Lambek Grammars, Tree Adjoining Grammars and Hyperedge Replacement Grammars

Richard Moot

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

TAG+ Workshop, Tuebingen, 7-8 June 2008

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.
- Kandulski (1988) showed that **NL** generates only context-free languages.
- On the positive side, though, de Groote (1999) showed we can parse **NL** grammars in polynomial time.

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

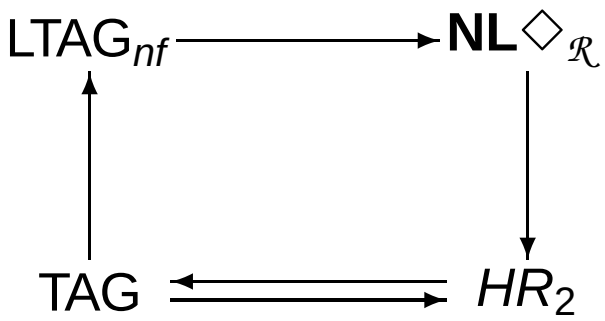
- In terms of practical parsing, a PSPACE bound is still quite high.
- We would like to find a restriction of the multimodal calculus which extends **NL**, but does so while keeping a polynomial parsing algorithm.
- The *mildly* context-sensitive languages and the different corresponding formal systems (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

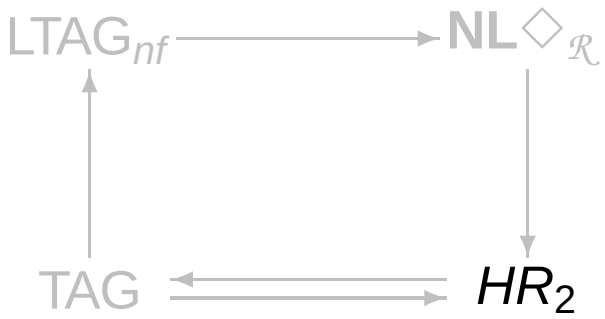
Overview of Lambek Calculi: Logic, Language, Complexity

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

Overview
Relations



Overview
Relations



Hyperedge Replacement Grammars

Introduction

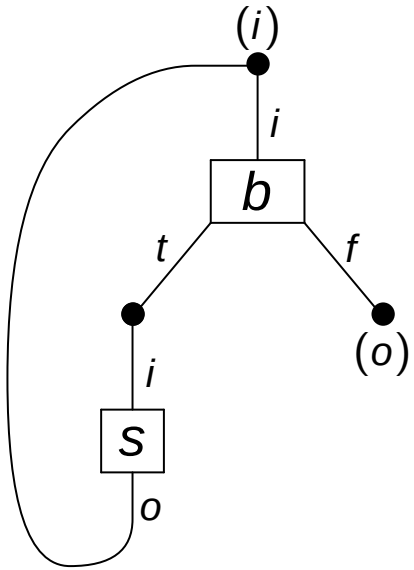
Hyperedge replacement grammars were introduced by Bauderon & Courcelle (1987) and Habel & Kreowski (1987) as a type of context-free graph grammars. The string, tree and graph languages generated by HRG have been well-studied.
Because of

- 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

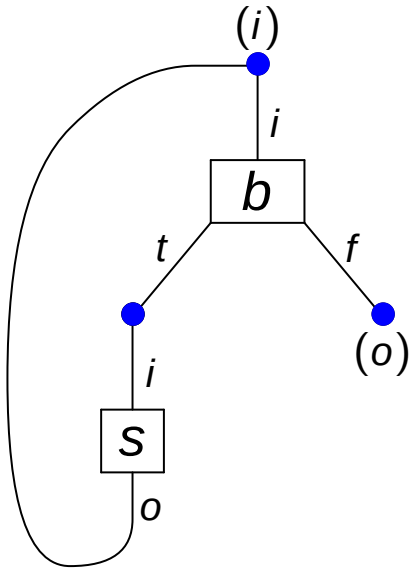
Hypergraphs With External Nodes



Hyperedge Replacement Grammars

Hypergraphs With External Nodes

– vertices are drawn as fat dots.

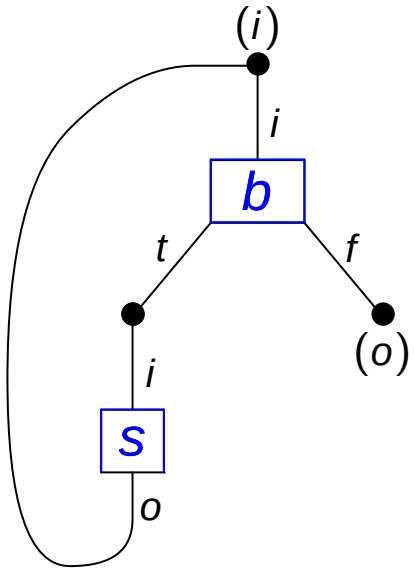


Hyperedge Replacement Grammars

Hypergraphs With External Nodes

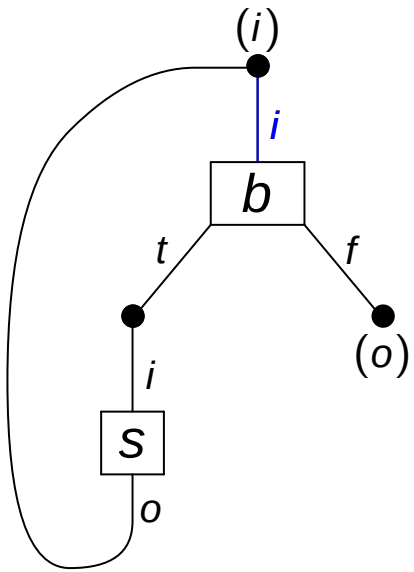
– vertices are drawn as fat dots.

– hyperedges are labeled rectangles.



