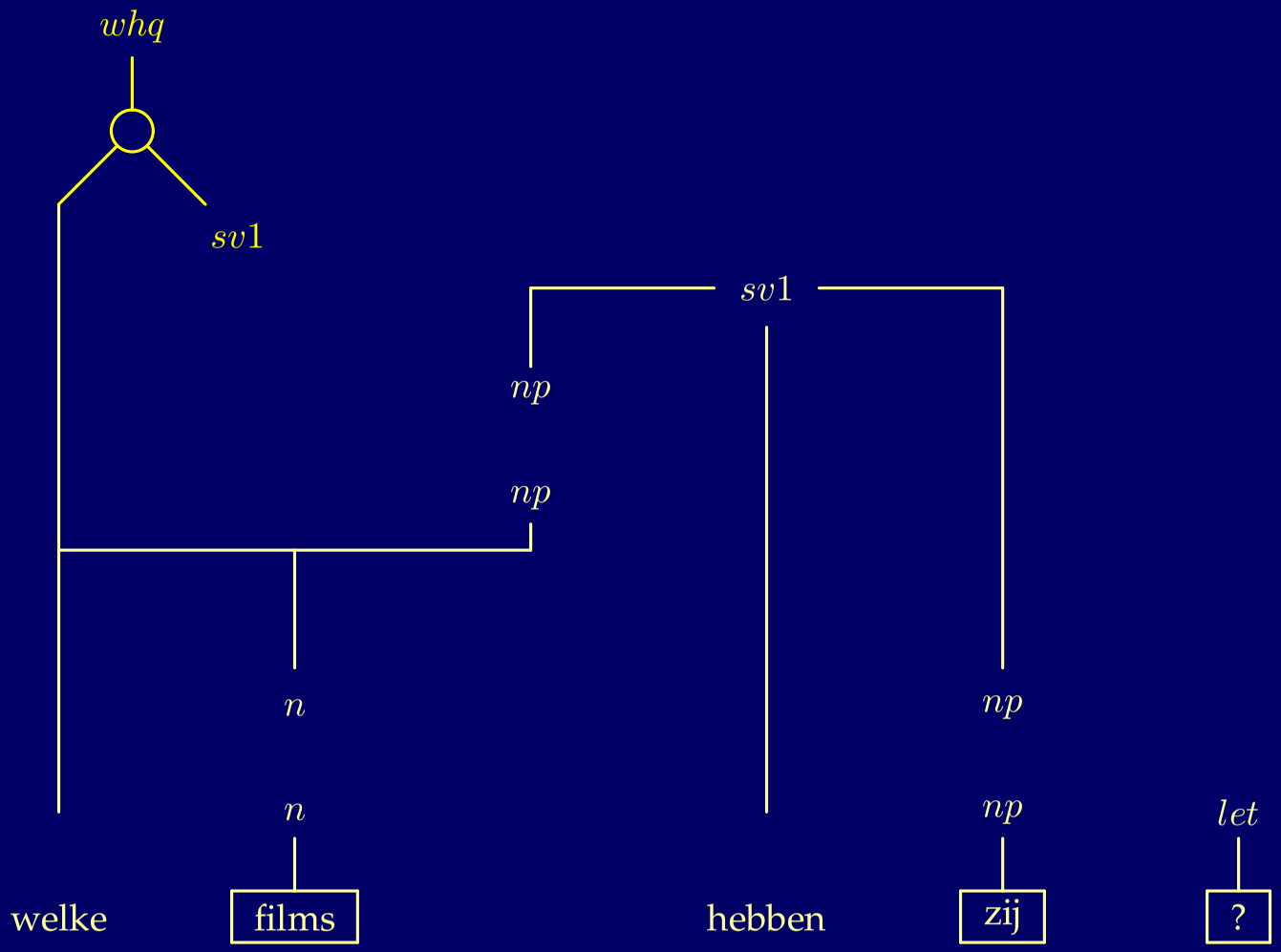
Translation



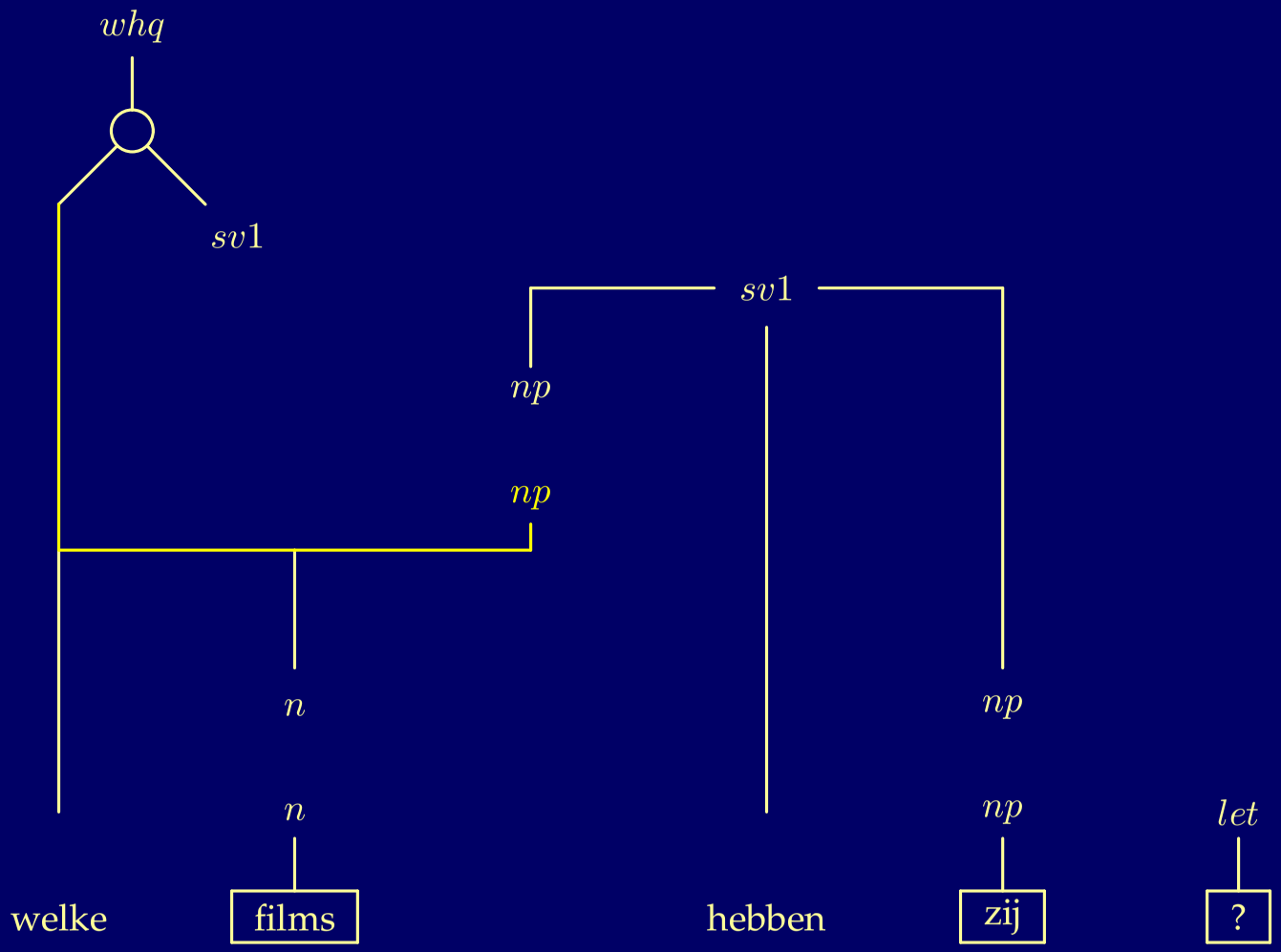
Translation



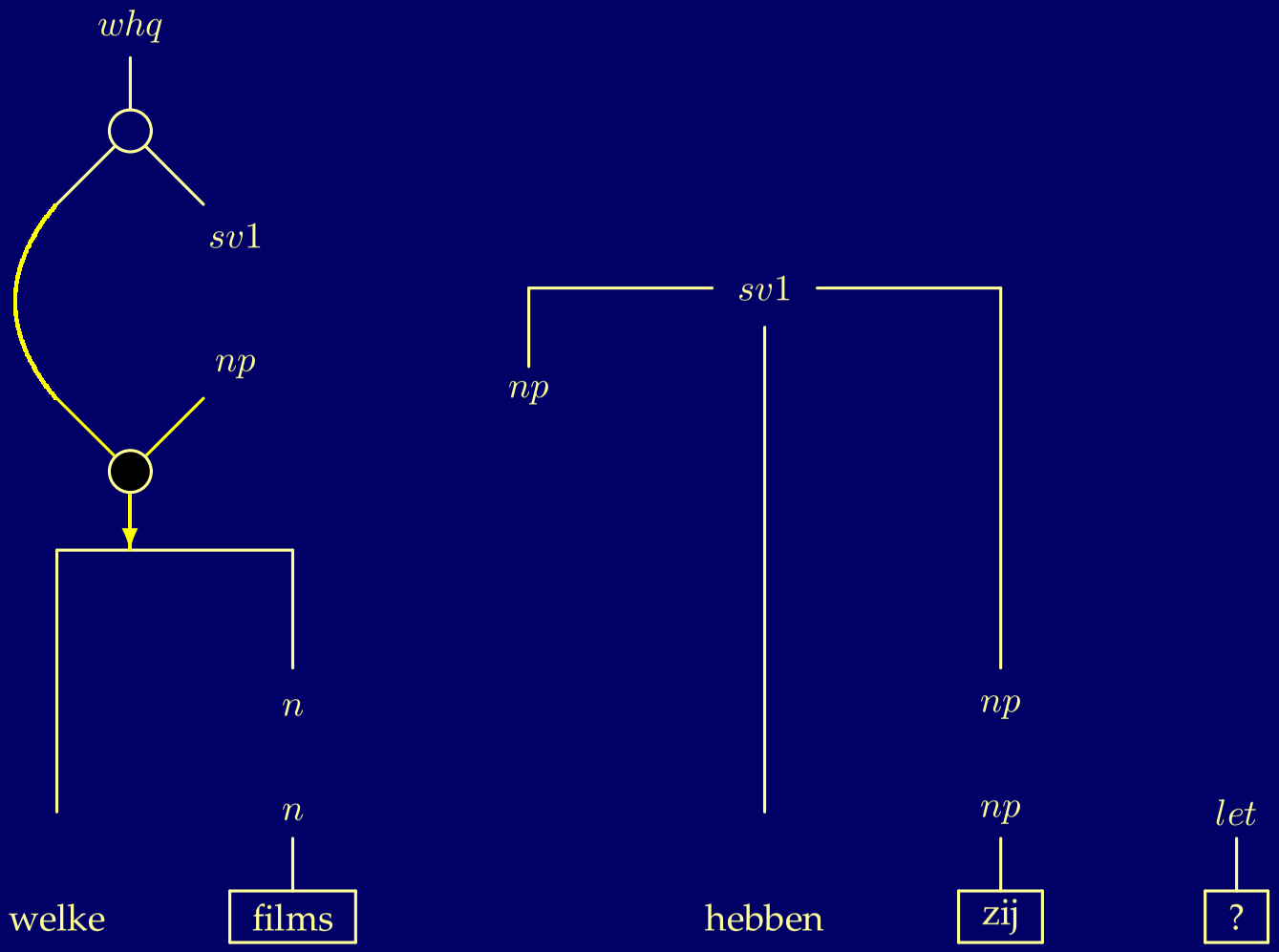
Translation



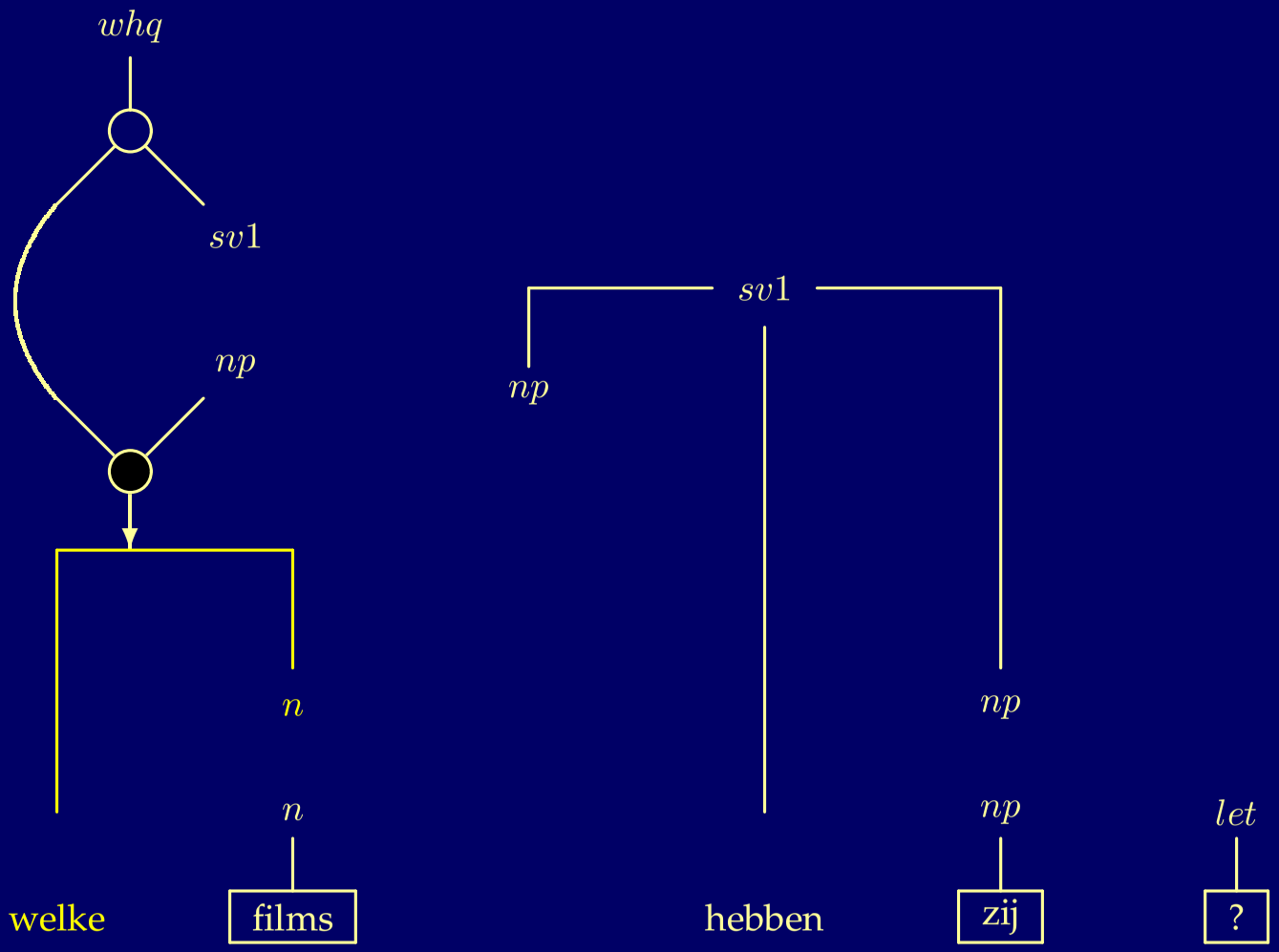
Translation



