

Lambek Grammars and Hyperedge Replacement Grammars

Richard Moot

LaBRI (CNRS), SIGNES (INRIA Bordeaux SW), Bordeaux University

50 Years Syntactic Calculus, Chieti, 10-11 July 2008

Introduction

1958: The Syntactic Calculus

The aim of this paper is to obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages such as English, or at least for fragments of such languages.

Lambek (1958) ‘On the Calculus of Syntactic Types’

Introduction

The Syntactic Calculus — Good News

- Decidable (Lambek 1958),
- A choice of proof systems, well-understood and with good properties, eg. cut-elimination (Lambek 1958)
- Transparent link to Montague-style lambda-term semantics through the Curry-Howard Isomorphism (van Benthem 1987),
- Effective algorithms which learn grammars from annotated structures (Buszkowski & Penn 1990, Kanazawa 1998),
- Parsing Lambek grammars can be transparently related to incremental sentence-processing by humans (Johnson 1998, Morrill 1998).

Introduction

The Syntactic Calculus — Bad News

Theorem (Pentus 1995, Pentus 1997)

(product-free) Lambek Grammars are Context-Free.

Theorem (Pentus 2003, Pentus 2006)

Lambek Calculus is NP-complete.

Introduction

The Syntactic Calculus — Bad News

Theorem (Pentus 1995, Pentus 1997)

(product-free) Lambek Grammars are Context-Free.

Theorem (Pentus 2003, Pentus 2006)

Lambek Calculus is NP-complete.

- Some natural language phenomena are outside the scope of Lambek grammars.
- There is little hope of obtaining efficient parsing algorithms for the Lambek calculus.

Introduction

The Syntactic Calculus — Can We Do More With Less?

- Can we do more with less? That is, can we
 - 1 *extend* the range of linguistic phenomena we can handle,
 - 2 while *reducing* the computational complexity of the calculus?
- There are no Syntactic Calculus solutions for phenomena like.
 - a medial *wh* extraction,
 - b non-peripheral quantifier scope,
 - c copying constructions, as found in Bambara but also in ‘X or no X’ construction in English,
 - d crossing dependencies, as found in Dutch, at least if we want to have the correct object-verb dependencies,
 - e scrambling, as found in Swiss-German,
- Can we find an extension to Syntactic Calculus which can efficiently parse these phenomena?

Introduction

The Non-Associative Lambek Calculus

- **NL** was introduced by Lambek (1961) as a restriction of the Lambek calculus.
- Since linguists tend to think of trees as the basic linguistic structures, a logic of trees seems (at least a priori) a good choice.
- de Groote (1999) showed we can parse **NL** grammars in polynomial time.
- Kandulski (1988) showed that **NL** — like the Syntactic Calculus — generates only context-free languages.

Introduction

The Multimodal Lambek Calculus

- Various extensions to **NL** have been proposed.
- Moortgat & Oehrle (1994) have given an analysis of Dutch verb clusters using the *multimodal* Lambek calculus $\mathbf{NL}\diamond_{\mathcal{R}}$, thereby showing this calculus generates more than just context-free languages.
- However, the large freedom in proposing structural rules in \mathcal{R} means the logic is Turing complete in general, even though a simply restriction on the structural rules gives a PSPACE formalism generating exactly the context-sensitive languages (Moot 2002).
- Little else is known about the exact classes of languages generated by multimodal Lambek calculi.

Introduction

The Multimodal Lambek Calculus

- We would like to find a restriction of the multimodal calculus which extends **NL**, but does so while keeping a polynomial parsing algorithm.
- The *mildly* context-sensitive languages and the different corresponding formal systems (TAGs, CCGs, etc.) seem a good compromise between parsing complexity and descriptive accuracy.
- So the question I want to answer today is: can we find a fragment of $\mathbf{NL} \diamond_{\mathcal{R}}$ which generates mildly context-sensitive languages, yet allows for polynomial parsing?

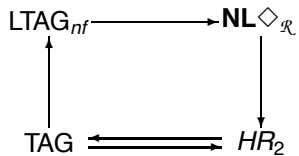
Introduction

Overview of Lambek Calculi: Logic, Language, Complexity

Logic	L	NL	???	NL $\diamond_{\mathcal{R}}$
Complexity	NP	P	P	PSPACE
Languages	CFL	CFL	MCSL	CSL

Overview

Relations



Overview

Relations



Hyperedge Replacement Grammars

Introduction

Hyperedge replacement grammars were introduced by Bauderon & Courcelle (1987) and Habel & Kreowski (1987) as a type of context-free graph grammars. The string, tree and graph languages generated by HRG have been well-studied.

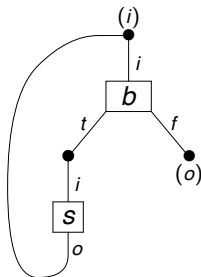
Because of

- the general data structure used (hypergraphs),
- the simplicity of the hyperedge replacement operation

hyperedge replacement grammars will serve as a bridge between tree adjoining grammars and proof nets for the multimodal Lambek calculus.

Hyperedge Replacement Grammars

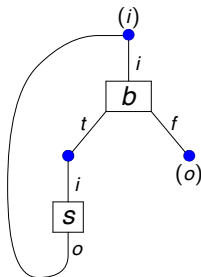
Hypergraphs With External Nodes



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

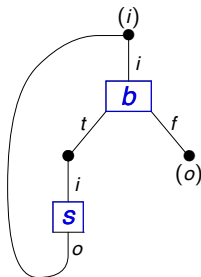
– vertices are drawn as fat dots.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

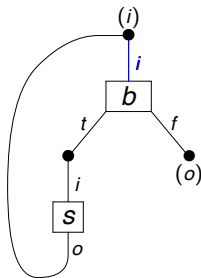
- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

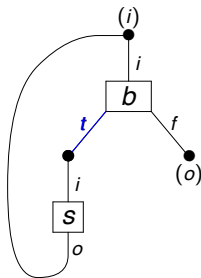
- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

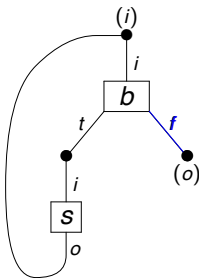
- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

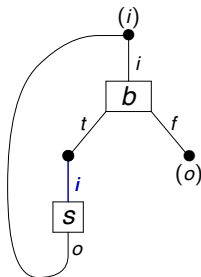
- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

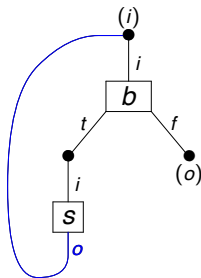
- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.



Hyperedge Replacement Grammars

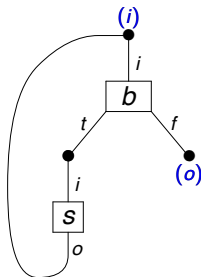
Hypergraphs With External Nodes

- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

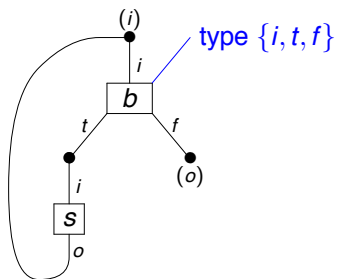


- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.
- some vertices of the hypergraph are labeled as external.

Hyperedge Replacement Grammars

Hypergraphs With External Nodes — Types

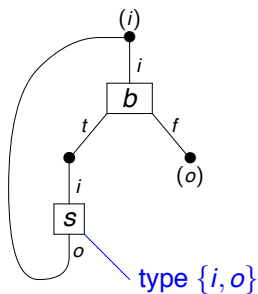
- the *type* of a hyperedge is the set of its tentacle labels.



Hyperedge Replacement Grammars

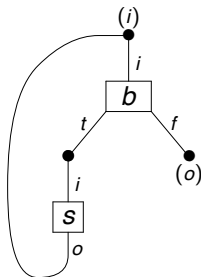
Hypergraphs With External Nodes — Types

- the *type* of a hyperedge is the set of its tentacle labels.



Hyperedge Replacement Grammars

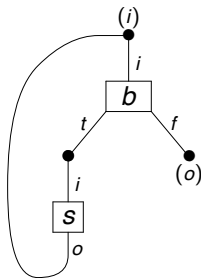
Hypergraphs With External Nodes — Types



- the *type* of a hyperedge is the set of its tentacle labels.
- the *type* of a hypergraph is the set of its external node labels: $\{i, o\}$ in this example.

Hyperedge Replacement Grammars

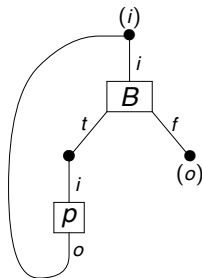
Hypergraphs With External Nodes — Types



- the *type* of a hyperedge is the set of its tentacle labels.
- the *type* of a hypergraph is the set of its external node labels: $\{i, o\}$ in this example.
- *hyperedge replacement* replaces a hyperedge by a hypergraph of the same type.

Hyperedge Replacement Grammars

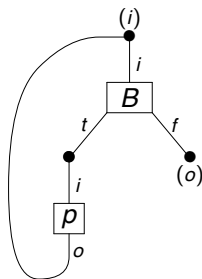
Hyperedge Replacement



H

Hyperedge Replacement Grammars

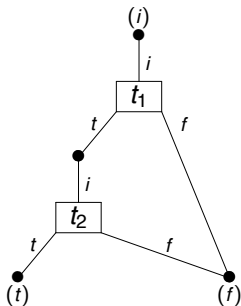
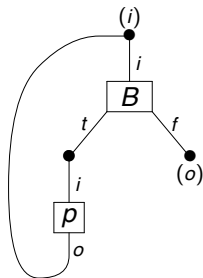
Hyperedge Replacement



edge e (labeled B)
of type $\{i, t, f\}$
 H

Hyperedge Replacement Grammars

Hyperedge Replacement



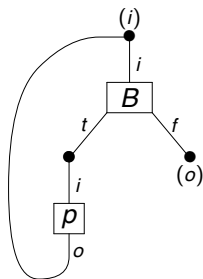
edge e (labeled B)
of type $\{i, t, f\}$

H

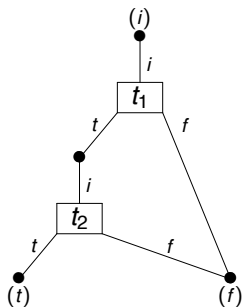
K

Hyperedge Replacement Grammars

Hyperedge Replacement



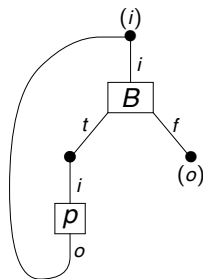
edge e (labeled B)
of type $\{i, t, f\}$
 H



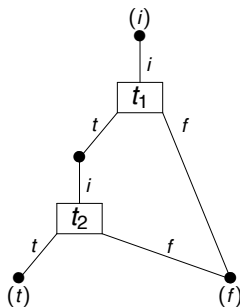
hypergraph K
of type $\{i, t, f\}$
 K

Hyperedge Replacement Grammars

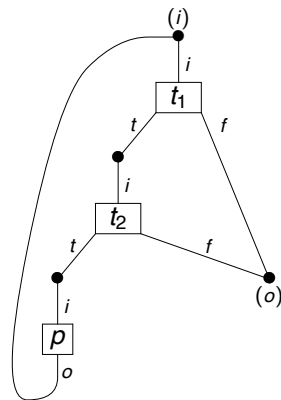
Hyperedge Replacement



edge e (labeled B)
 of type $\{i, t, f\}$
 H



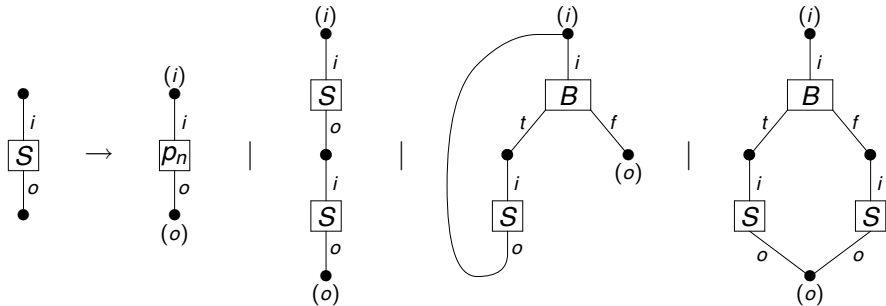
hypergraph K
 of type $\{i, t, f\}$
 K



$H[e := K]$

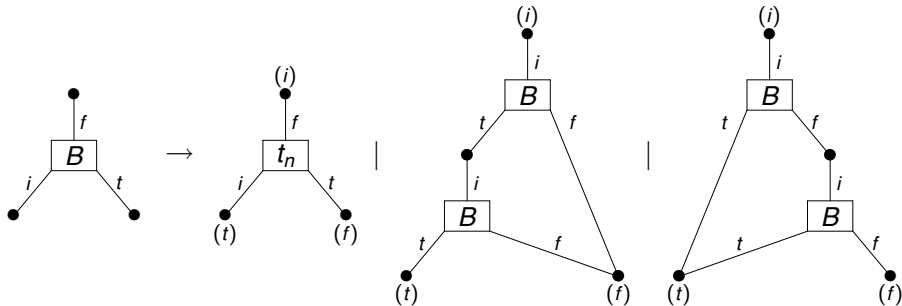
Hyperedge Replacement Grammars

Example Grammar



Hyperedge Replacement Grammars

Example Grammar



Hyperedge Replacement Grammars

Rank

Definition

The *rank* of a hyperedge replacement grammar is the maximum number of tentacles of a nonterminal of the grammar.