Hyperedge Replacement Grammars

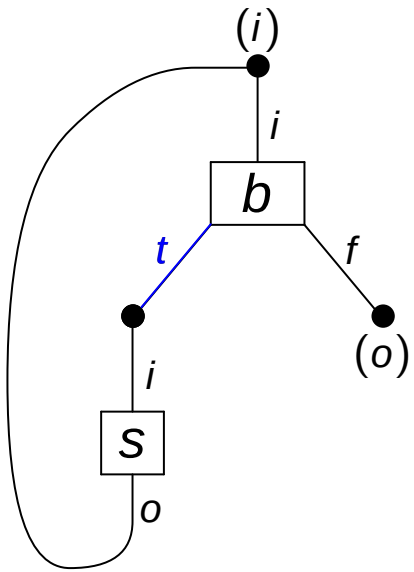
Hypergraphs With External Nodes



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

Hyperedge Replacement Grammars

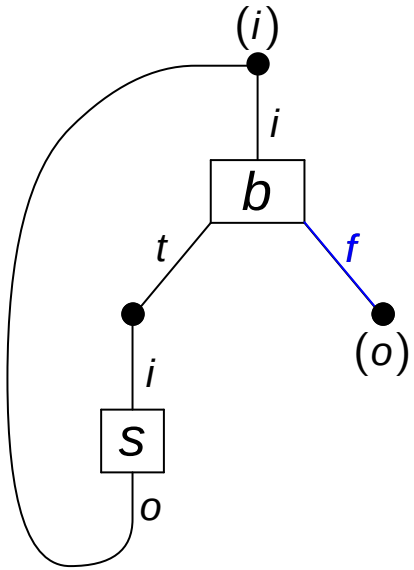
Hypergraphs With External Nodes



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

Hyperedge Replacement Grammars

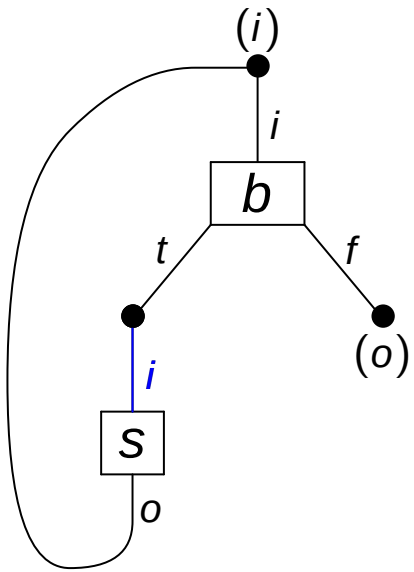
Hypergraphs With External Nodes



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

Hyperedge Replacement Grammars

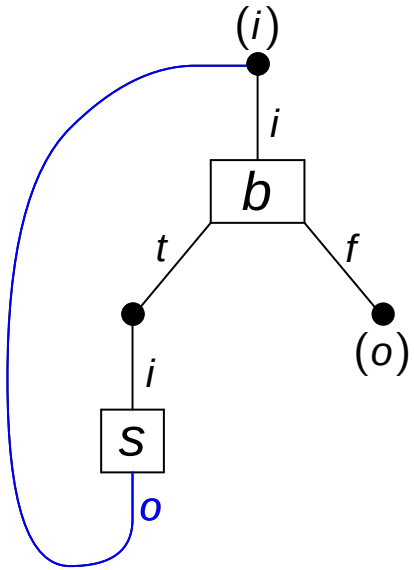
Hypergraphs With External Nodes



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

Hyperedge Replacement Grammars

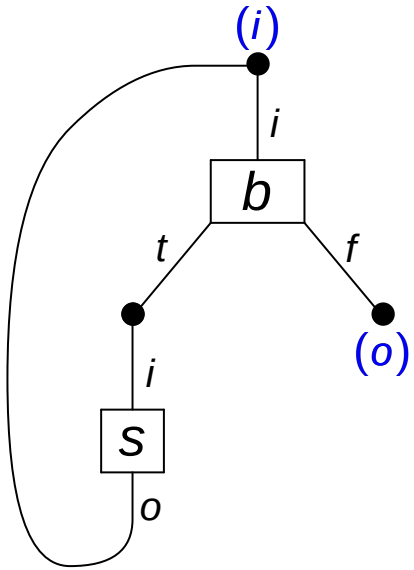
Hypergraphs With External Nodes



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

Hyperedge Replacement Grammars

Hypergraphs With External Nodes

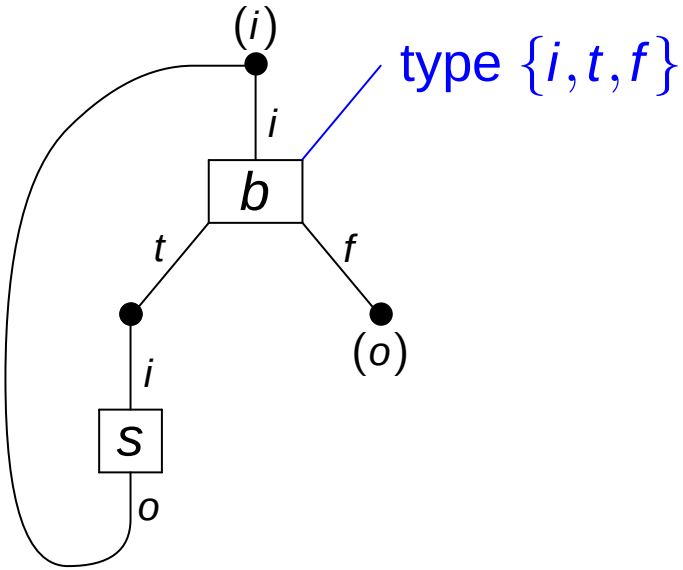


- vertices are drawn as fat dots.
- hyperedges are labeled rectangles.
- labeled ‘tentacles’ connect hyperedges and vertices.
- some vertices of the hypergraph are labeled as external.