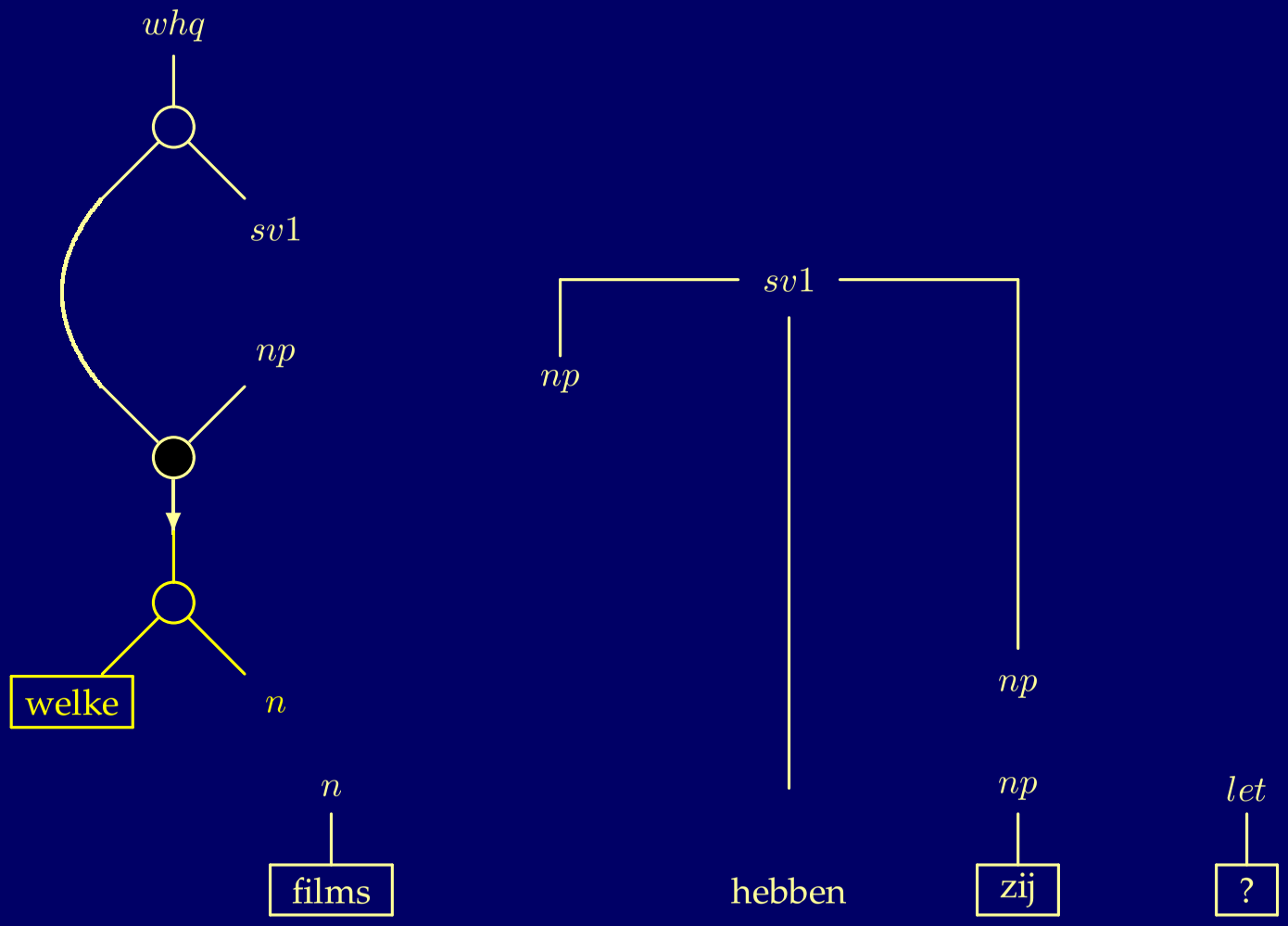
Translation



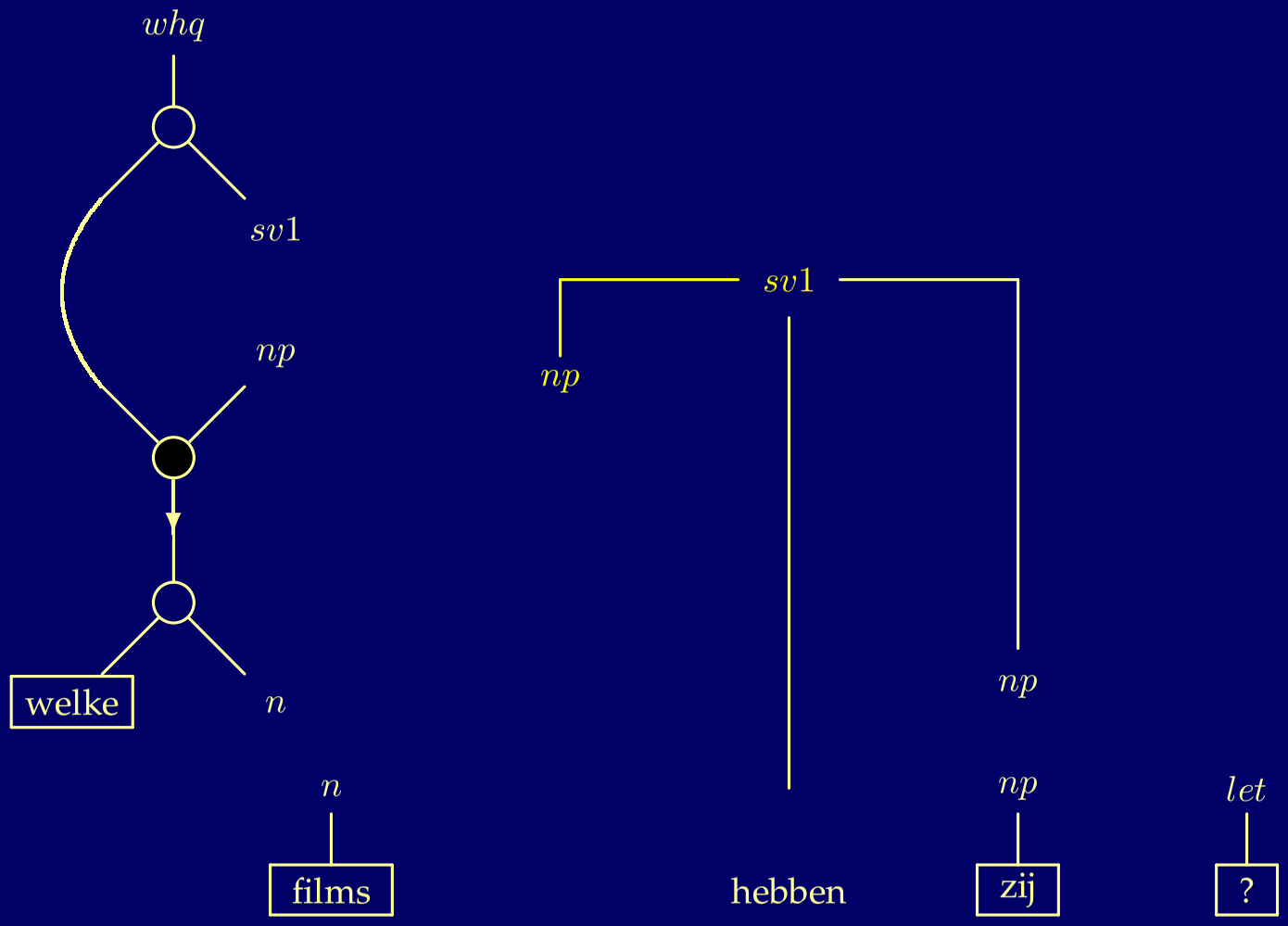
Translation



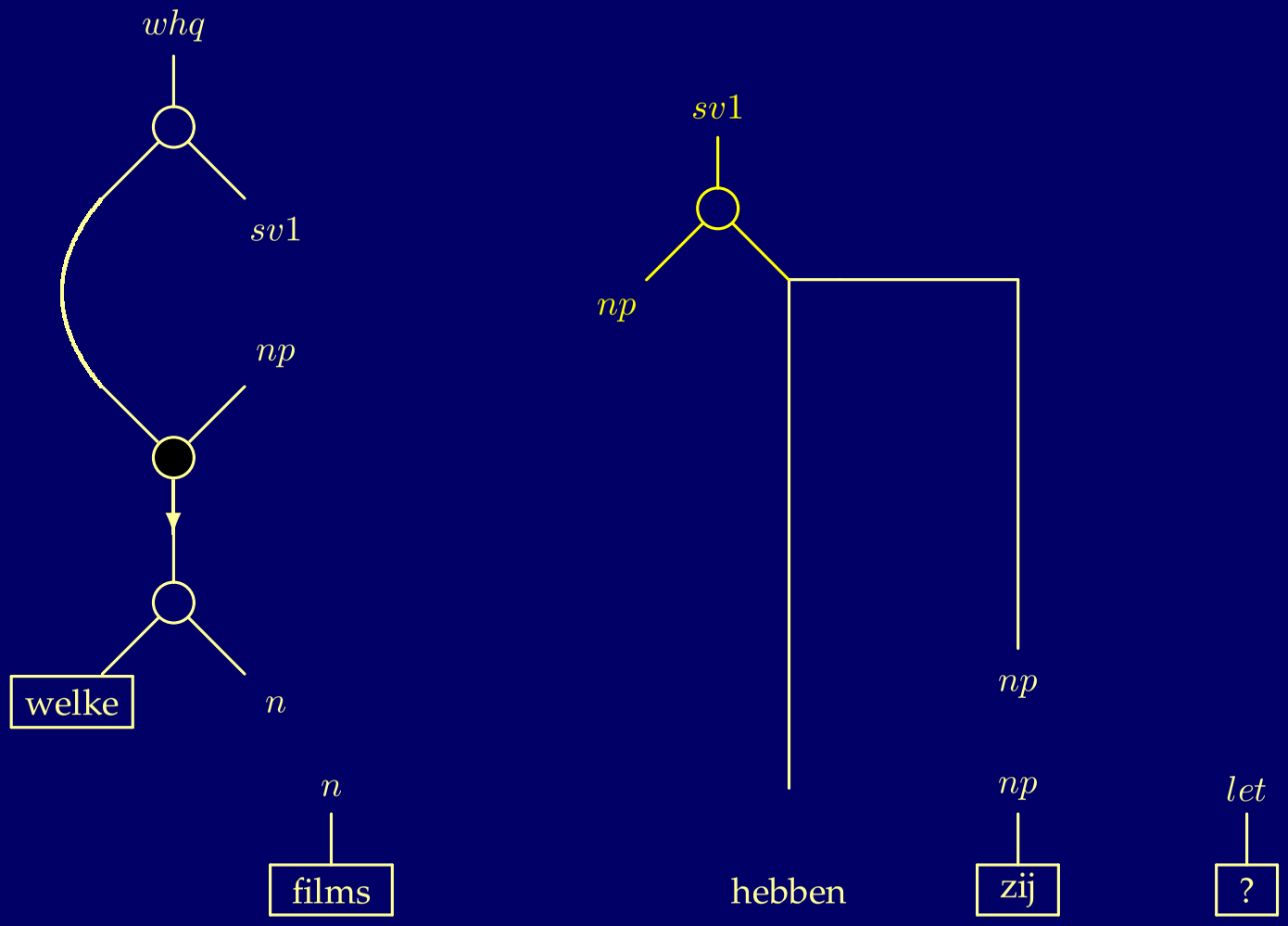
Translation



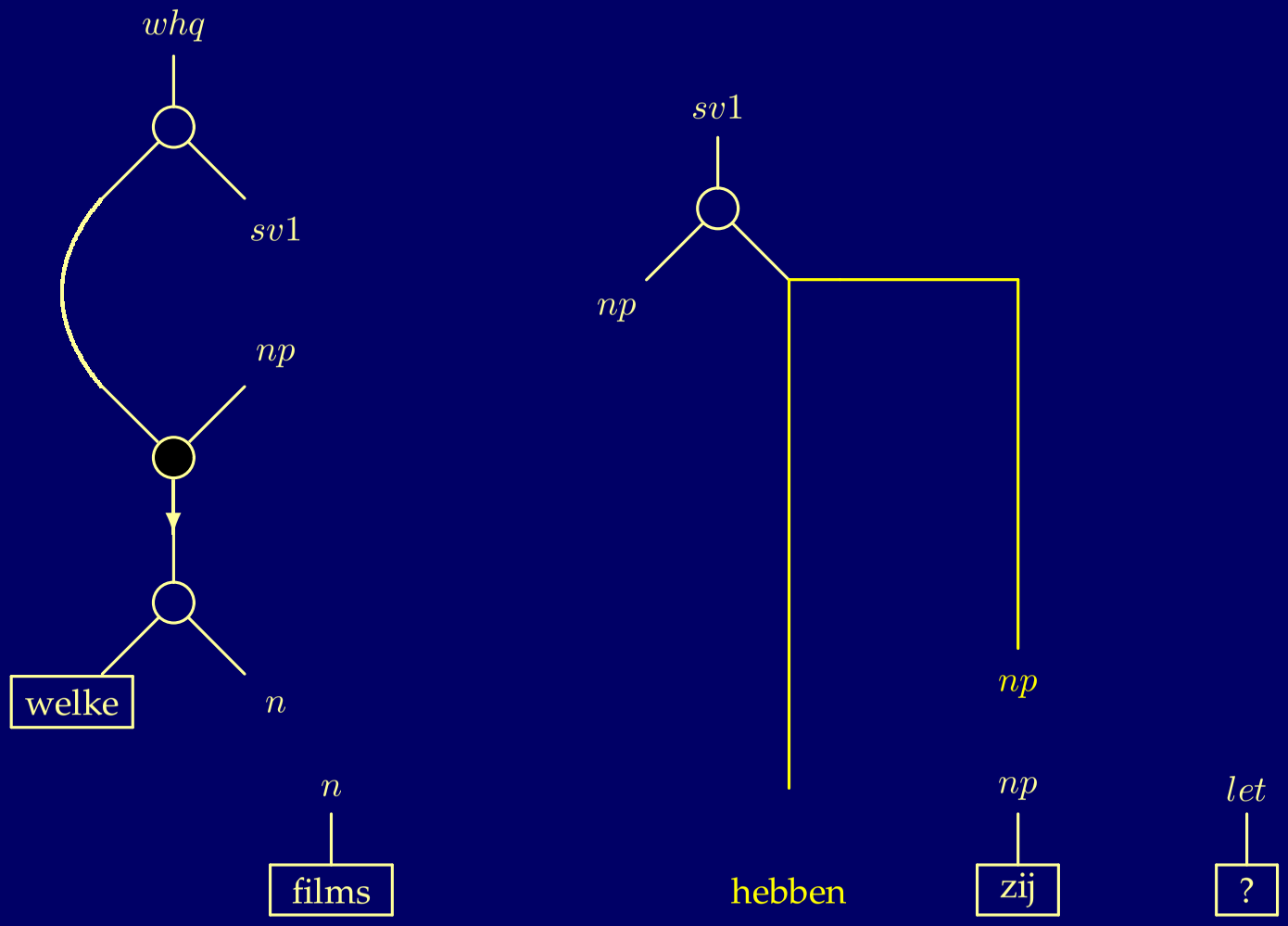
Translation



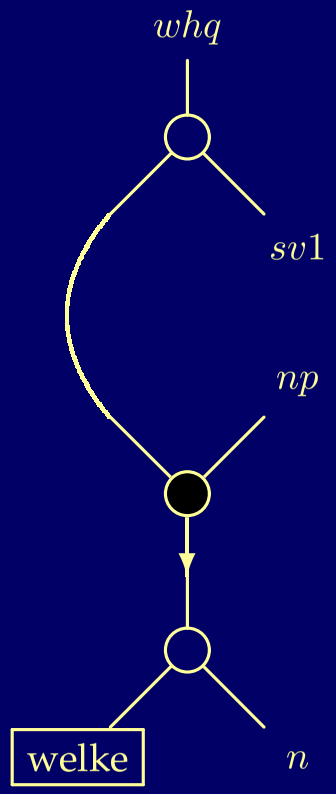