- In the previous example, the rank of the grammar is three: the B nonterminal is of type $\{i, t, f\}$, the S nonterminal of type $\{i, o\}$.
- In general, the classes of string, tree and graph languages generated by hyperedge replacement grammars increase as the rank of the grammar increases (Habel & Kreowski 1987, Weir 1992)

Tree Adjoining Grammars

Introduction

- mathematically elegant formalism, which gives linguistically relevant structural descriptions,
- very simple basic operations,
- many results with respect to the classes of languages generated, the computational complexity and parsing.

Tree Adjoining Grammars as HR Grammars

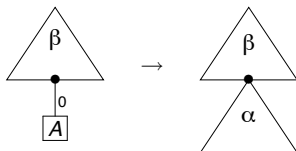
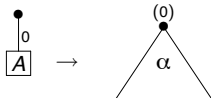
Introduction

In the context of HR grammars, Tree Adjoining Grammars can be seen as a special case of hyperedge replacement grammars where:

- every non-terminal hyperedge label has at most two tentacles, that is, the rank of the grammar is (at most) two.
- every right-hand side of a HR rule is either:
 - a tree with the root as its sole external node.
 - a tree with a root and a leaf as its external nodes.

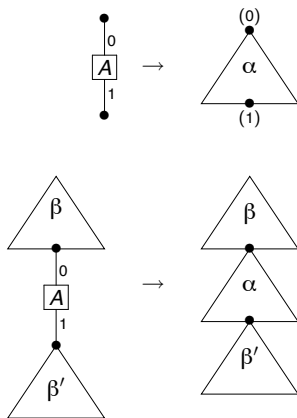
Tree Adjoining Grammars as HR Grammars

Substitution



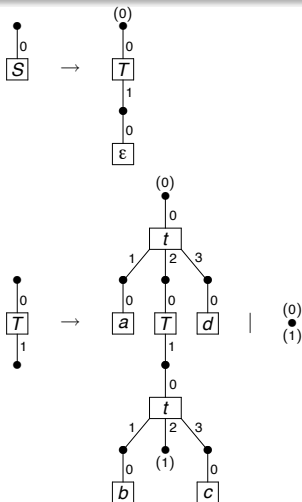
Tree Adjoining Grammars as HR Grammars

Adjunction



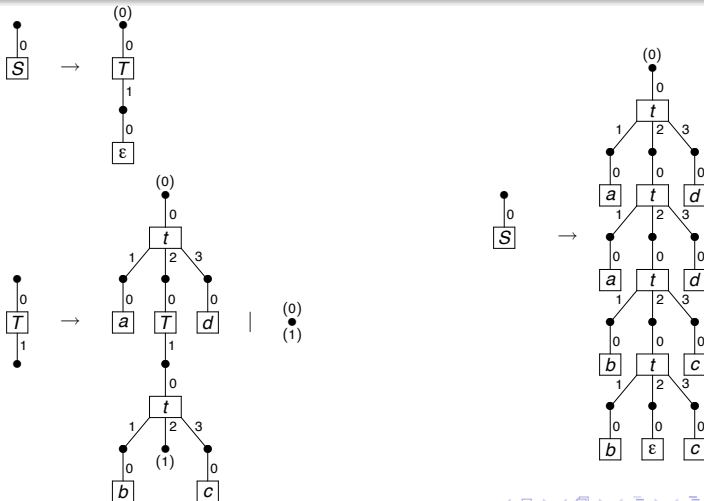
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



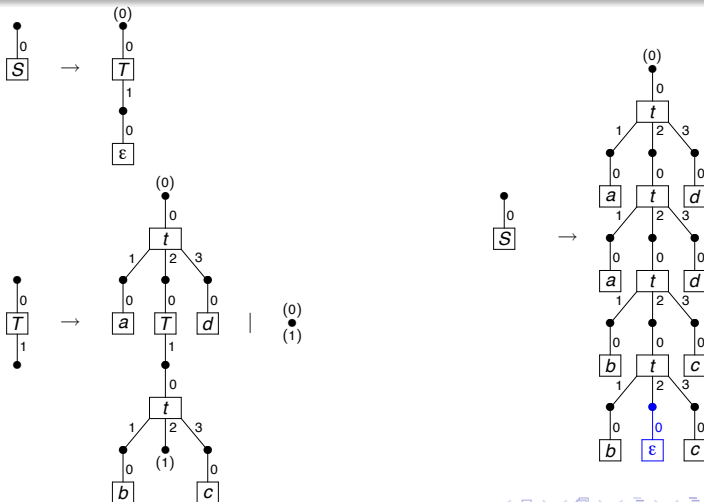
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



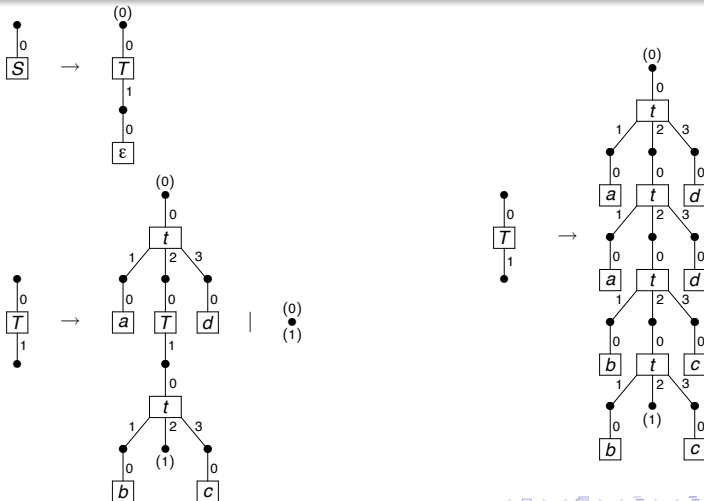
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



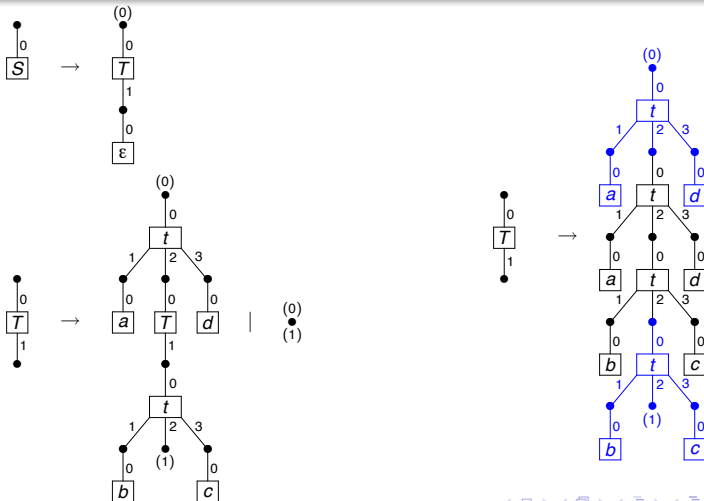
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



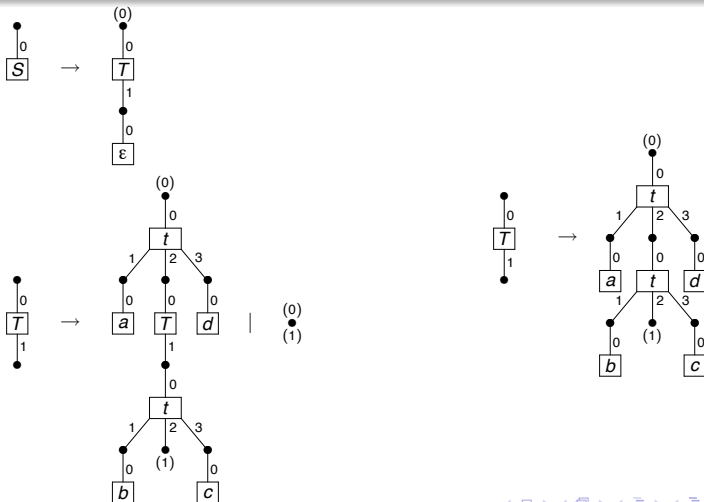
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



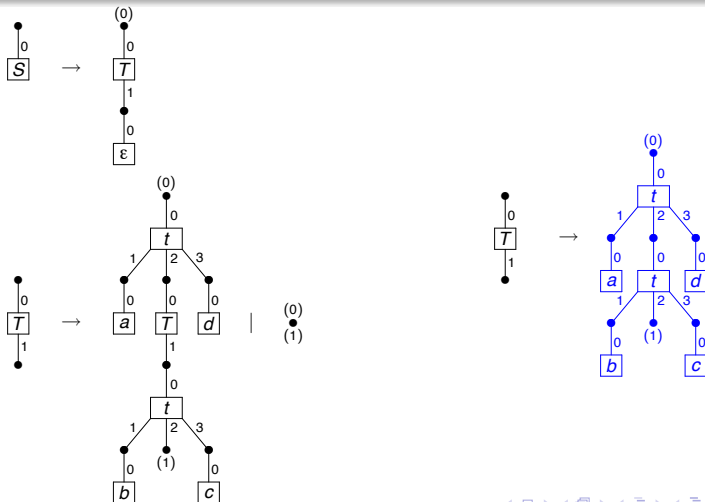
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



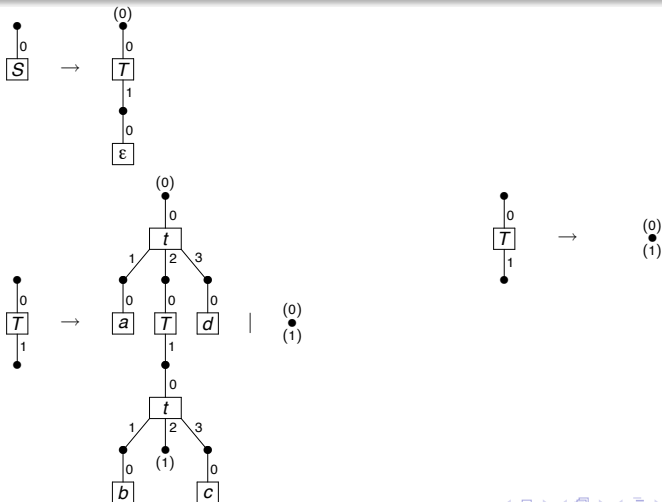
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



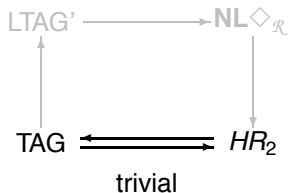
Tree Adjoining Grammars as HR Grammars

Example: $a^n b^n c^n d^n$



Intermezzo

Relations so far



Tree Adjoining Grammars

A Normal Form for LTAGs

Definition

An LTAG G is in *normal form* if it satisfies the following conditions.

- all internal nodes of elementary trees have exactly two daughters,
- all foot nodes in the grammar have the null adjunction constraint,
- every adjunction node either specifies the null adjunction or the obligatory adjunction constraint without any selectional restrictions,
- every adjunction node is on the path from the (unique) *lexical anchor* to the root of the tree.

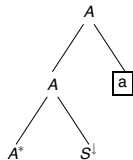
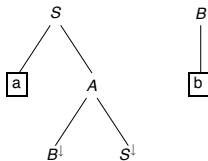
Tree Adjoining Grammars

A Normal Form for LTAGs

- The definition of normal form proposed here is very close to the definition of *spinal form* used by Vijay-Shanker & Weir (1994) to establish inclusion of tree adjoining languages into the languages generated by combinatory categorial grammar.
- It will serve a similar role here.
- The crucial point about this normal form is that all adjunctions take place at subformulas of *negative* polarity.
- Transforming an LTAG into a weakly equivalent LTAG in normal form is very similar to transforming an LTAG into spinal form.

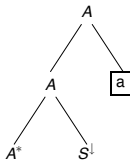
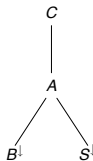
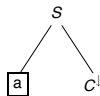
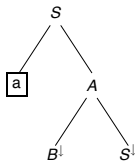
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



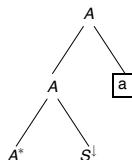
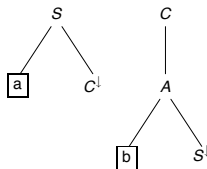
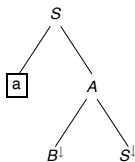
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



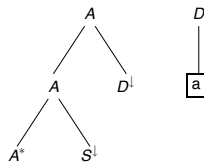
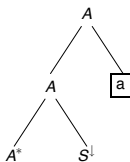
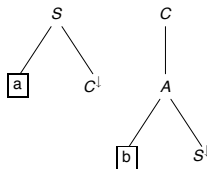
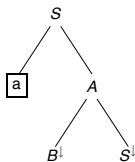
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



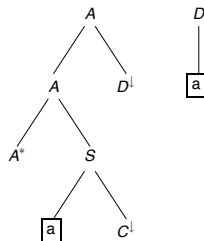
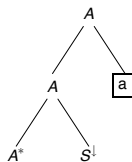
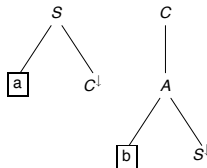
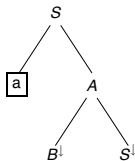
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



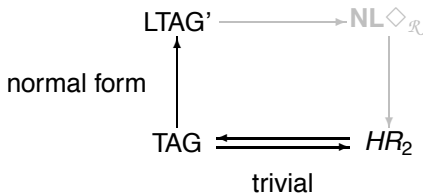
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



Intermezzo

Relations so far



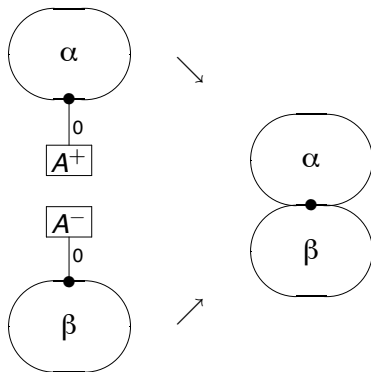
Proof Nets

Introduction

- Proof nets are a non-redundant way of representing proofs in linear logic and different Lambek calculi using (hyper-)graphs.
- The proof nets used here are (a hypergraph version of) the multimodal proof nets of Moot & Puite (2002).
- Proof nets for AB are trees which combine using the axiom rule, which corresponds to the substitution operation in tree adjoining grammars.
- Contractions and structural rules give a mechanism to rewrite proof nets into trees.
- The contractions and structural rules together allow us to simulate adjunction.