Hyperedge Replacement Grammars

Hypergraphs With External Nodes — Types

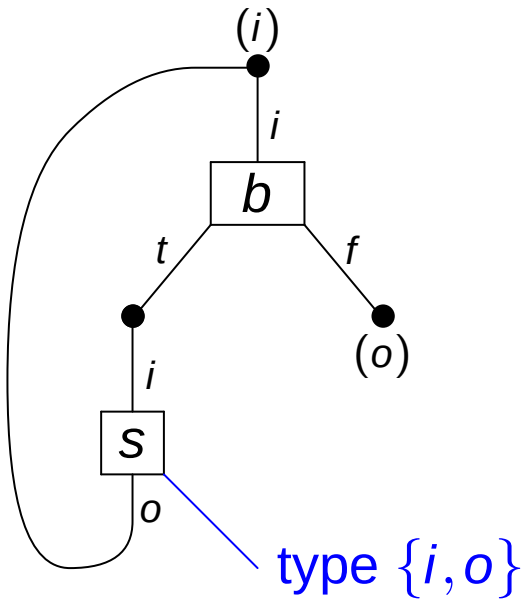
– the *type* of a hyperedge is the set of its tentacle labels.



Hyperedge Replacement Grammars

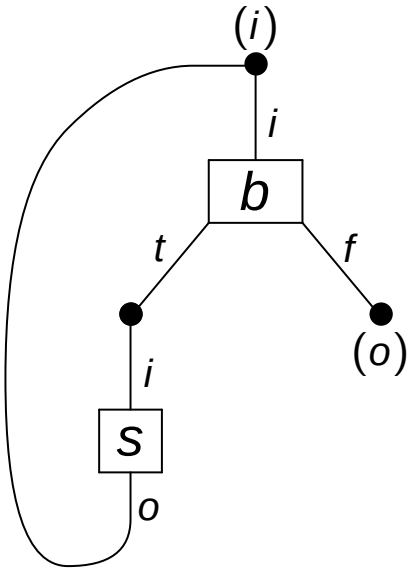
Hypergraphs With External Nodes — Types

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



Hyperedge Replacement Grammars

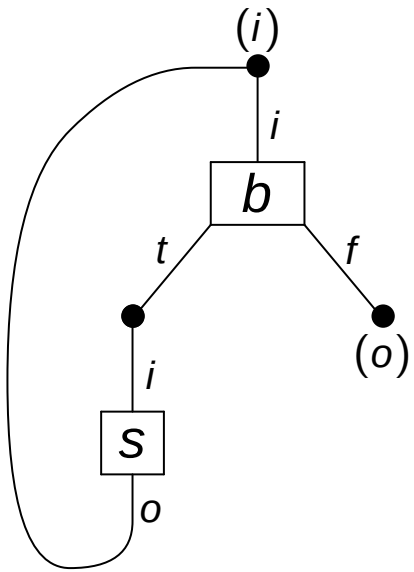
Hypergraphs With External Nodes — Types



- 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

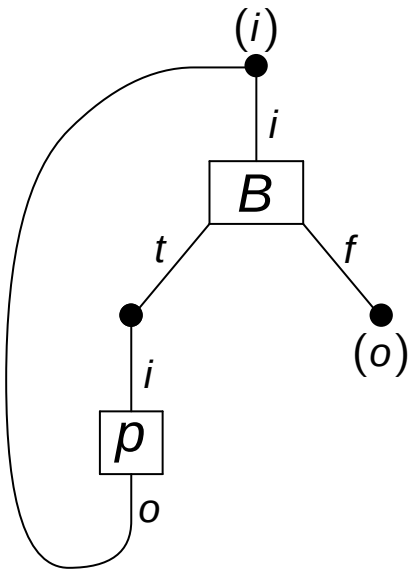
Hypergraphs With External Nodes — Types



- 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

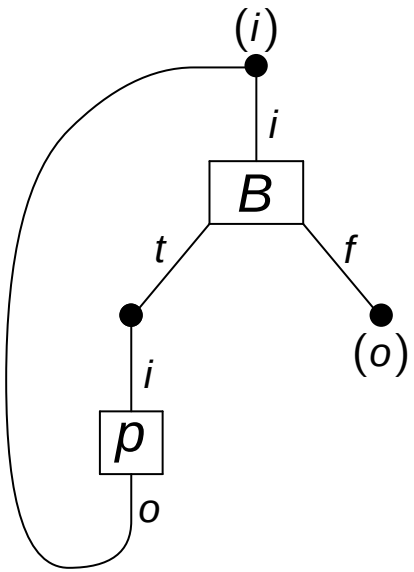
Hyperedge Replacement



H

Hyperedge Replacement Grammars

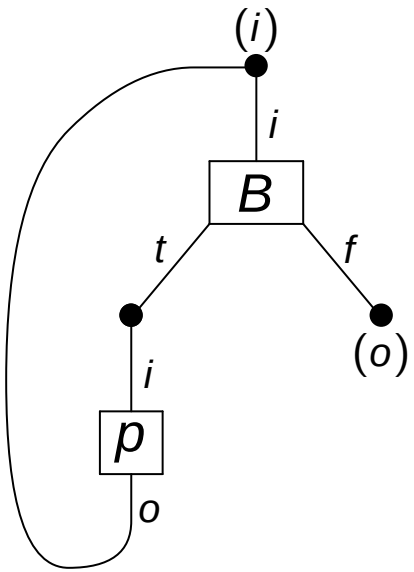
Hyperedge Replacement

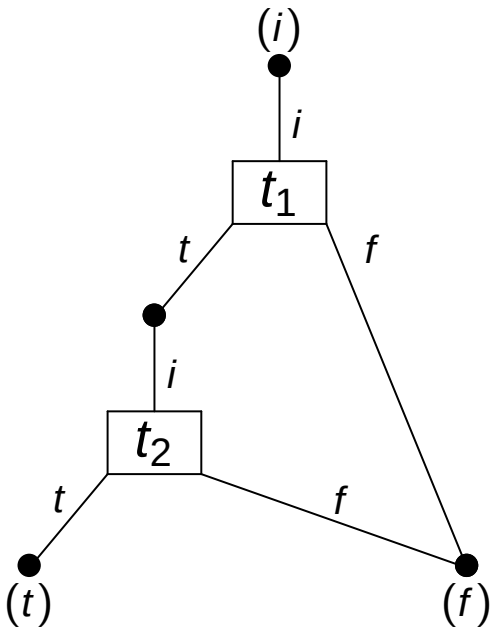


edge e (labeled B)
of type $\{i, t, f\}$
H

Hyperedge Replacement Grammars

Hyperedge Replacement



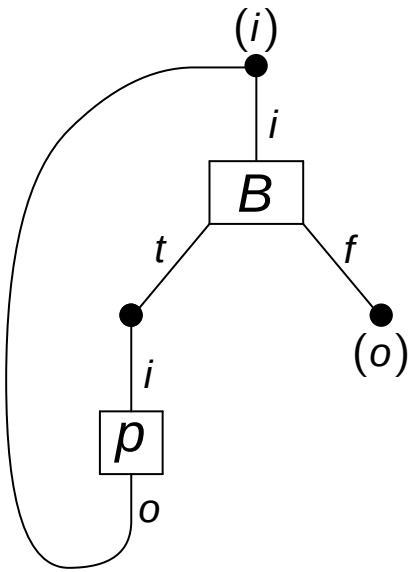


edge e (labeled B)
of type $\{i, t, f\}$

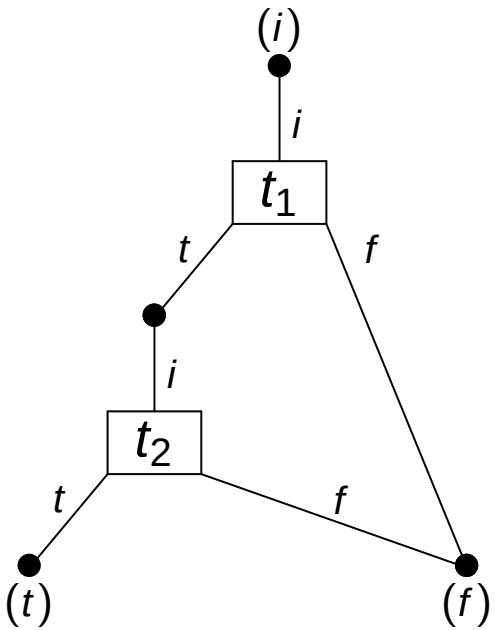
H K

Hyperedge Replacement Grammars

Hyperedge Replacement



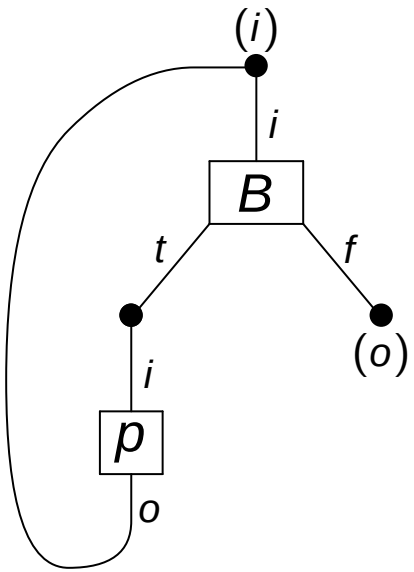
edge e (labeled B)
of type $\{i, t, f\}$
H