Translation

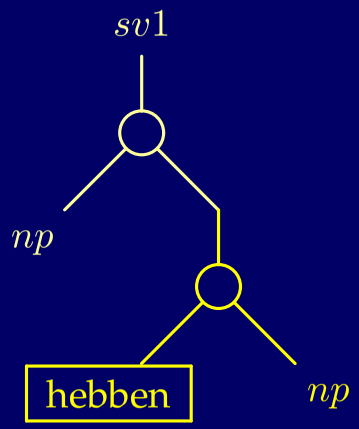


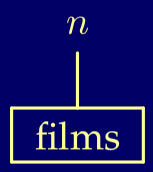
Translation



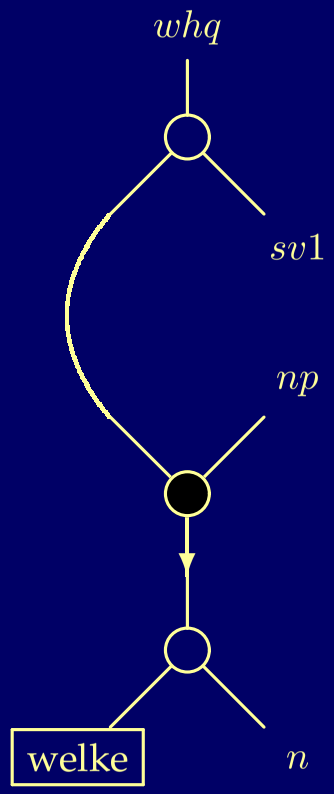
Translation

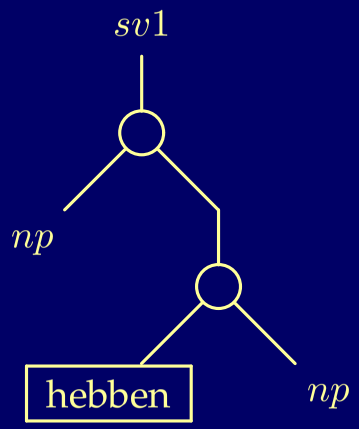


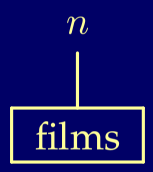




Translation







Refinements

Refinements

Order of Constituents

Refinements

Order of Constituents

- Many verb entries will be generated for different word orders in different sentence types.