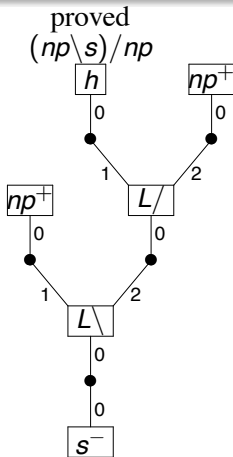
Proof Nets for AB

The Axiom Rule



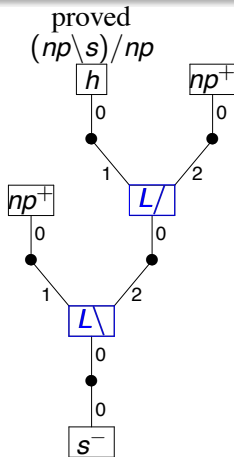
Proof Nets for AB

The Anatomy of a Lexical Entry



Proof Nets for AB

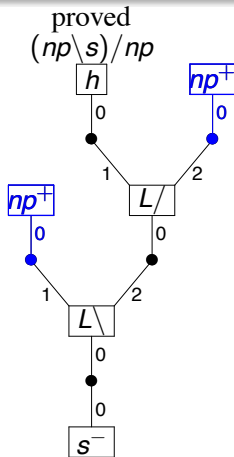
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names

Proof Nets for AB

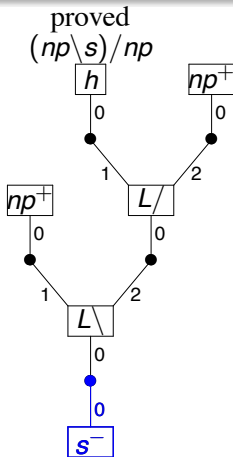
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates “I require an np resource”

Proof Nets for AB

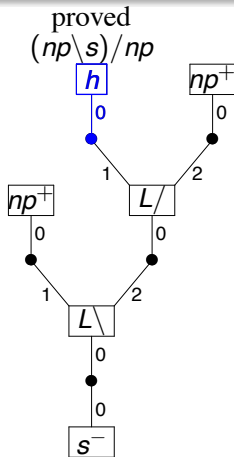
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates “I require an np resource”
- inversely, s^- indicates “I provide an s resource”

Proof Nets for AB

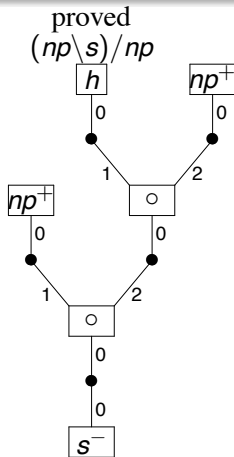
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates “I require an np resource”
- inversely, s^- indicates “I provide an s resource”
- h indicates the position of the lexical anchor (hypothesis)

Proof Nets for AB

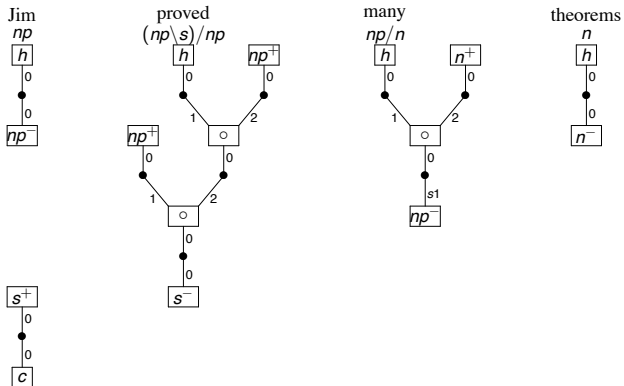
The Anatomy of a Lexical Entry



For maximum economy: the lexical anchor allows us to uniquely determine the rule name, so we can replace all rule names by a generic label

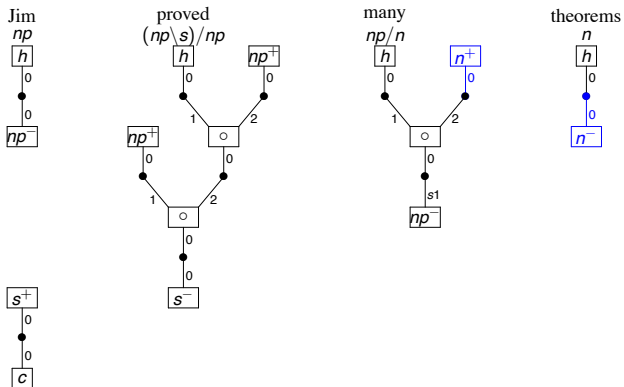
Proof Nets for AB

Example



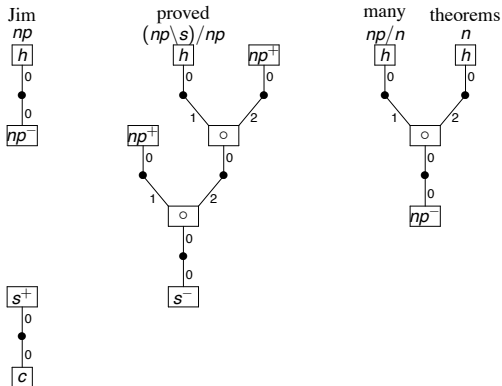
Proof Nets for AB

Example



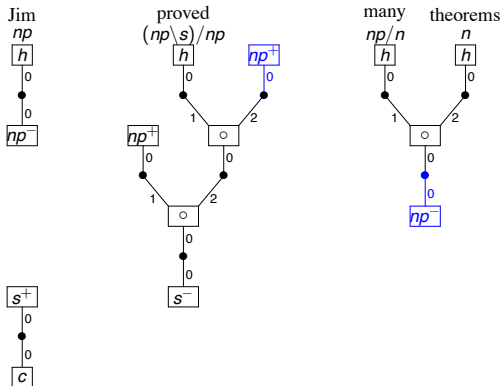
Proof Nets for AB

Example



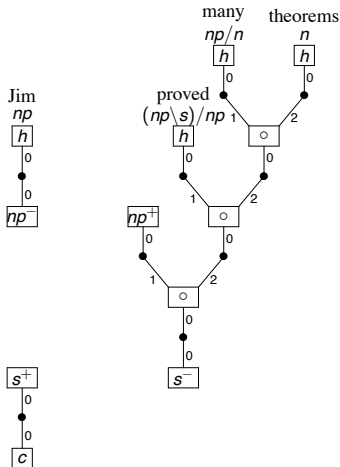
Proof Nets for AB

Example



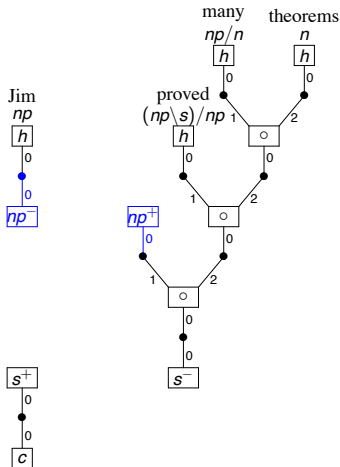
Proof Nets for AB

Example



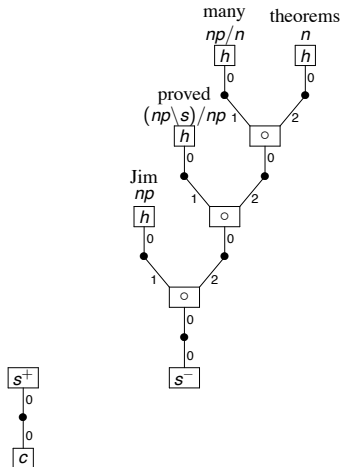
Proof Nets for AB

Example



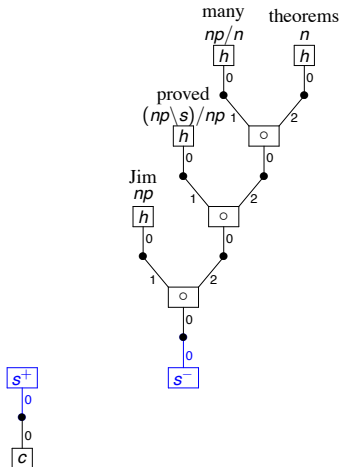
Proof Nets for AB

Example



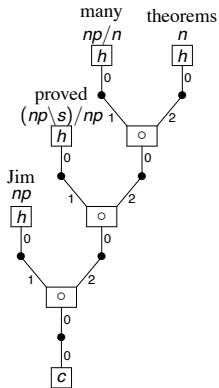
Proof Nets for AB

Example



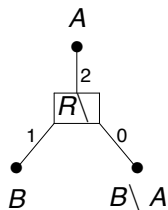
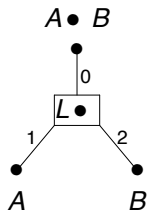
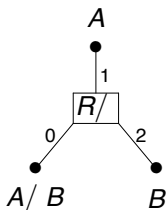
Proof Nets for AB

Example



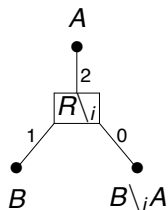
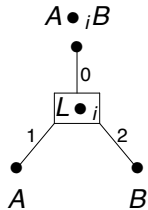
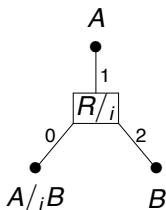
Multimodal Proof Nets

New Links



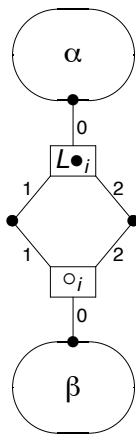
Multimodal Proof Nets

New Links — And Modes



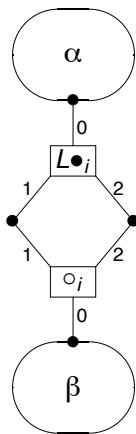
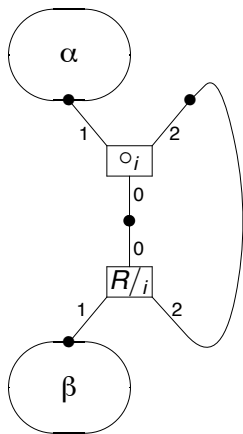
Multimodal Proof Nets

Contractions



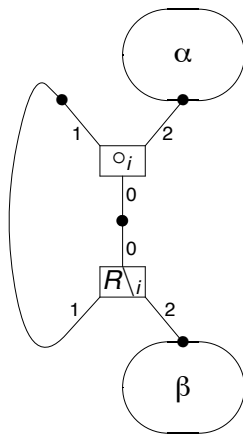
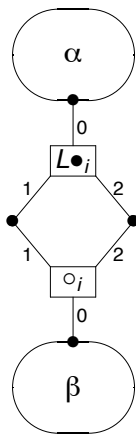
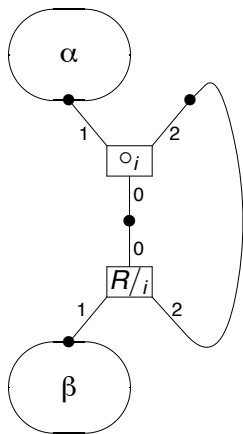
Multimodal Proof Nets

Contractions



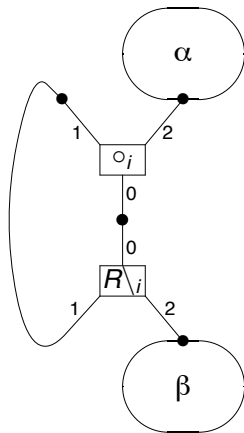
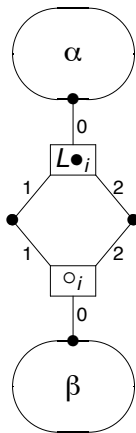
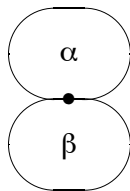
Multimodal Proof Nets