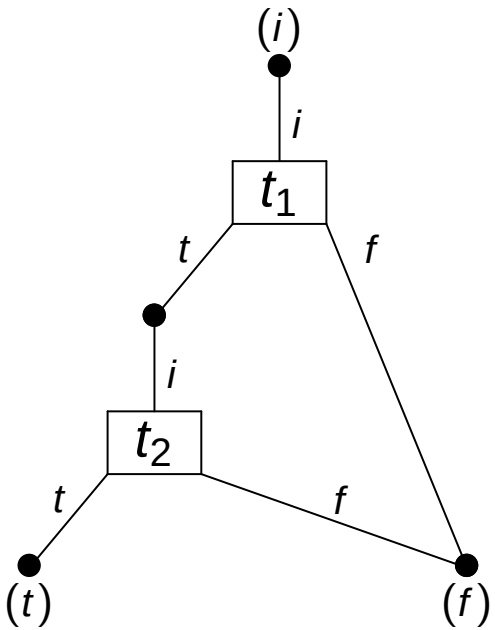
hypergraph K
of type $\{i, t, f\}$
K

Hyperedge Replacement Grammars

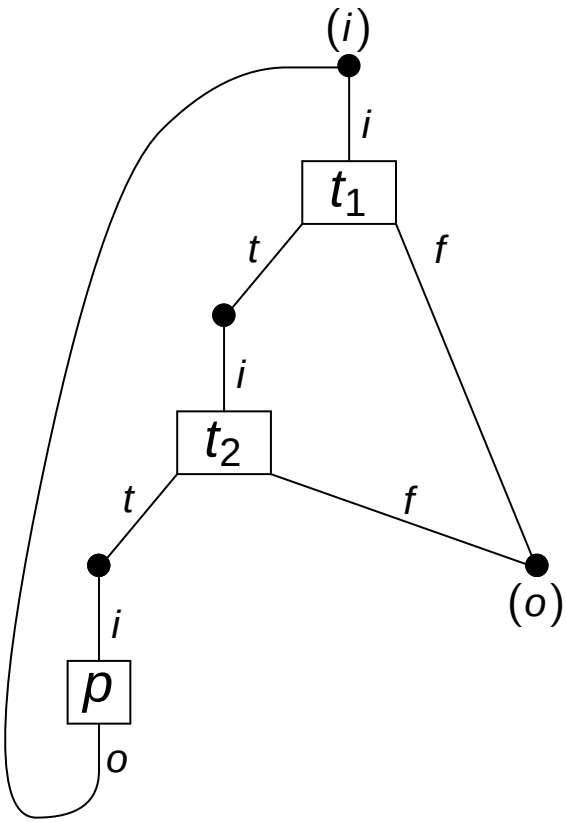
Hyperedge Replacement



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



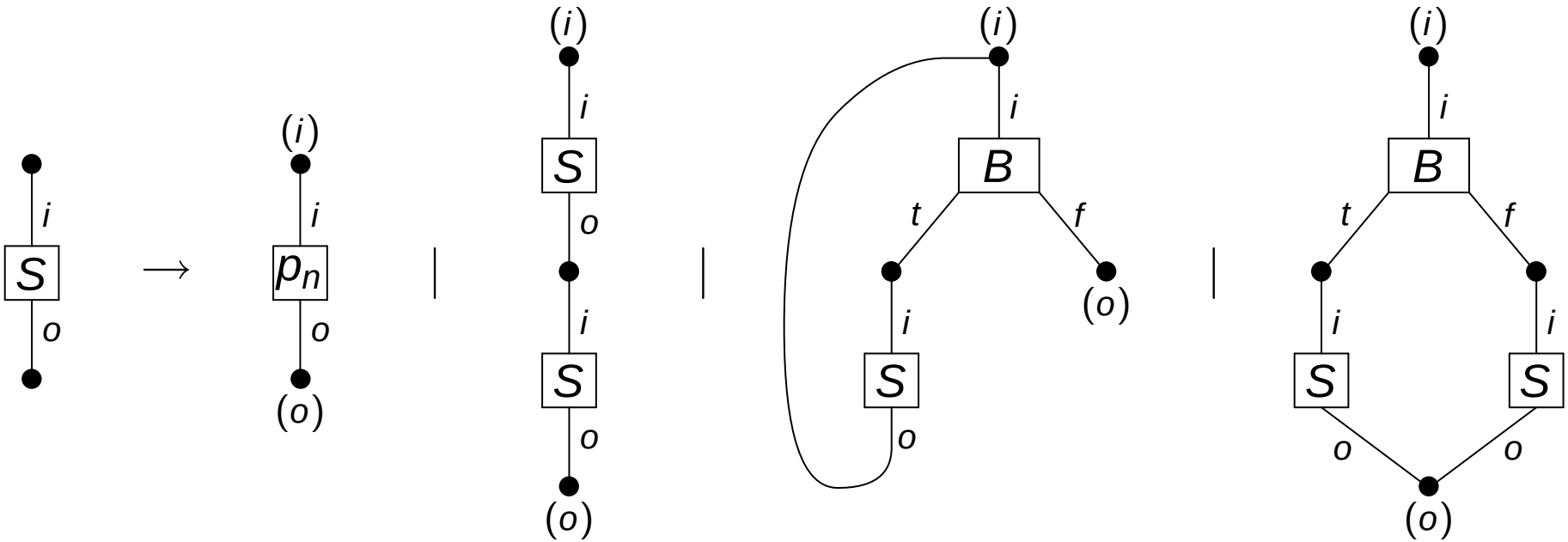
hypergraph K
of type $\{i, t, f\}$
K



$H[e := K]$