Refinements

Order of Constituents

- 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.

Refinements

Order of Constituents

- 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

Treatment of Multiple Dependencies

- Multiple dependencies are translated using auxiliary constructors.

Treatment of Multiple Dependencies

- Multiple dependencies are translated using auxiliary constructors.
- However, the formulas produces by the translation look a bit unfamiliar.

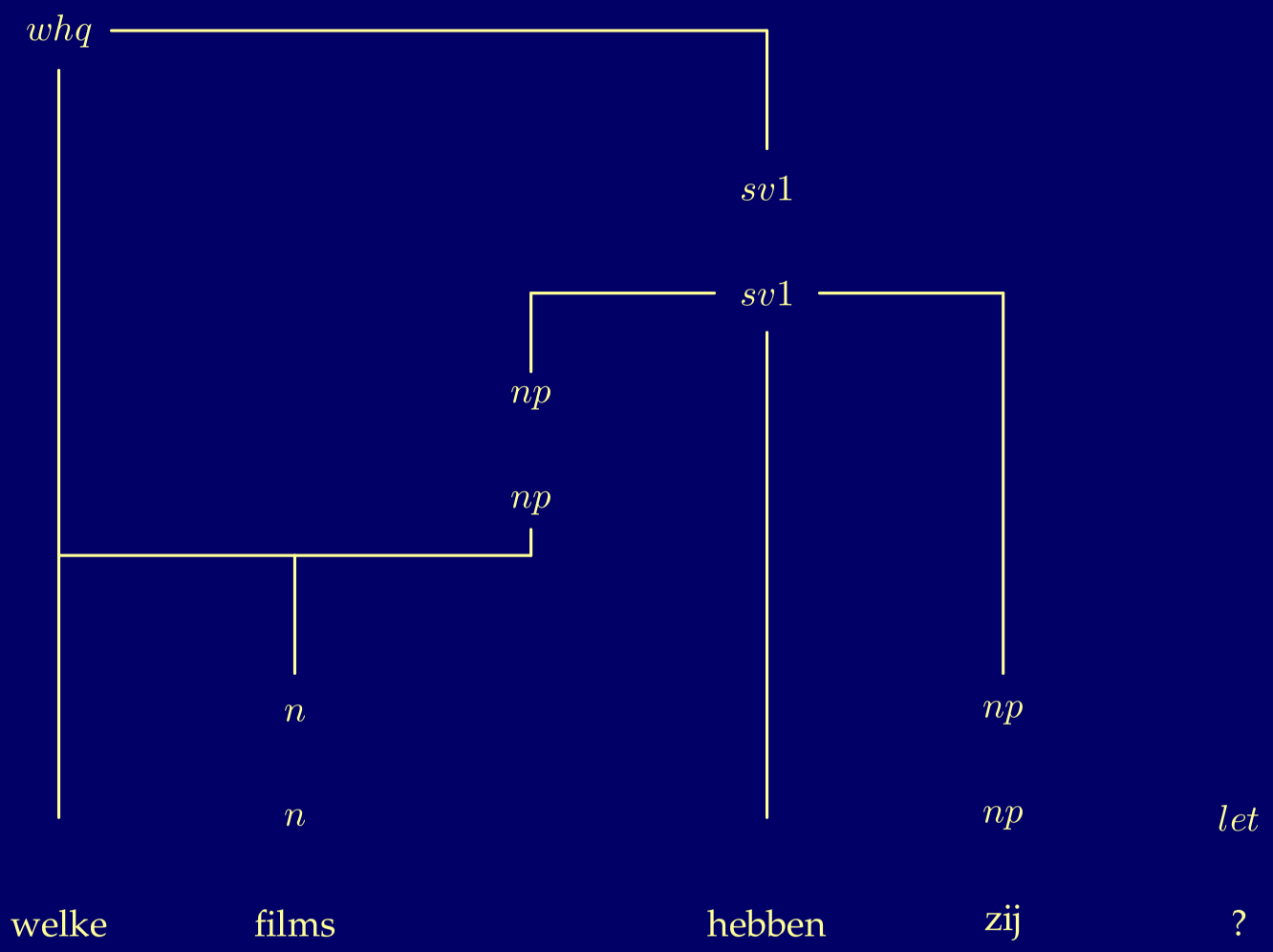
Treatment of Multiple Dependencies

- Multiple dependencies are translated using auxiliary constructors.
- However, the formulas produces by the translation look a bit unfamiliar.
- For example, when we compute the formula assigned to 'welke' it is $((whq/sv1) \bullet np)/n$.

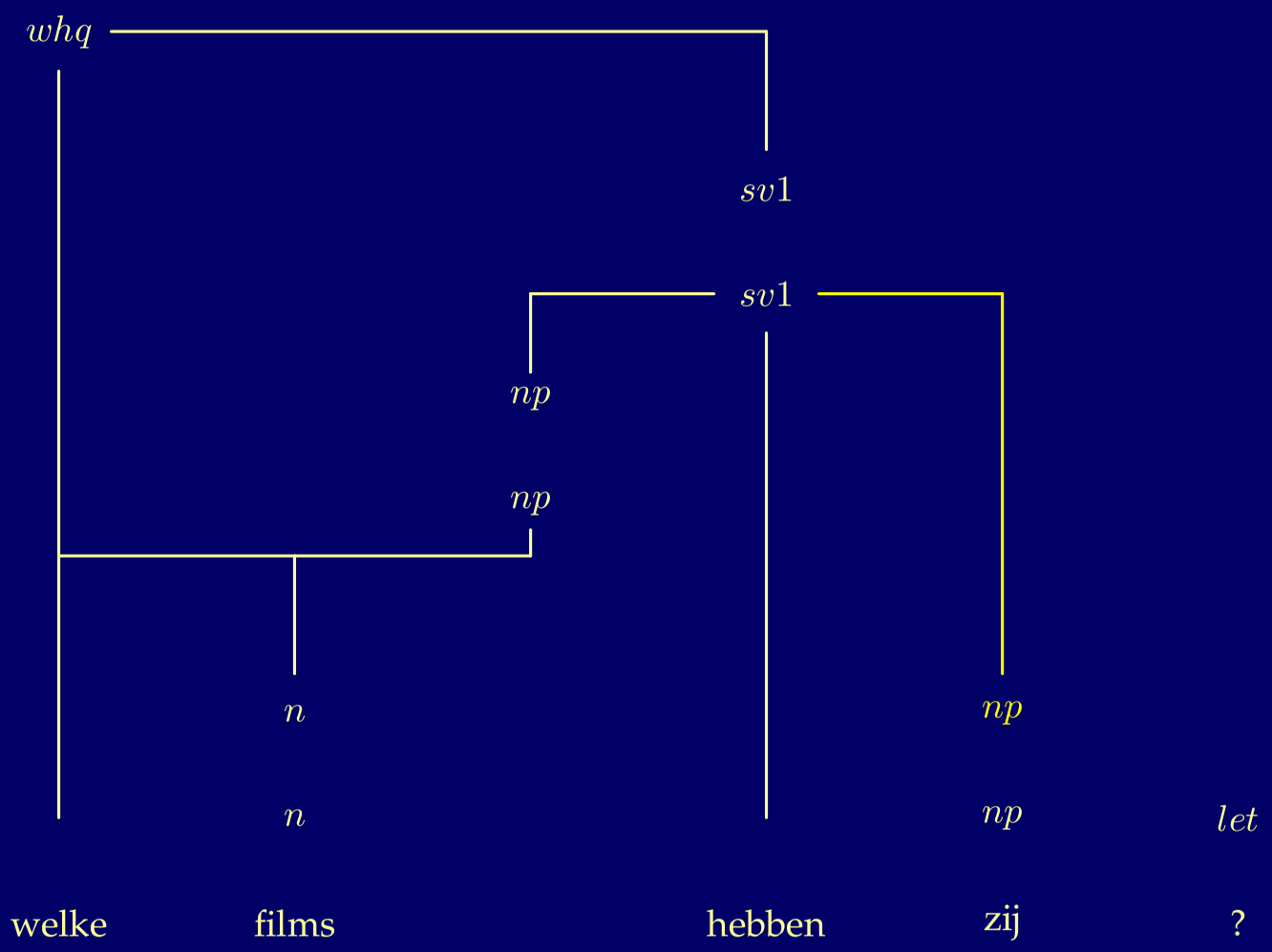
Treatment of Multiple Dependencies

- Multiple dependencies are translated using auxiliary constructors.
- However, the formulas produces by the translation look a bit unfamiliar.
- For example, when we compute the formula assigned to 'welke' it is $((whq/sv1) \bullet np)/n$.
- A solution is to attach these dependencies at the argument category of which the second parent is a child.