Contractions



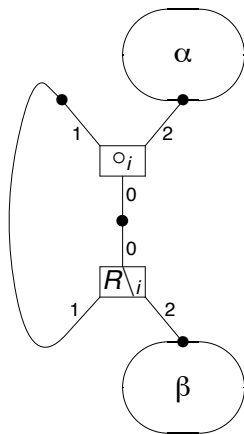
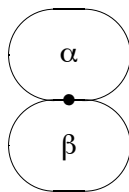
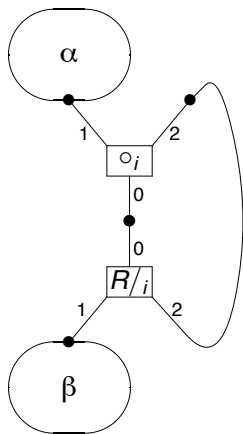
Multimodal Proof Nets

Contractions



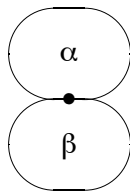
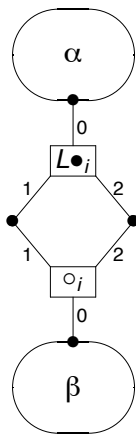
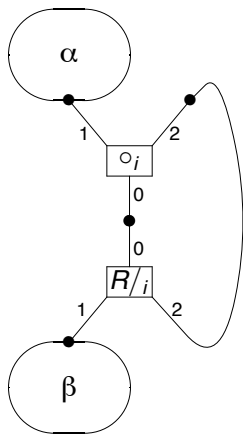
Multimodal Proof Nets

Contractions



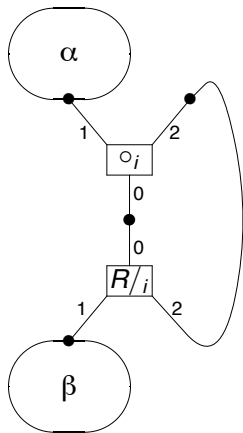
Multimodal Proof Nets

Contractions



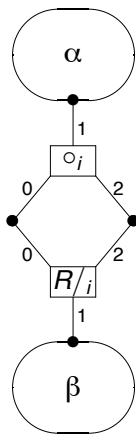
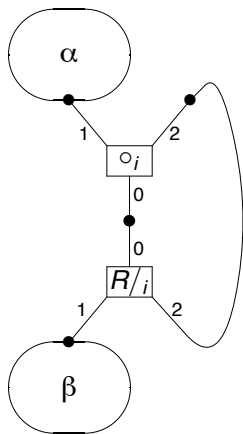
Multimodal Proof Nets

Contractions — Rotation



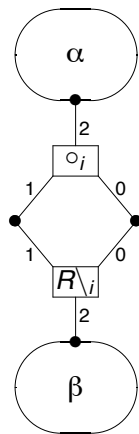
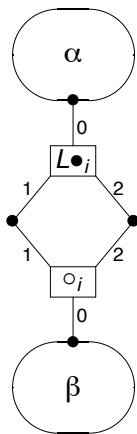
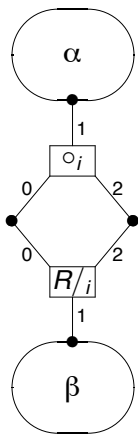
Multimodal Proof Nets

Contractions — Rotation



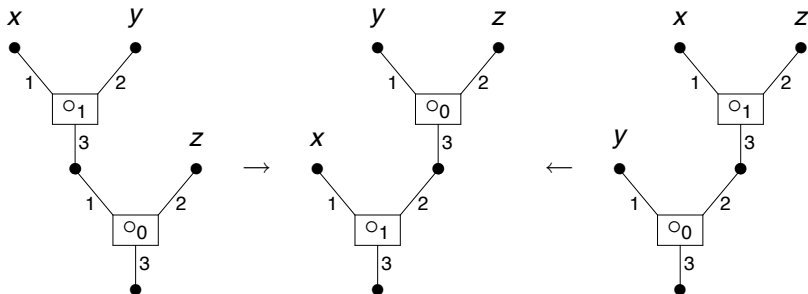
Multimodal Proof Nets

Contractions — Rotated



Structural Rules

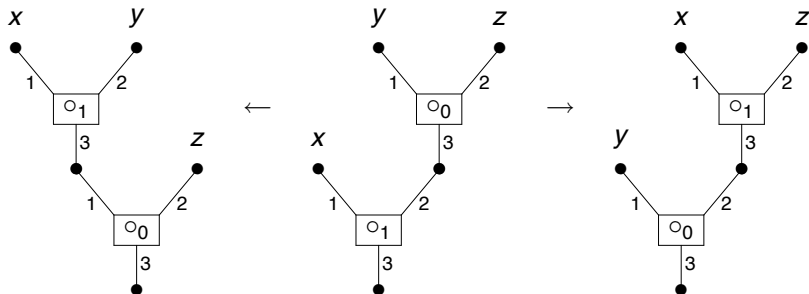
Mixed Associativity and Commutativity— Extraction



Structural Rules

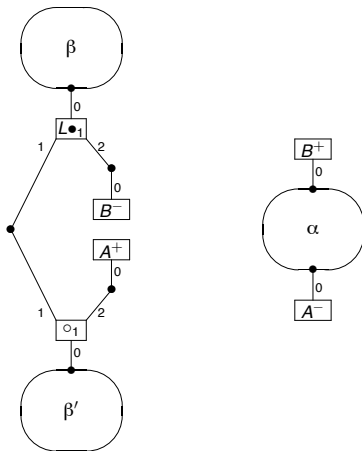
Mixed Associativity and Commutativity

— Infixation



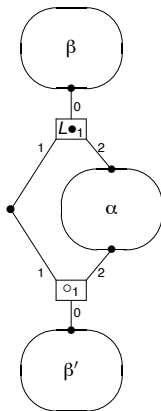
Simulating Adjunction

Initial Configuration of Adjunction Point



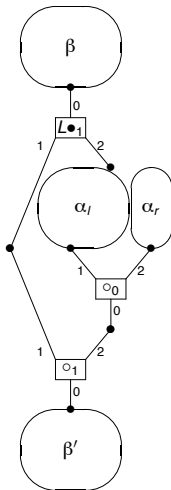
Simulating Adjunction

Axioms



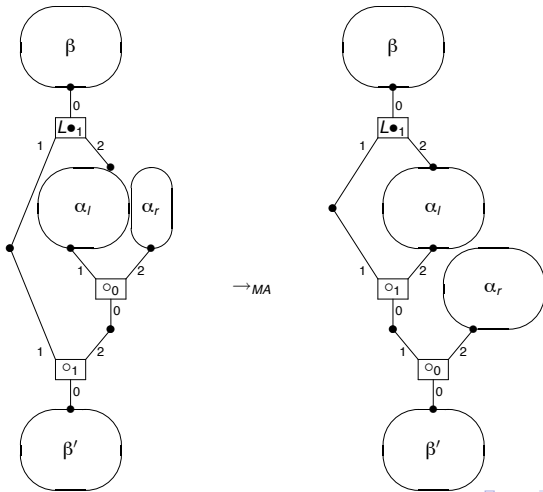
Simulating Adjunction

Structural Rules — Left



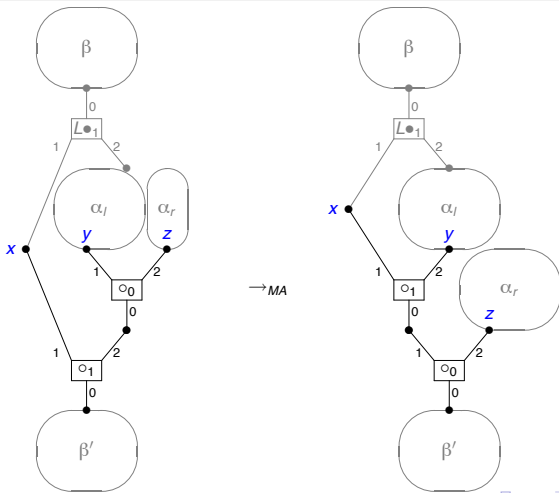
Simulating Adjunction

Structural Rules — Left



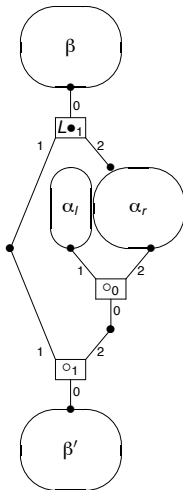
Simulating Adjunction

Structural Rules — Left



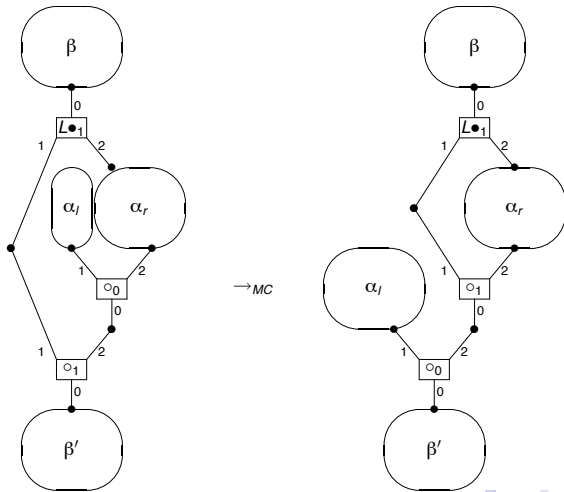
Simulating Adjunction

Structural Rules — Right



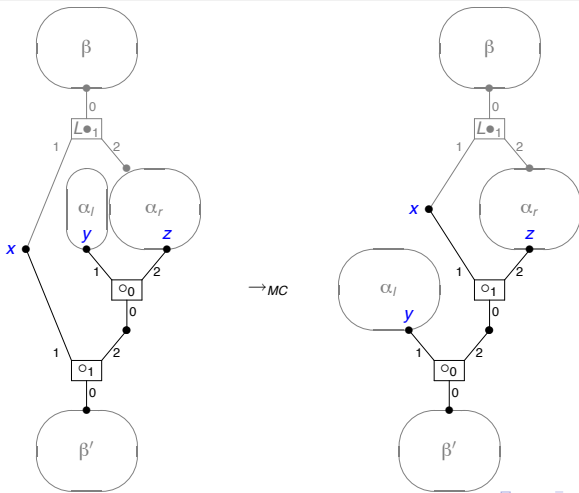
Simulating Adjunction

Structural Rules — Right



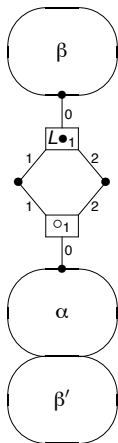
Simulating Adjunction

Structural Rules — Right



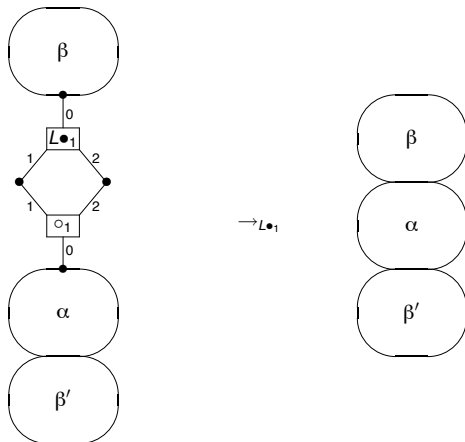
Simulating Adjunction

Structural Rules — Contraction and Final Result



Simulating Adjunction

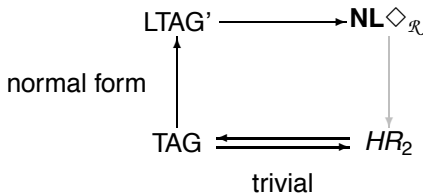
Structural Rules — Contraction and Final Result



Intermezzo

Relations so far

adjunction as contraction
and structural rules



HRG and Proof Nets

Nonterminal Symbols

- S of type \emptyset : “start”,

HRG and Proof Nets

Nonterminal Symbols

- S of type \emptyset : “start”,
- T of type $\{0, 1\}$: “proof net contracting to a *tree*”,

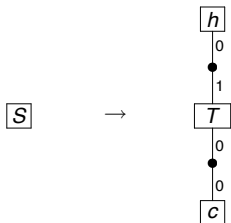
HRG and Proof Nets

Nonterminal Symbols

- S of type \emptyset : “start”,
- T of type $\{0, 1\}$: “proof net contracting to a *tree*”,
- V of type $\{0, 1\}$: “proof net contracting to a *vertex*”.