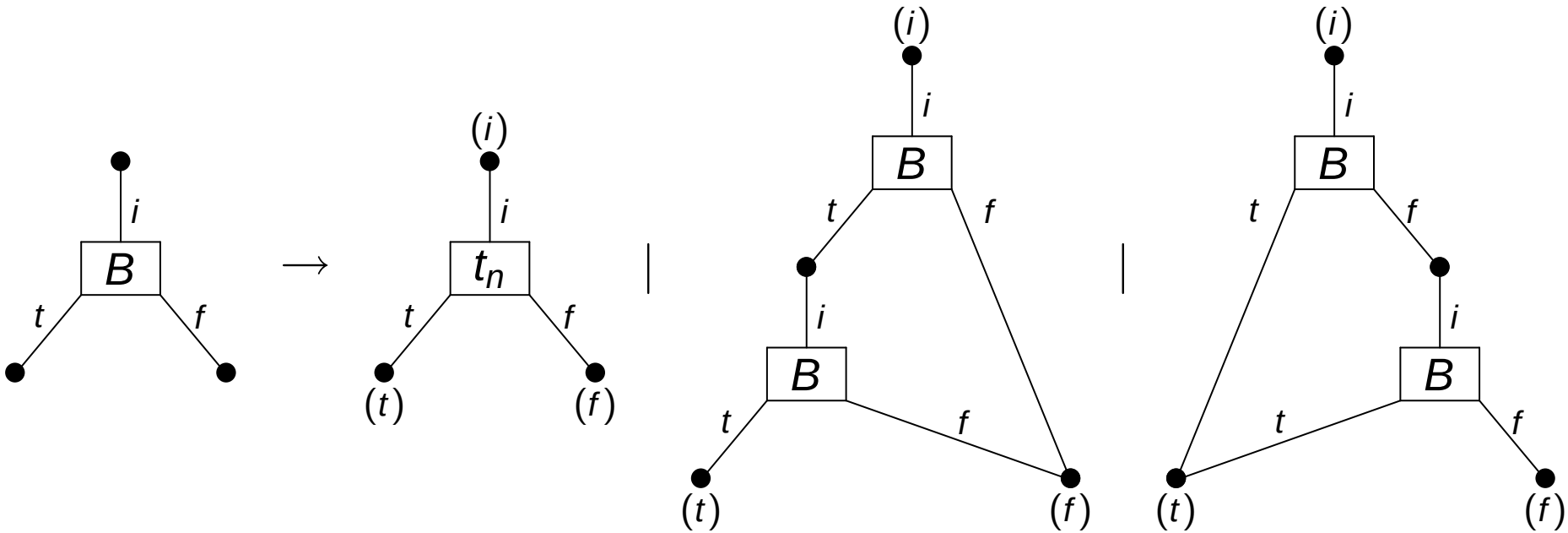
Hyperedge Replacement Grammars

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 as HR Grammars

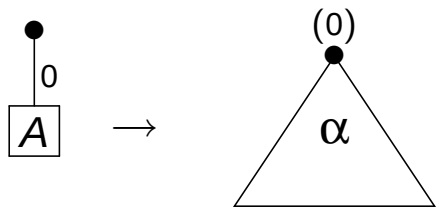
Introduction

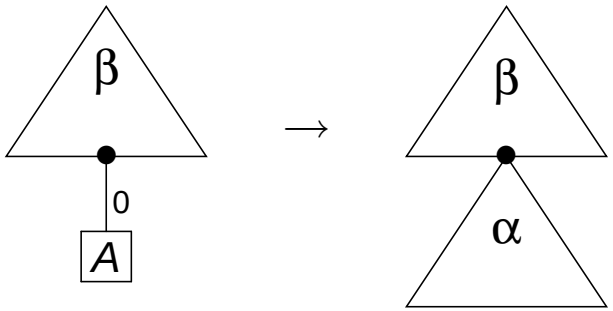
In the context of HR grammars, Tree Adjoining Grammars can be see as a special case of hyperedge replacement grammars where:

- 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

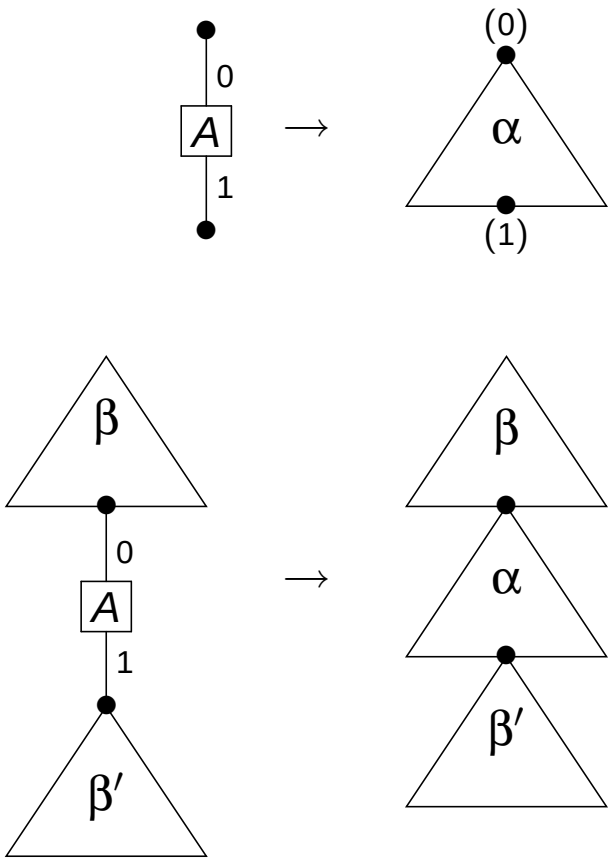
Substitution





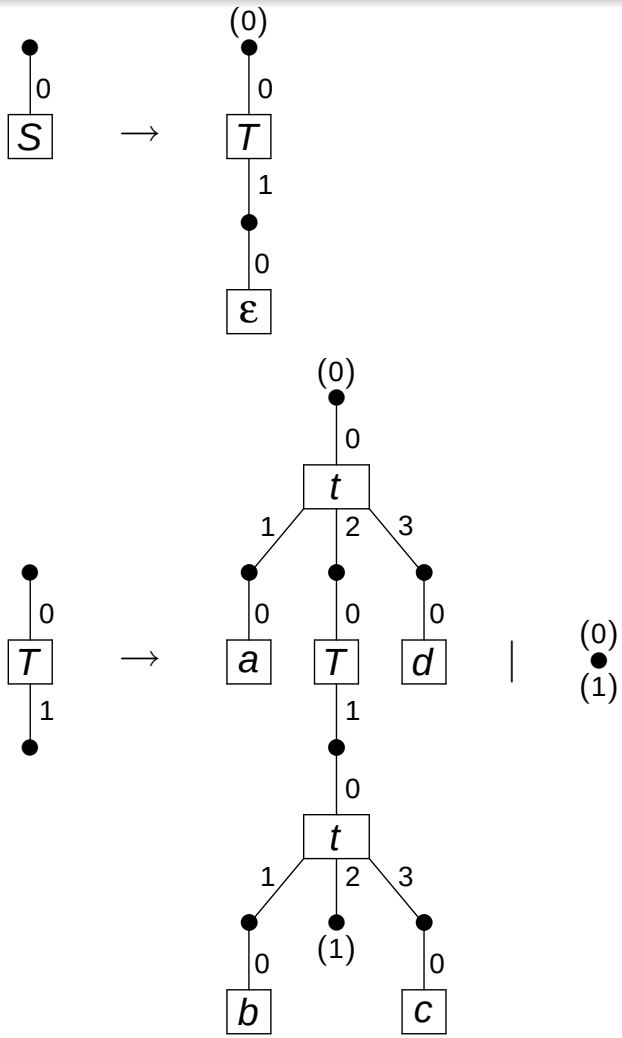
Tree Adjoining Grammars as HR Grammars

Adjunction



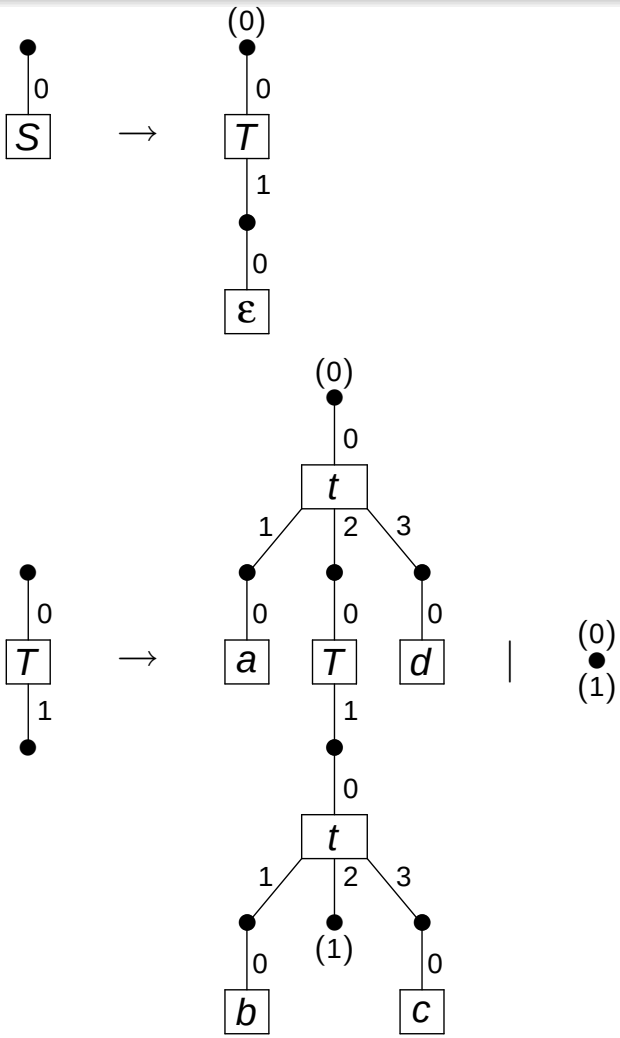