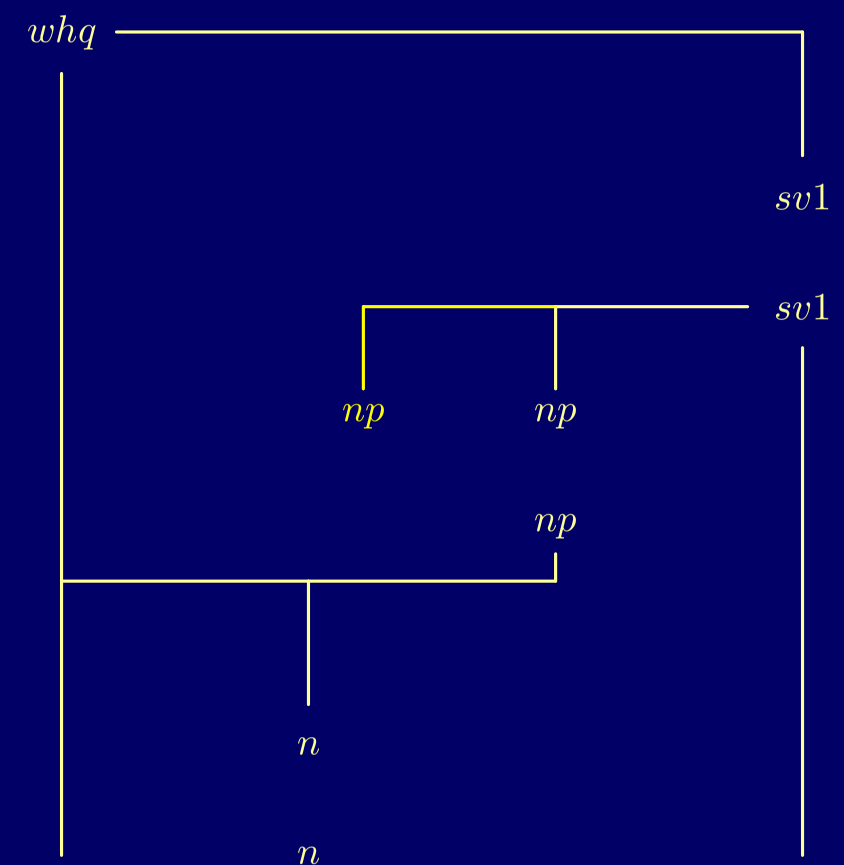
Translation: Revised



Translation: Revised



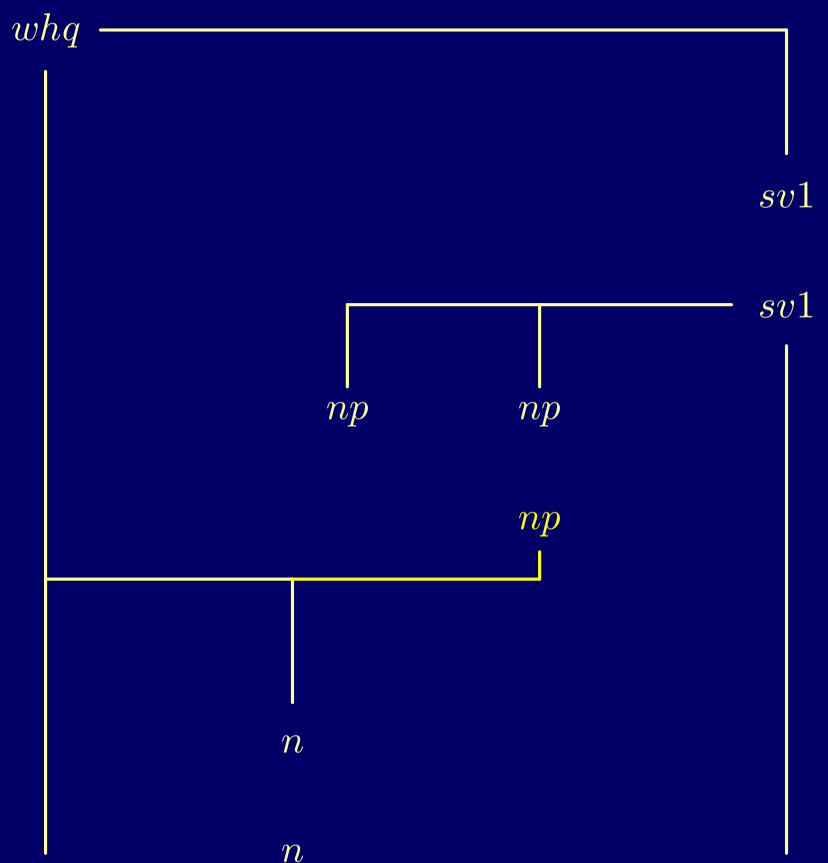
Translation: Revised



welke films hebben zij ?