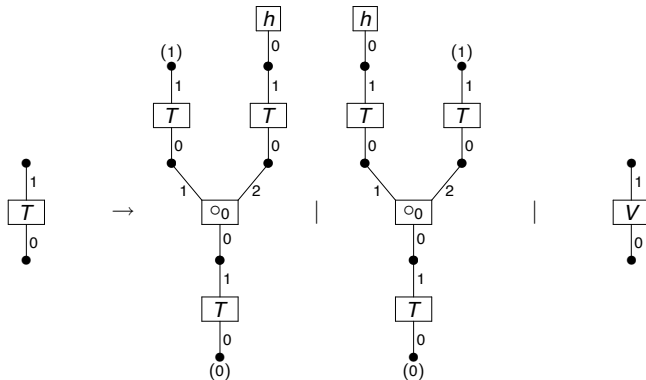
HRG and Proof Nets

Initial Axiom



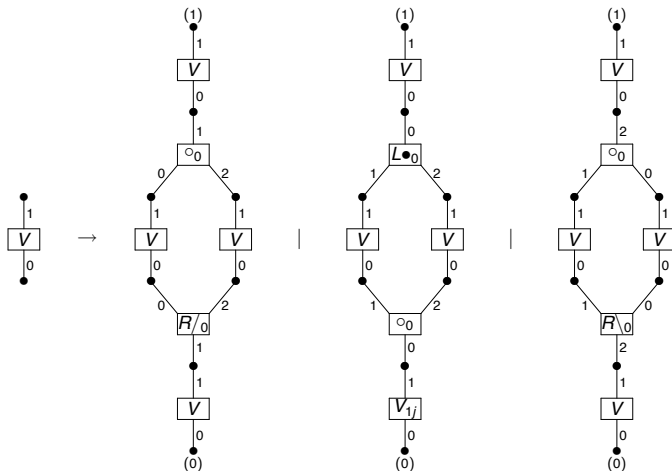
HRG and Proof Nets

Tensor Trees



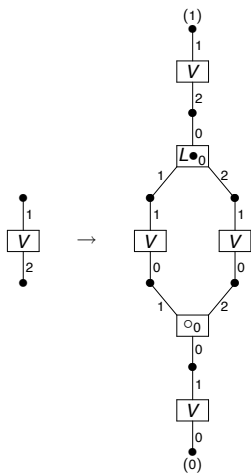
HRG and Proof Nets

Contractions



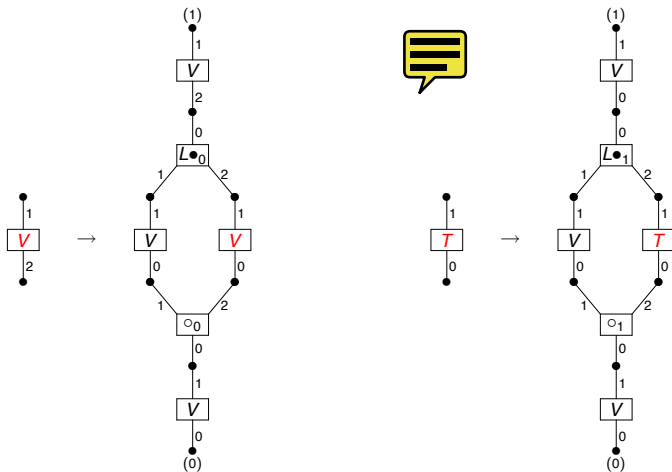
HRG and Proof Nets

Structural Rules: Mixed Associativity, Mixed Commutativity



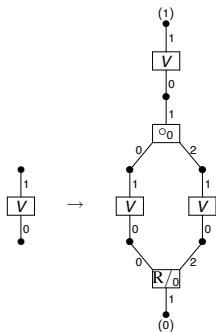
HRG and Proof Nets

Structural Rules: Mixed Associativity, Mixed Commutativity



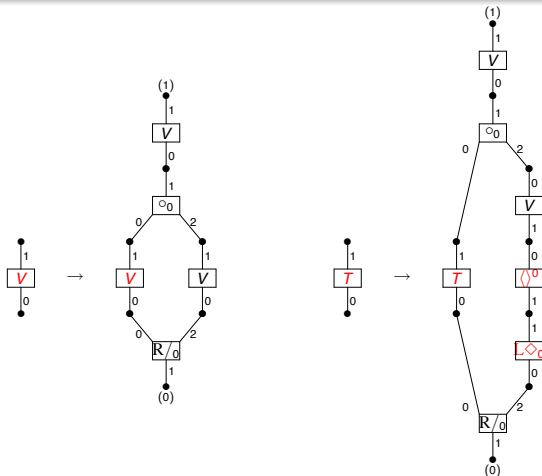
HRG and Proof Nets

Structural Rules: Mixed Associativity, Mixed Commutativity With Unary Control



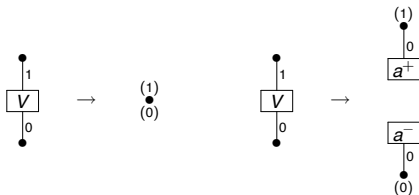
HRG and Proof Nets

Structural Rules: Mixed Associativity, Mixed Commutativity With Unary Control



HRG and Proof Nets

Cut, Flow, Axiom



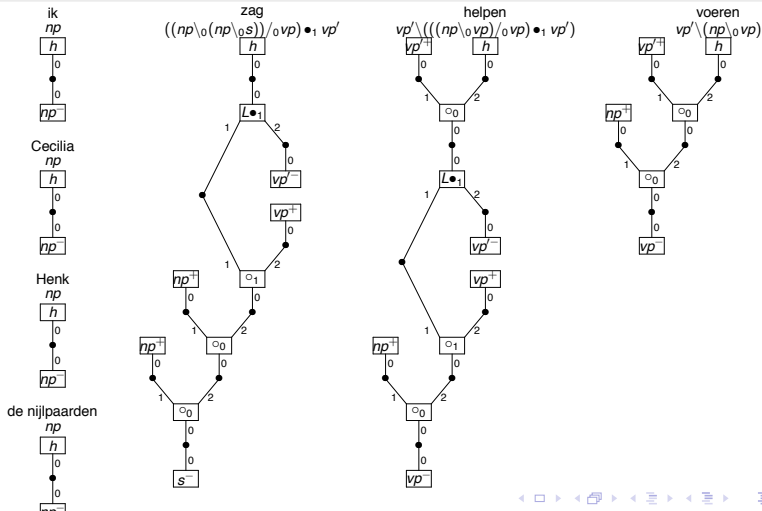
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars

- Hyperedge replacement grammars — even those of rank 2 — can generate NP complete graph languages.
- Lautemann (1990) gives a dynamic programming algorithm for parsing graphs with hyperedge replacement grammars.
- In addition, he gives conditions for which the complexity of hyperedge replacement languages is $\text{LOGCFL} \subseteq \text{P}$.
- His first condition interests us here: it gives polynomial parsing by limiting the number of disconnected subgraphs with respect to the rank of the grammar.
- In addition, though this is not used by Lautemann, we can use a yield function to keep track of the (pairs of) strings we are generating to improve the performance of the parser.

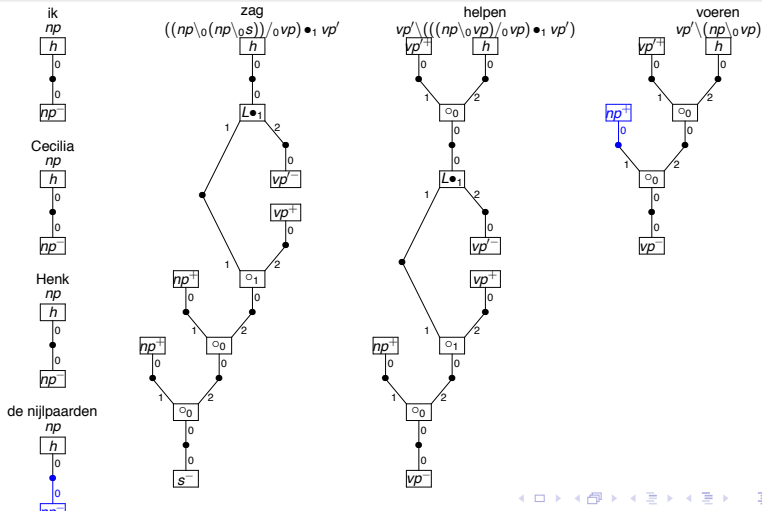
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



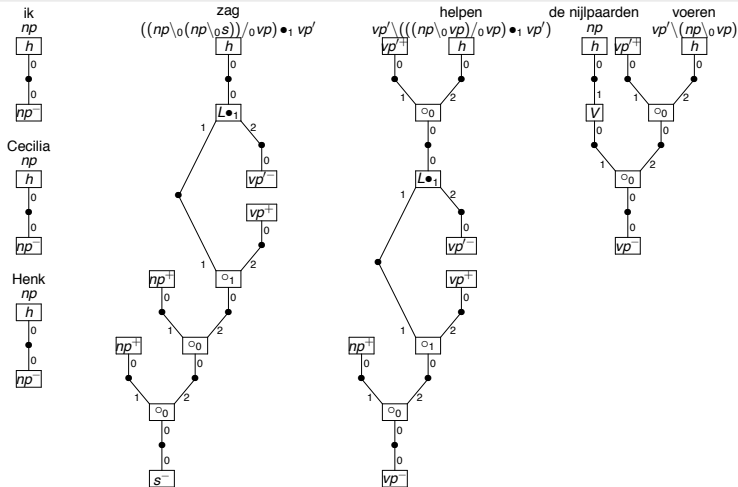
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



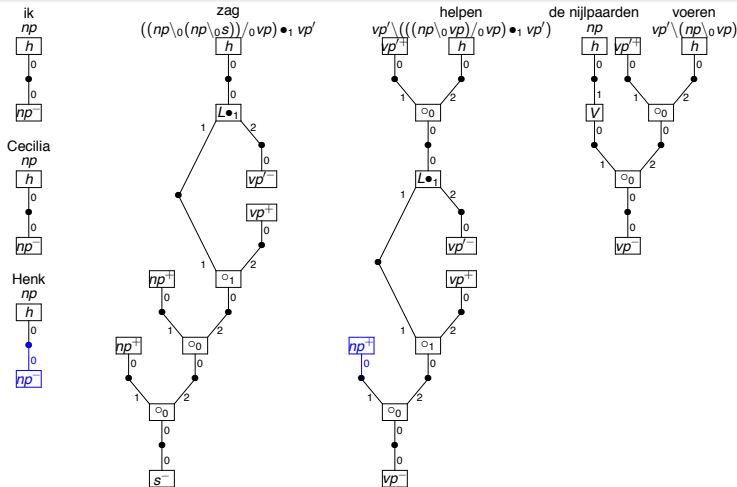
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



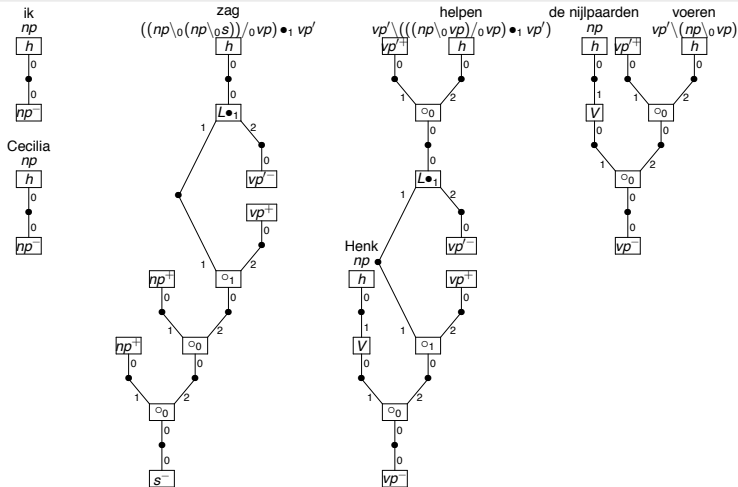
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



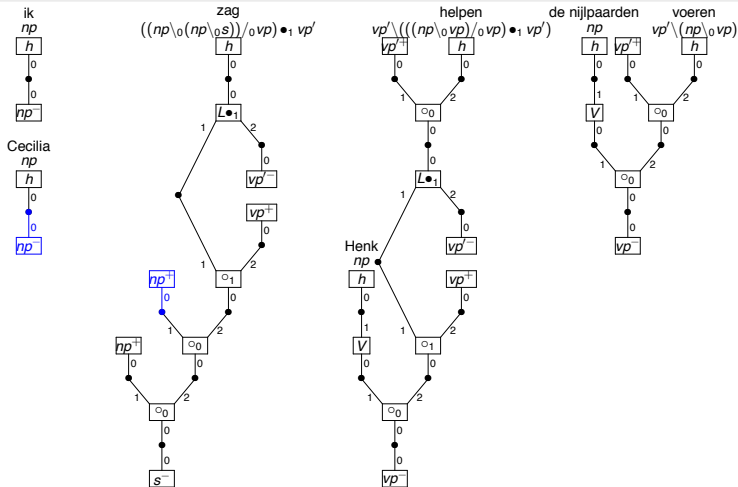
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



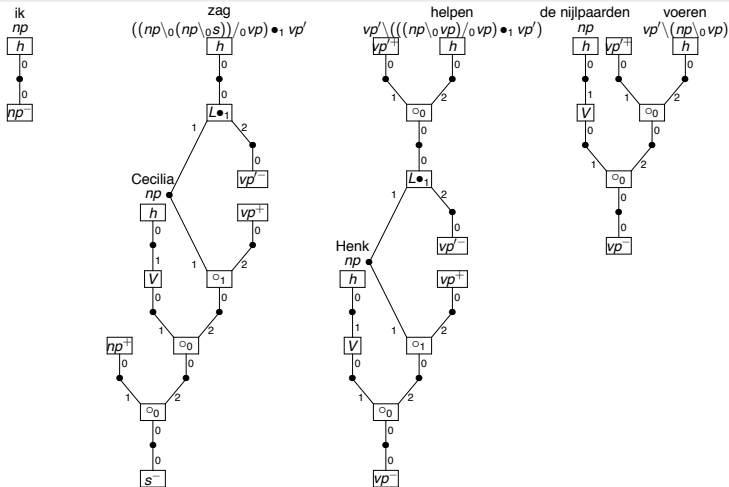
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