Tree Adjoining Grammars as HR Grammars

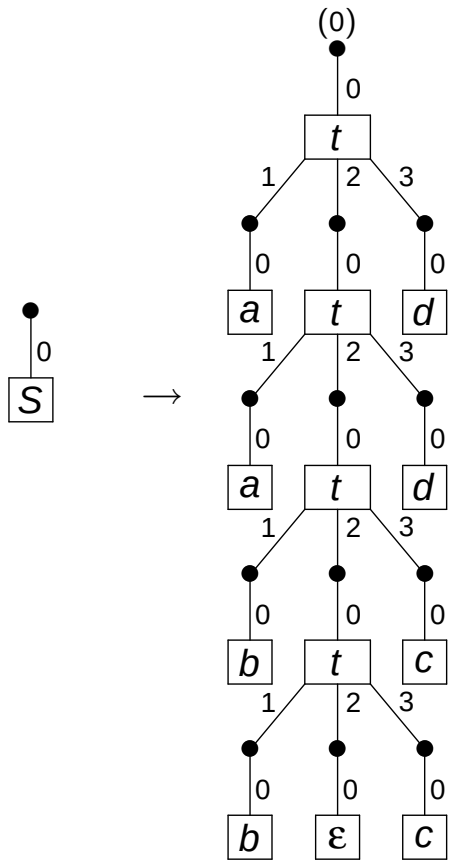
Example: $a^n b^n c^n d^n$



Tree Adjoining Grammars as HR Grammars

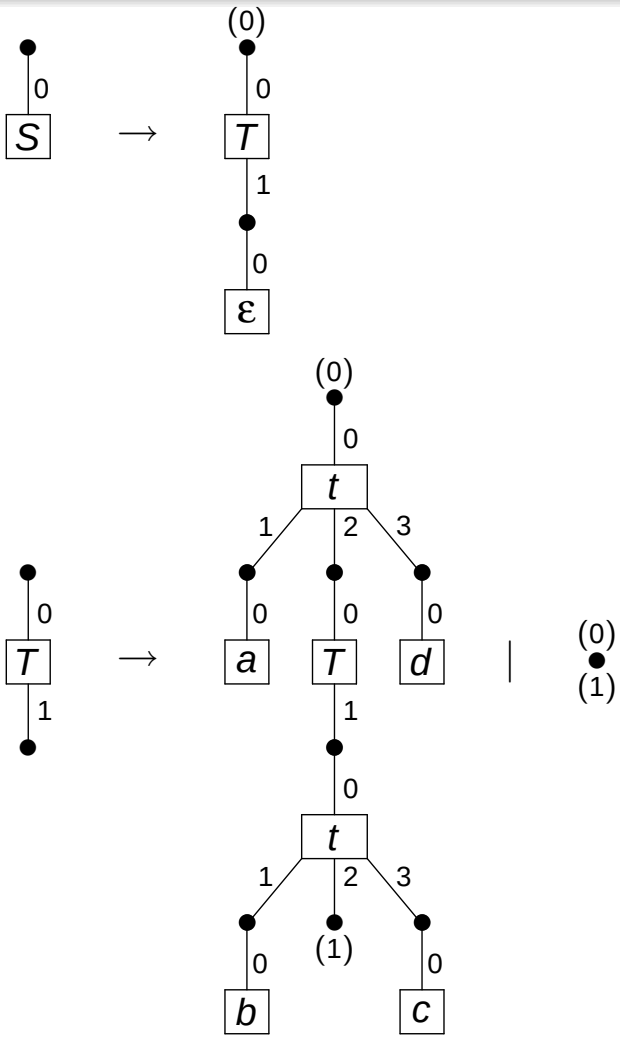
Example: $a^n b^n c^n d^n$

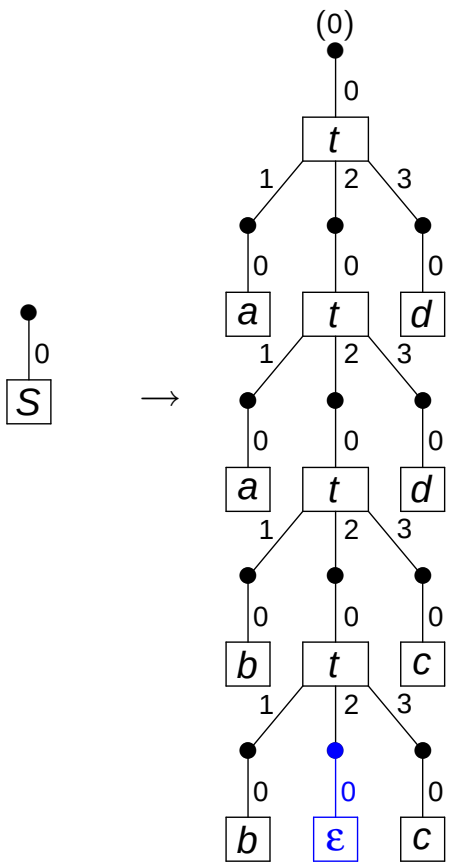