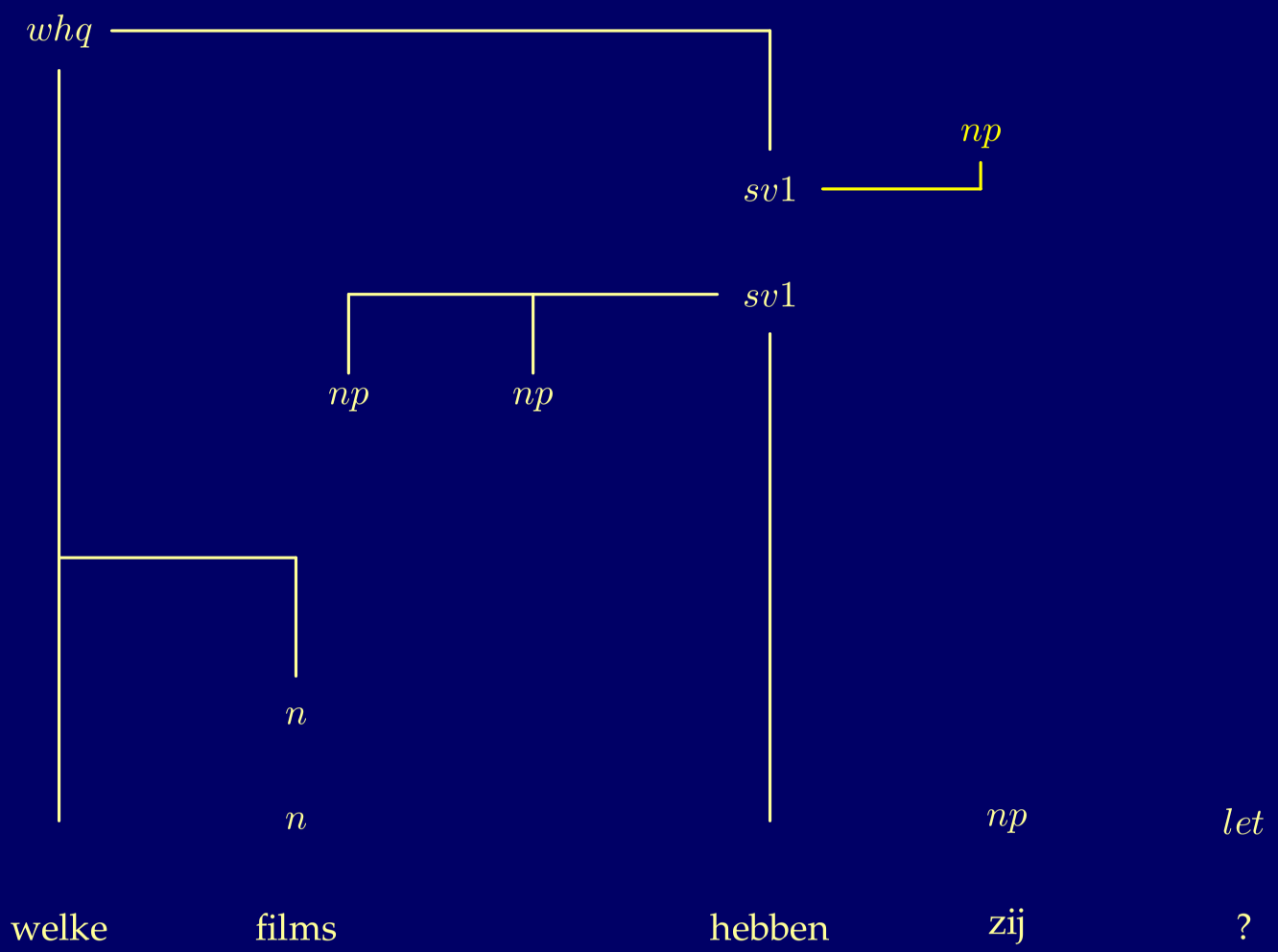
np *let*

Translation: Revised

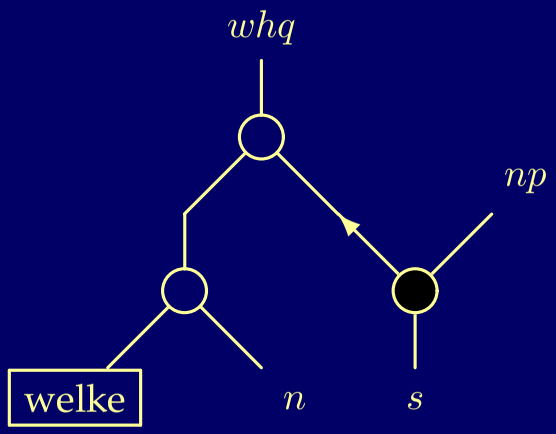


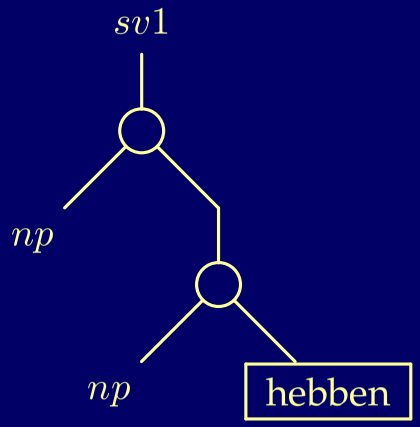
welke films hebben zij ? np let

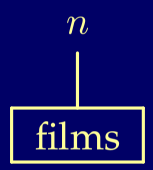
Translation: Revised



Translation







Evaluation

What happens when we parse sentences with the generated lexicon?

Evaluation

What happens when we parse sentences with the generated lexicon?

- The algorithm produces a very large lexicon.

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
  - 2.849 distinct trees,
  - many common words with over 38 lexical entries.

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.

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.
- For large sentence, we have to consider many possible connections

5 Lexical Ambiguity

5 Lexical Ambiguity

Collapsing Lexical Entries

5 Lexical Ambiguity

Collapsing Lexical Entries

- Reduce the size of the lexicon by collapsing similar entries.

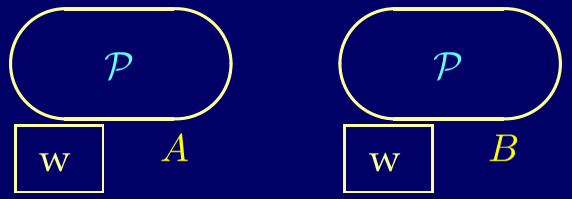
5 Lexical Ambiguity

Collapsing Lexical Entries

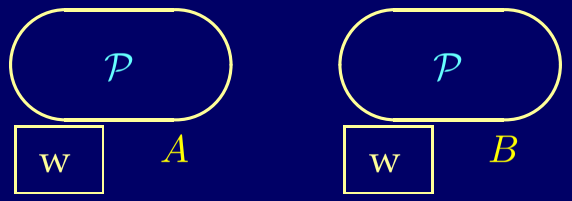
- Reduce the size of the lexicon by collapsing similar entries.
- When are two lexical entries for a word similar?

- Suppose word w has two lexical entries of the following form

- Suppose word w has two lexical entries of the following form

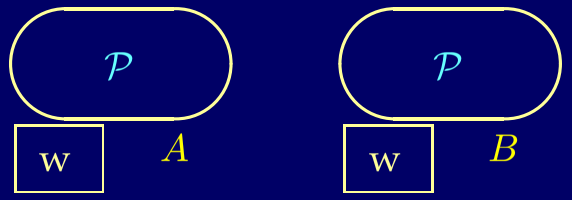


- Suppose word w has two lexical entries of the following form



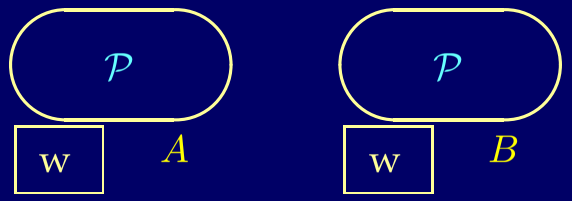
  - $\mathcal{P} \neq \emptyset$

- Suppose word w has two lexical entries of the following form



  - $\mathcal{P} \neq \emptyset$
  - A, B atomic formulas

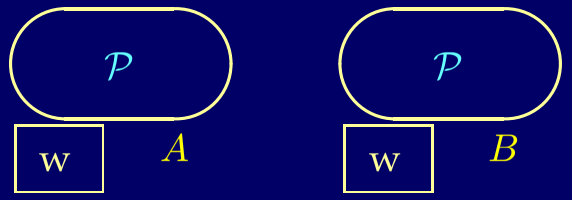
- Suppose word w has two lexical entries of the following form



  - $\mathcal{P} \neq \emptyset$
  - A, B atomic formulas

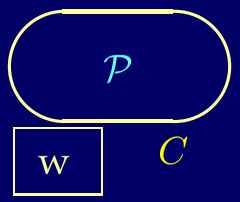
- We want to reduce these entries to a single entry of the form

- Suppose word w has two lexical entries of the following form



  - $\mathcal{P} \neq \emptyset$
  - A, B atomic formulas

- We want to reduce these entries to a single entry of the form



for some formula C.

Radical

Radical

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

Radical

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

$+$ many identifications.

Radical

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

$+$ many identifications.

$-$ shifts part of the lexical complexity to the connection stage.

Conservative

Conservative

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

Conservative

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

Whenever a word has both a $\mathcal{P}[A]$ and a $\mathcal{P}[B]$ entry, we can replace both by a single $\mathcal{P}[C]$.

Conservative

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

Whenever a word has both a $\mathcal{P}[A]$ and a $\mathcal{P}[B]$ entry, we can replace both by a single $\mathcal{P}[C]$.

$+$ no accidental identifications.

Conservative

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

Whenever a word has both a $\mathcal{P}[A]$ and a $\mathcal{P}[B]$ entry, we can replace both by a single $\mathcal{P}[C]$.

$+$ no accidental identifications.

$-$ shift part of the lexical complexity to the contraction stage.

Conservative

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

Whenever a word has both a $\mathcal{P}[A]$ and a $\mathcal{P}[B]$ entry, we can replace both by a single $\mathcal{P}[C]$.

$+$ no accidental identifications.

$-$ shift part of the lexical complexity to the contraction stage.

$-$ identical influence on the complexity of performing the connections as the previous method.

Supertagging

Supertagging

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

Supertagging

- 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).

Supertagging

- 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).
- Results: faster parsing, reasonable accuracy (92.2%).

Supertagging

- 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).
- Results: faster parsing, reasonable accuracy (92.2%).
- Extensible to also take dependency information into account.

Supertagging

- 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).
- Results: faster parsing, reasonable accuracy (92.2%).
- Extensible to also take dependency information into account.
- Can be extended to other lexicalized grammar formalisms.