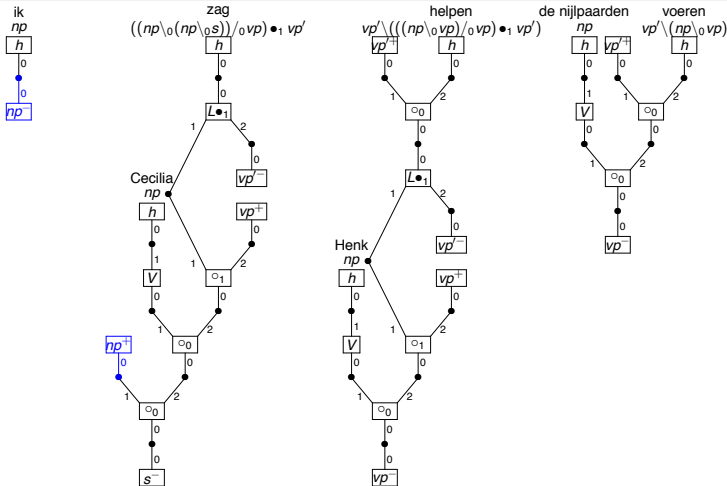
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



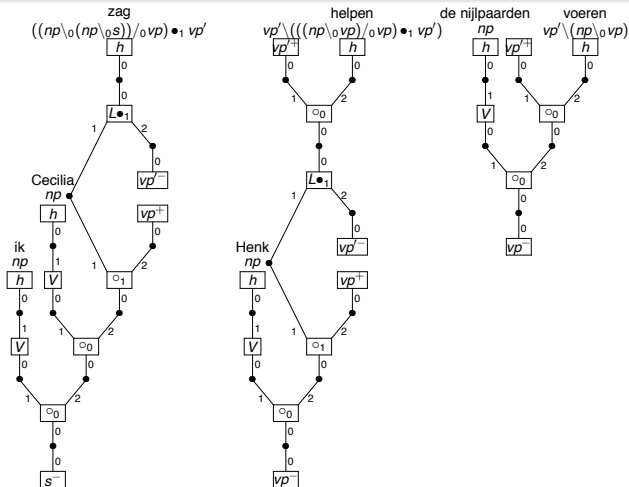
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



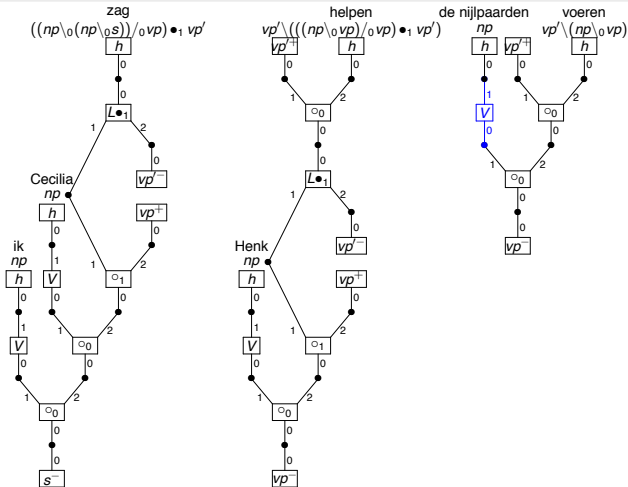
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



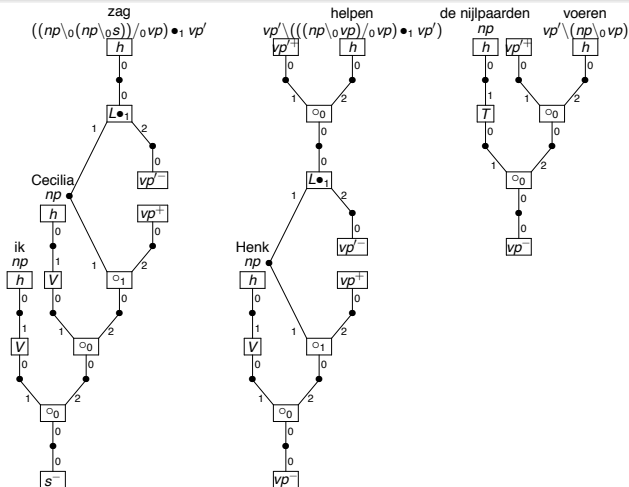
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



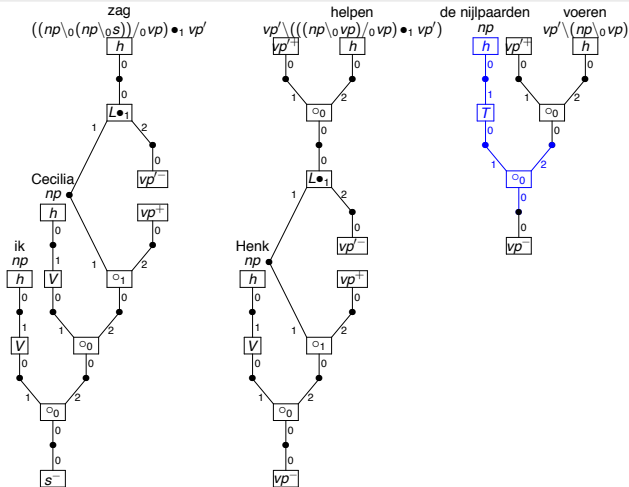
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



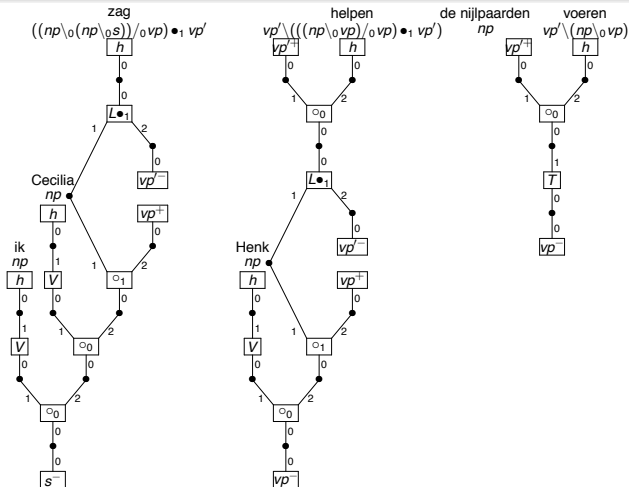
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



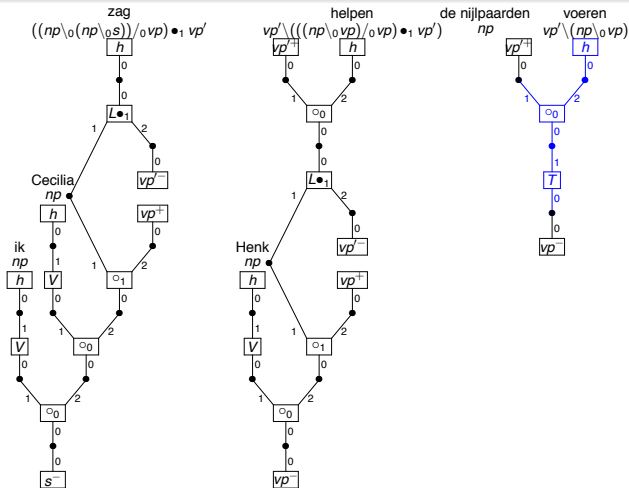
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



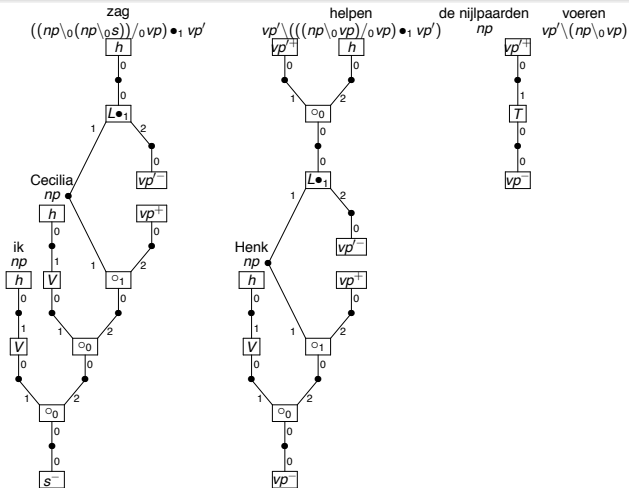
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



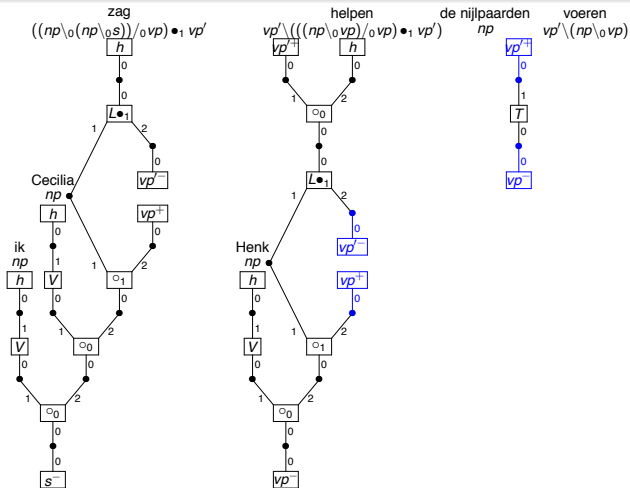
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



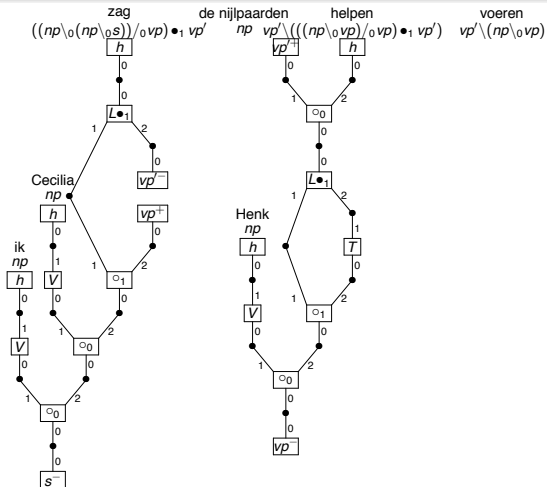
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



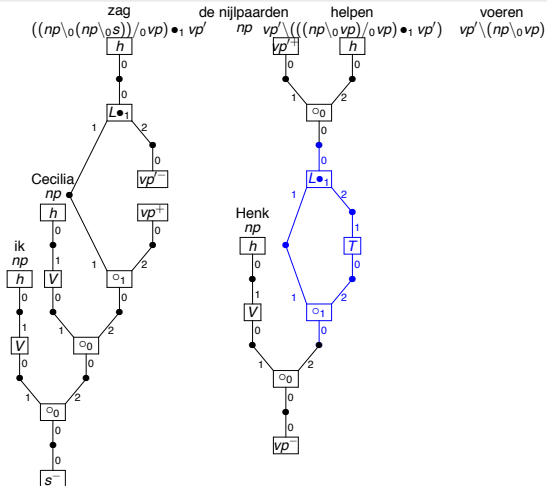
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



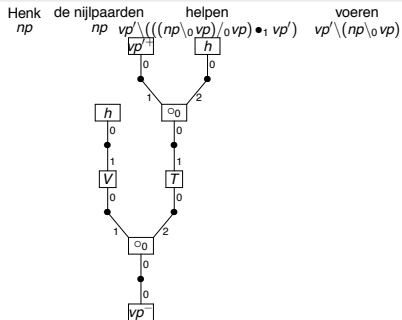
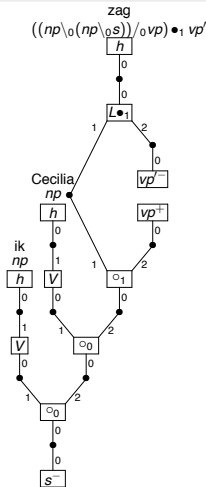
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



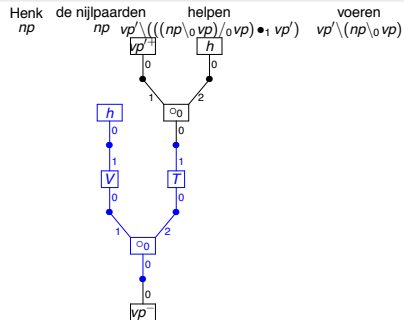
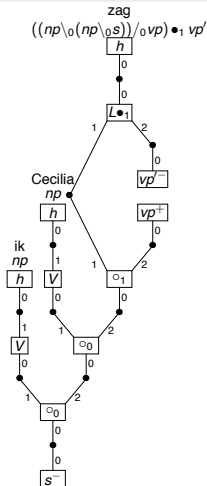
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



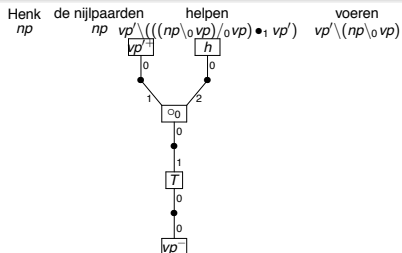
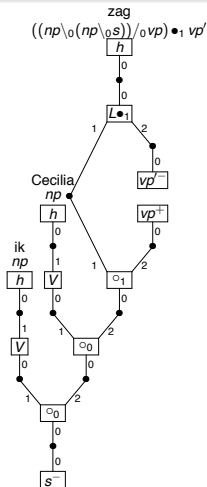
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



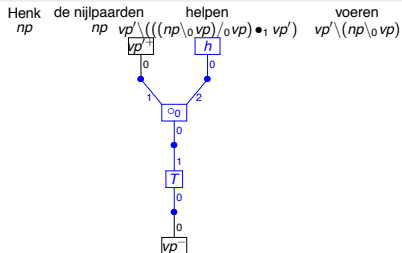
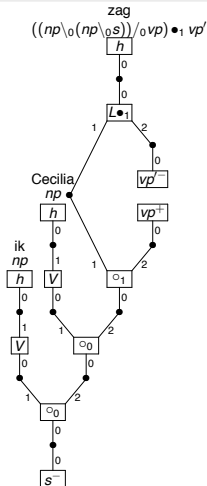
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



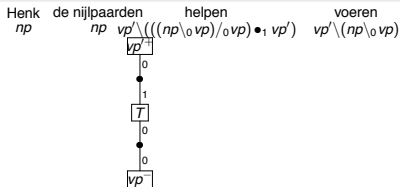
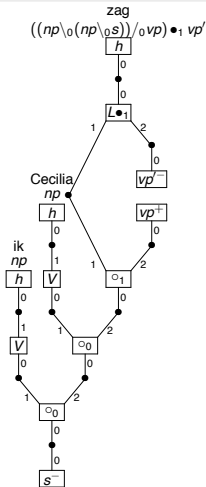
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



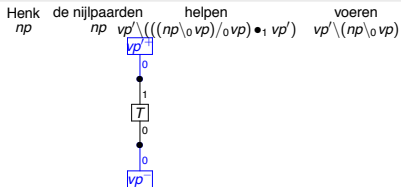
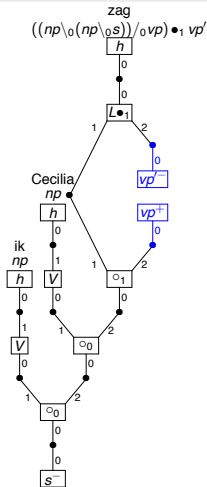
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



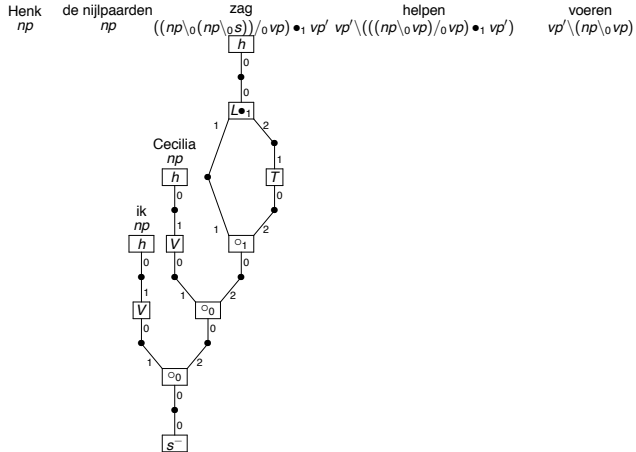
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

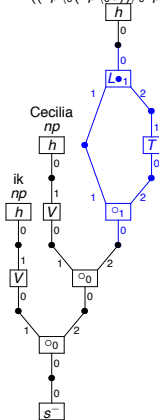
Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

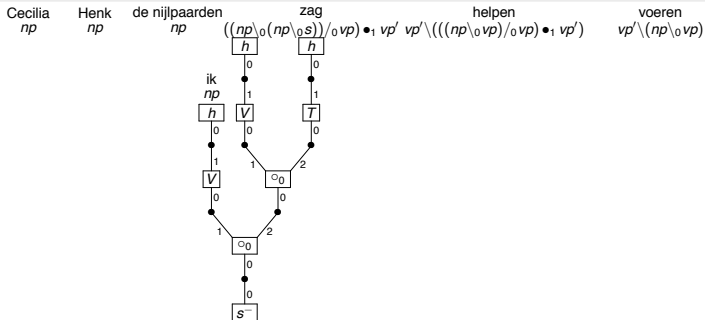
Parsing With Hyperedge Replacement Grammars

Henk np de np zag $((np \backslash_0 (np \backslash_0 s)) /_0 vp) \bullet_1 vp' \backslash \backslash ((np \backslash_0 vp) /_0 vp) \bullet_1 vp'$ helpen $vp' \backslash (np \backslash_0 vp)$ voeren $vp' \backslash (np \backslash_0 vp)$



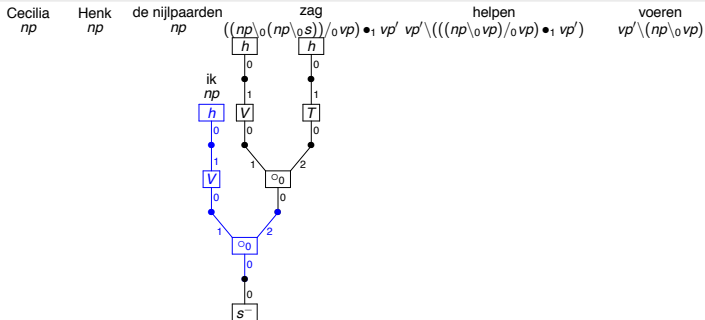
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



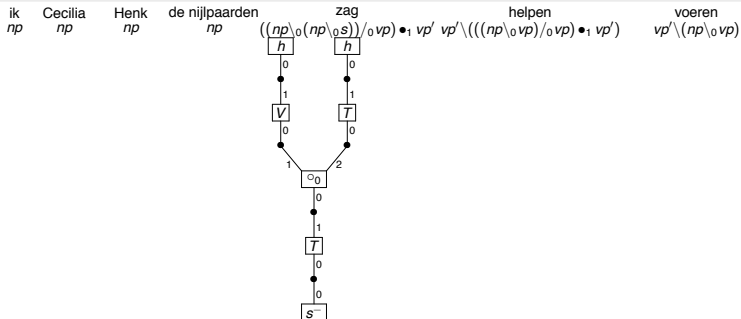
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



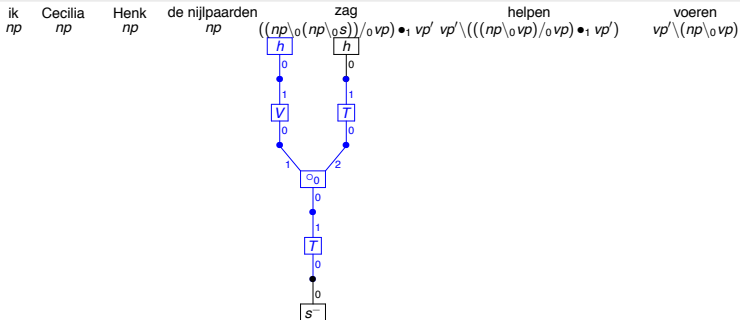
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



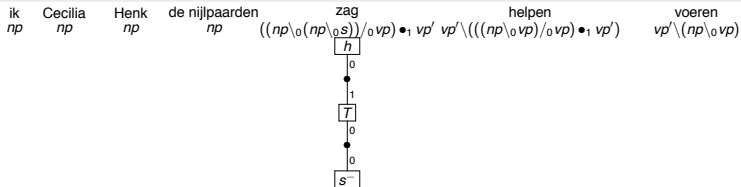
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



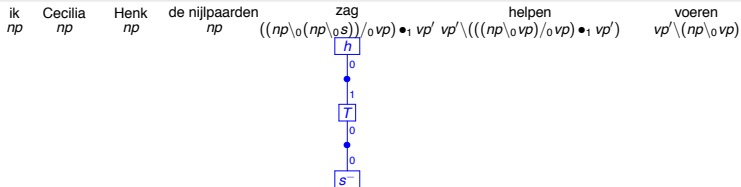
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

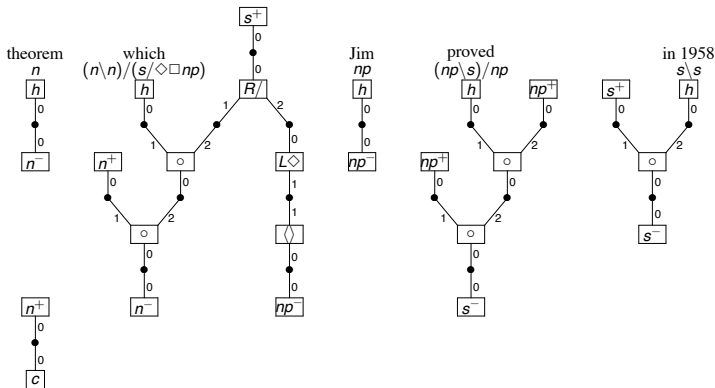
Parsing With Hyperedge Replacement Grammars

ik Cecilia Henk de nijlpaarden zag helpen voeren
np np np np $((np \backslash_0 (np \backslash_0 s)) /_0 vp) \bullet_1 vp' vp' \backslash (((np \backslash_0 vp) /_0 vp) \bullet_1 vp')$ $vp' \backslash (np \backslash_0 vp)$

S

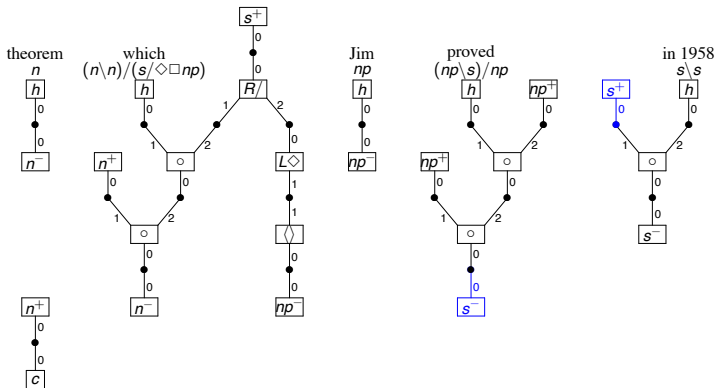
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



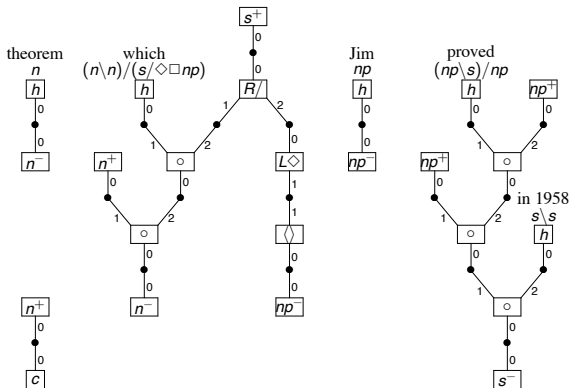
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



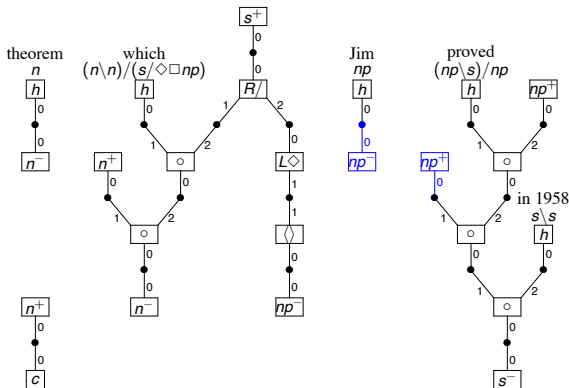
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



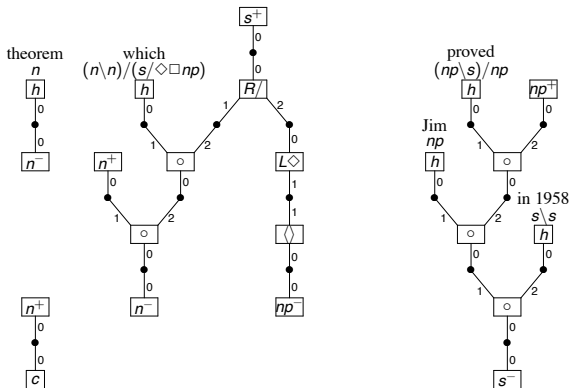
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



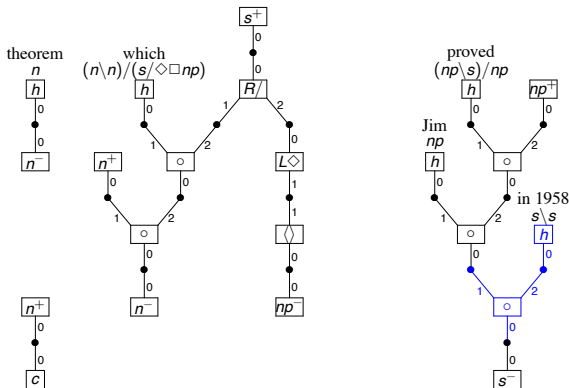
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



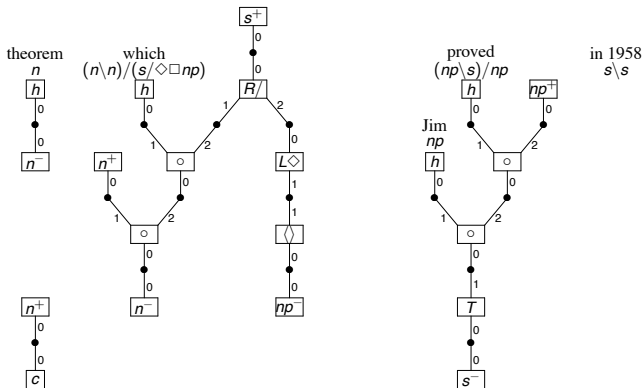
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