Some Differences

Some Differences

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

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 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

- 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 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.

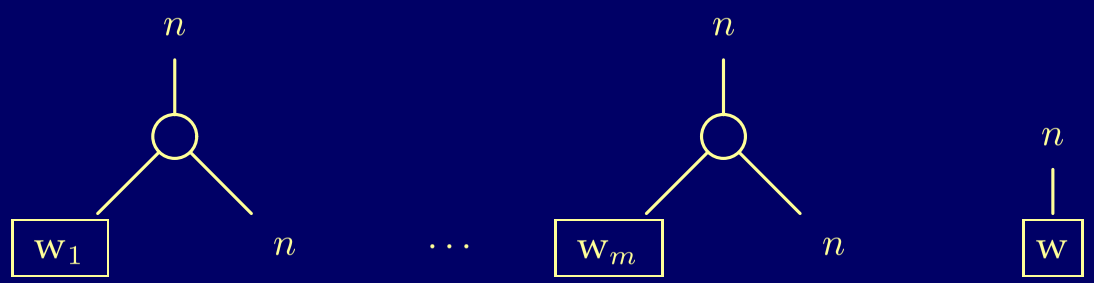
6 Connecting Formulas

6 Connecting Formulas

The Problem

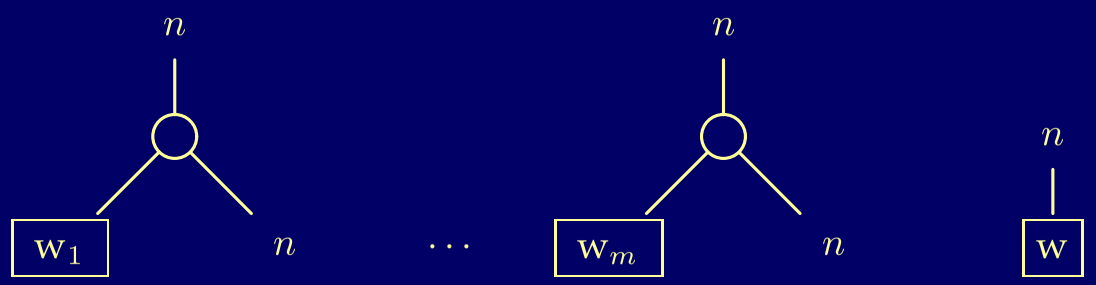
6 Connecting Formulas

The Problem



6 Connecting Formulas

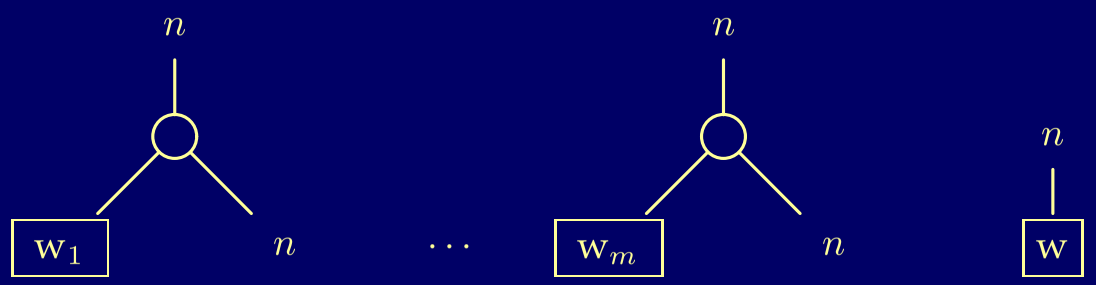
The Problem



$m!$ different ways of performing the connections.

6 Connecting Formulas

The Problem



$m!$ different ways of performing the connections.

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

Weighted Links

Weighted Links

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

Weighted Links

- 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 lexical entry

Weighted Links

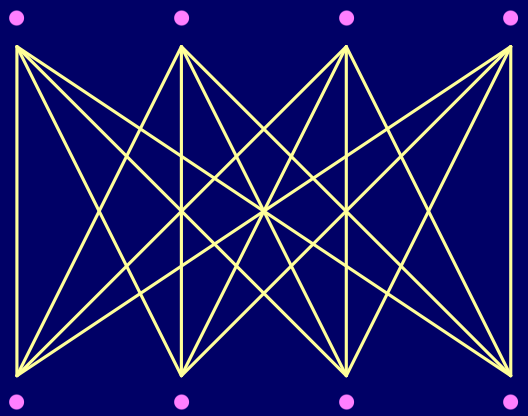
- Suppose, however, that we assign a weight to every possible connection.
- A weight function could be based on our tree-bank data:
  - given an atomic formula in a lexical entry
  - we keep track of the distance to the formula it is linked to when we disconnect it.

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 lexical entry
  - we keep track of the distance to the formula it is linked to when we disconnect it.
- Other weight function are also possible.

Weighted Bipartite Graphs

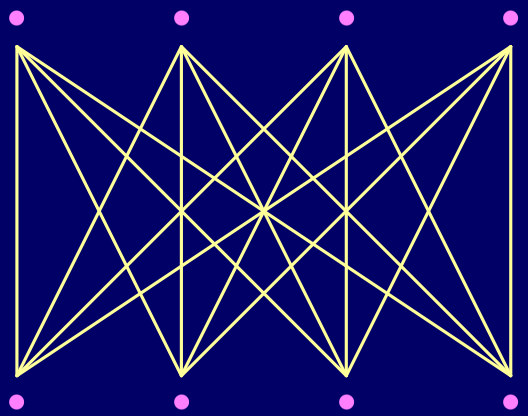
Weighted Bipartite Graphs



positive formulas

negative formulas

Weighted Bipartite Graphs

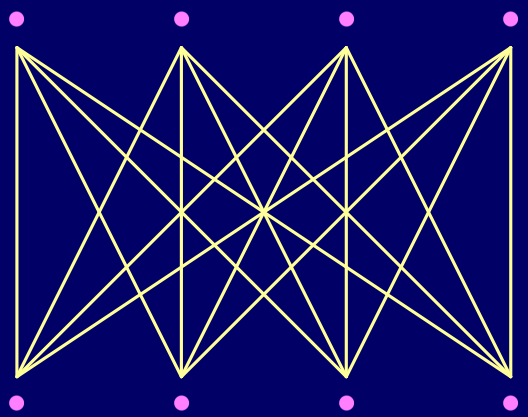


positive formulas

negative formulas

We have polynomial algorithms to:

Weighted Bipartite Graphs



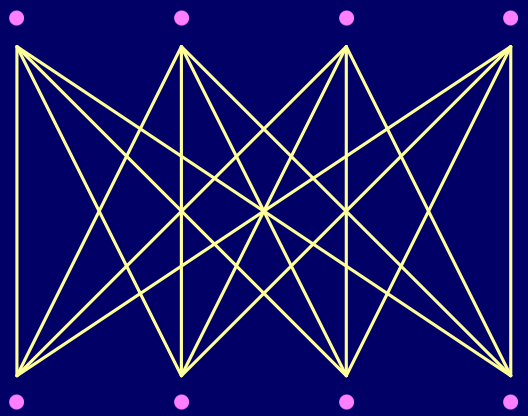
positive formulas

negative formulas

We have polynomial algorithms to:

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

Weighted Bipartite Graphs



positive formulas

negative formulas

We have polynomial algorithms to:

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

7 Conclusions

7 Conclusions

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

7 Conclusions

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

7 Conclusions

- We can automatically extract a large type-logical treebank from the CGN syntactic annotation.
- This produces rather unwieldy type-logical grammars:
  - massive lexical ambiguity,

7 Conclusions

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

7 Conclusions

- We can automatically extract a large type-logical treebank from the CGN syntactic annotation.
- This produces rather unwieldy type-logical grammars:
  - massive lexical ambiguity,
  - a prohibitive number of possible connections.
- We have sketched how statistical methods may help us solve both problems.

References

Chegireddy, C. R. & Hamacher, H. W. (1987), 'Algorithms for finding k-best perfect matchings', *Discrete Applied Mathematics* **18**, 155–165.

Joshi, A. & Srinivas, B. (1994), Disambiguation of super parts of speech (or supertags): Almost parsing, *in* 'Proceedings of the 17th International Conference on Computational Linguistics', Kyoto.

Kuhn, H. W. (1955), 'The hungarian method for the assignment problem', *Naval Research Logistics Quarterly* **2**, 83–97.

Contents