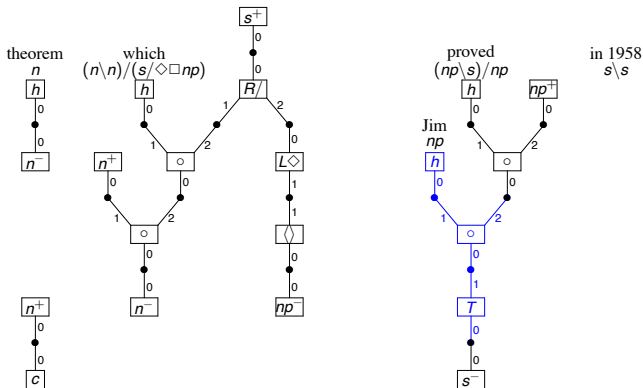
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



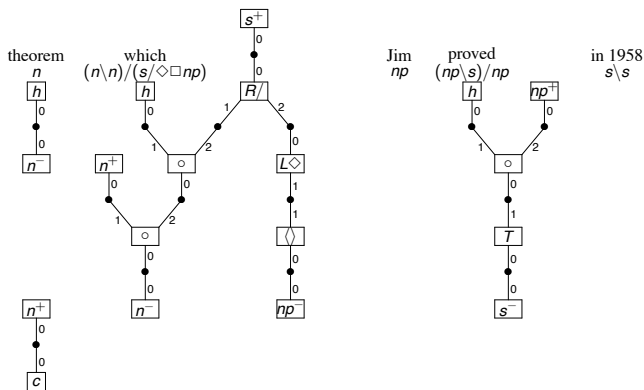
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



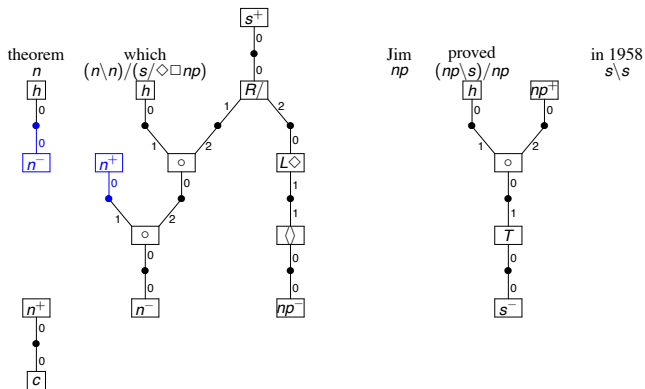
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



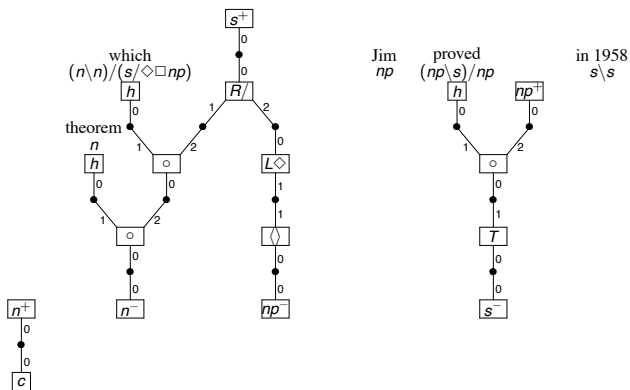
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



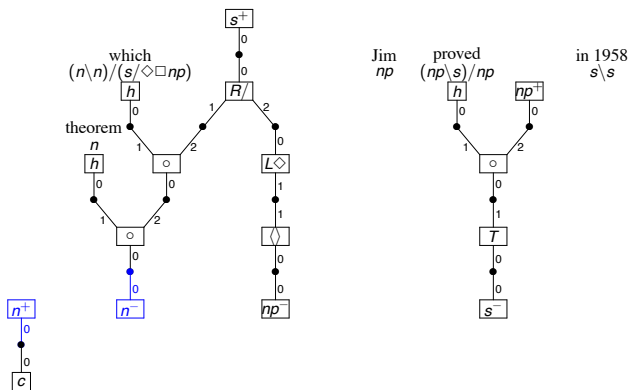
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



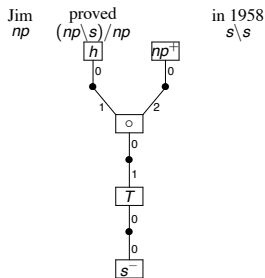
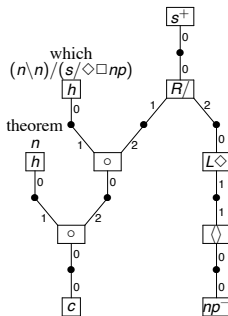
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



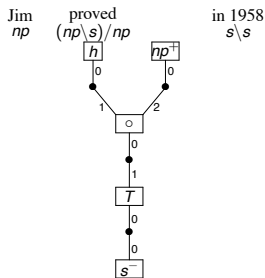
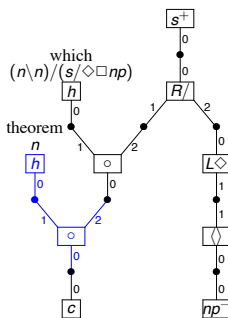
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



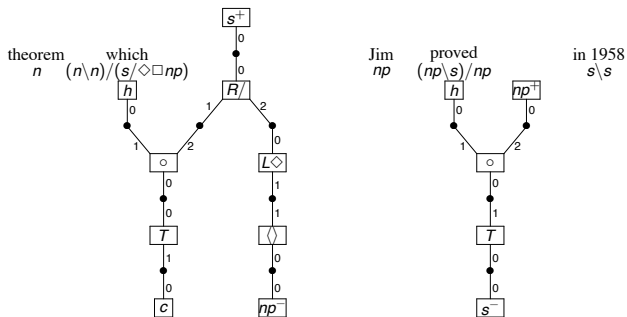
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



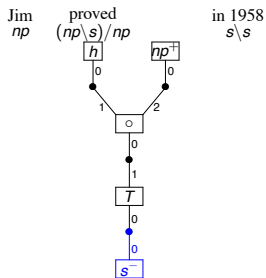
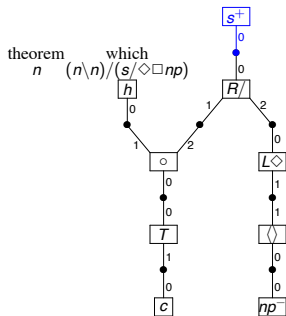
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



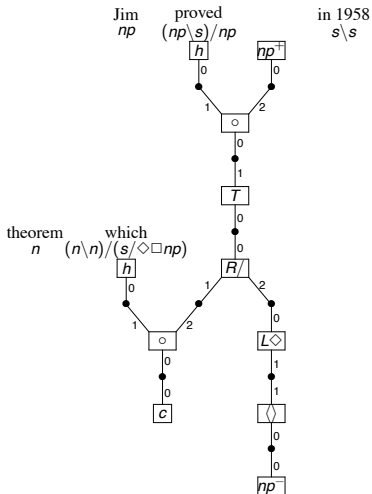
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



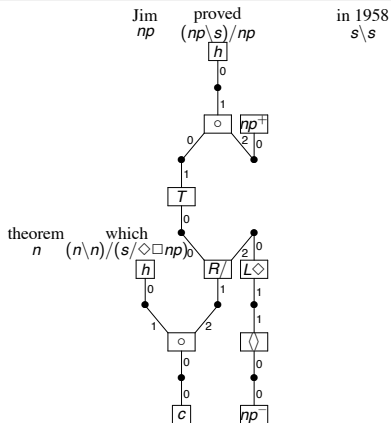
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



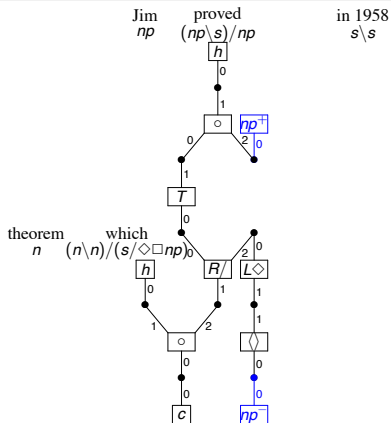
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



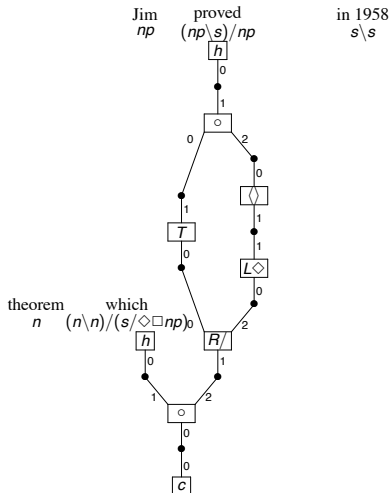
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



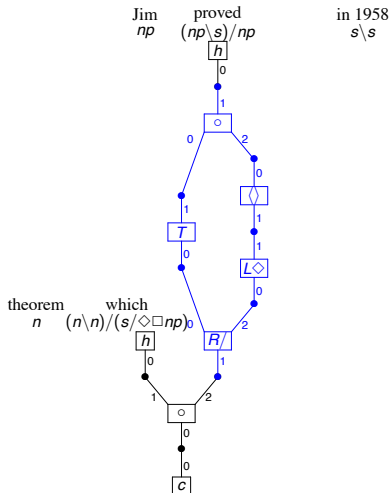
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



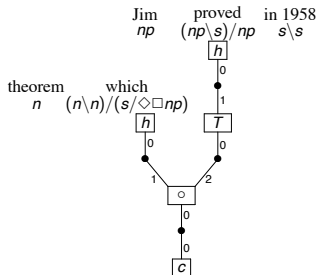
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



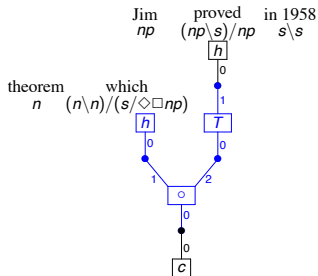
HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

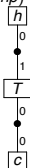
Parsing With Hyperedge Replacement Grammars



HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars

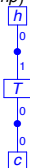
theorem which Jim proved in 1958
 n $(n \setminus n) / (s / \diamond \square np)$ np $(np \setminus s) / np$ $s \setminus s$



HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars

theorem which Jim proved in 1958
 n $(n \setminus n) / (s / \diamond \square np)$ np $(np \setminus s) / np$ $s \setminus s$



HRG and Proof Nets

Parsing With Hyperedge Replacement Grammars

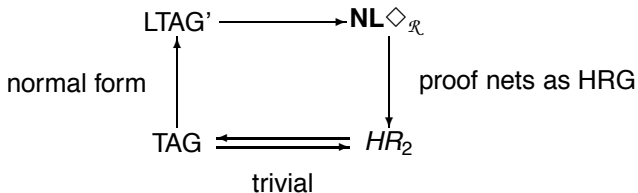
theorem which Jim proved in 1958
 n $(n \setminus n) / (s / \diamond \square np)$ np $(np \setminus s) / np$ $s \setminus s$

\square

Final Intermezzo

Relations between formalisms

adjunction as contraction
and structural rules



Conclusions

- We can extend **NL** with structural rules while keeping a polynomial parsing algorithm.
- The proposed extension can handle all cases which are problematic for the Syntactic Calculus, with the exception of scrambling, which we can do up to two levels just like TAG.
- The method sketched here can be extended to other packages of structural rules and calculi, notably the Lambek-Grishin calculus with the class IV interactions.
- Looking at graph grammars opens the way for radically new parsing algorithms for Lambek grammars.












Conclusions...

... And Work in Progress/Future Work!

- *Hyperion*, a compact parser for hyperedge replacement grammars is currently under development.
- For parsing, *apex grammars* and graph algorithms for graphs of *bounded treewidth* seem to offer intriguing new possibilities.
- The TAG translation gives analyses for many phenomena which differ radically from the standard multimodal analyses. Is there a way to bring these analyses closer to the known solutions.
- It is well-known that augmenting the rank of the hyperedge replacement grammar augments the string (and tree) generating power. For example, an tree-generating HRG of rank n can generate $2n$ counting dependencies. What are the corresponding categorial grammars?

References

-  Eades, M. & Courcelle, B. (1987), 'Graph expressions and graph rewritings', *Mathematical Systems Theory* **20**(1), 83–127.
-  Habel, A. & Kreowski, H.-J. (1987), May we introduce to you: Hyperedge replacement, in 'Graph Grammars and Their Application to Computer Science', Vol. 291 of *Lecture Notes in Computer Science*, Springer, pp. 15–26.
-  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.
-  Lautemann, C. (1990), 'The complexity of graph languages generated by hyperedge replacement', *Acta Informatica* **27**(5), 399–421.
-  Mootgat, M. & Oehle, R. T. (1994), Adjacency, dependency and order, in 'Proceedings 9th Amsterdam Colloquium', pp. 447–466.
-  Moot, R. (2002), Proof Nets for Linguistic Analysis, PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University.
-  Moot, R. & Puite, Q. (2002), 'Proof nets for the multimodal Lambek calculus', *Studia Logica* **71**(3), 415–442.
-  Wolf, D. (1992), Linear context-free rewriting systems and deterministic tree-walking transducers, in 'Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics', Association for Computational Linguistics, Morristown, New Jersey, pp. 136–143.