Tree Adjoining Grammars as HR Grammars

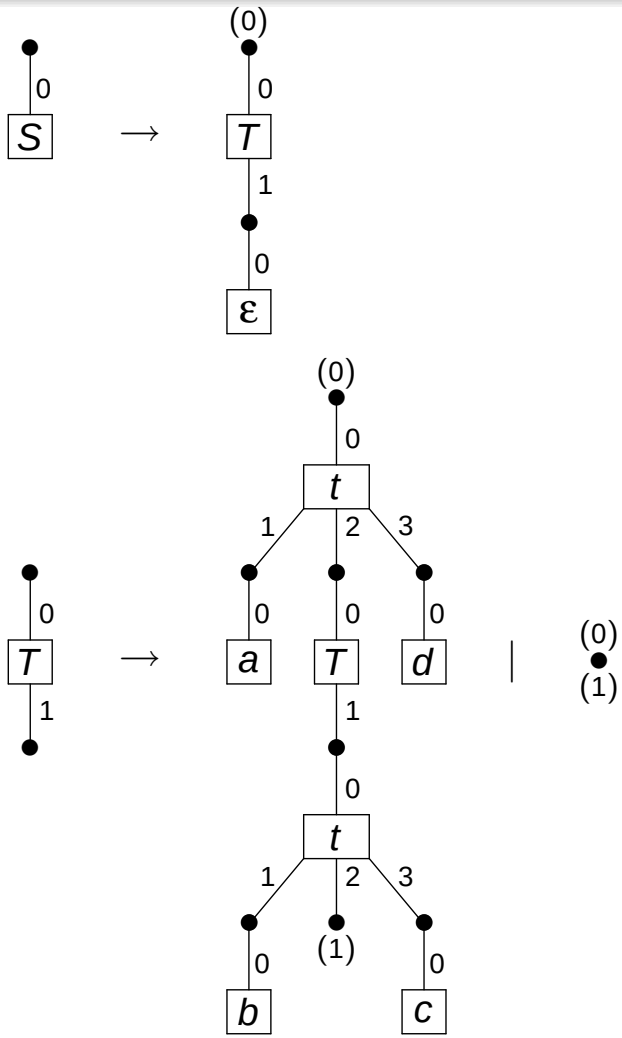
Example: $a^n b^n c^n d^n$

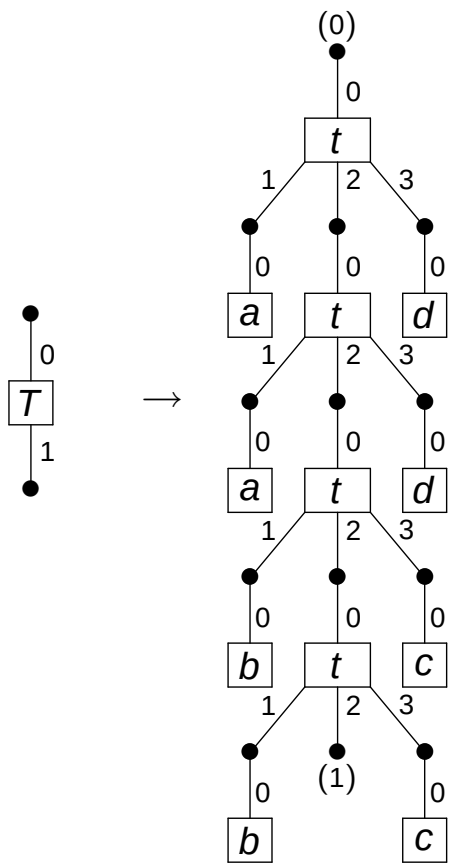




Tree Adjoining Grammars as HR Grammars

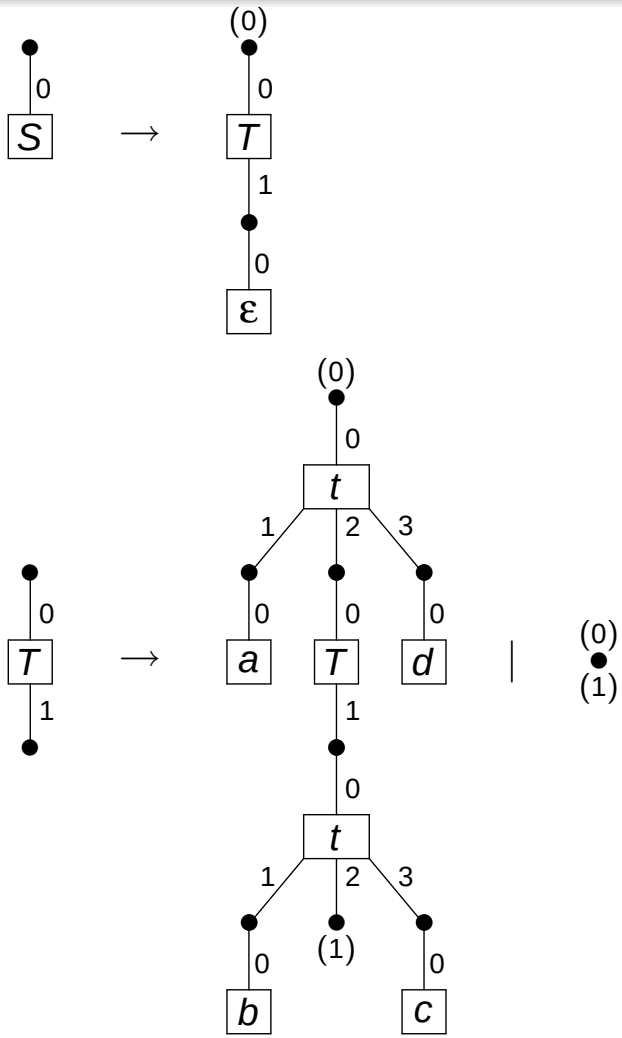
Example: $a^n b^n c^n d^n$

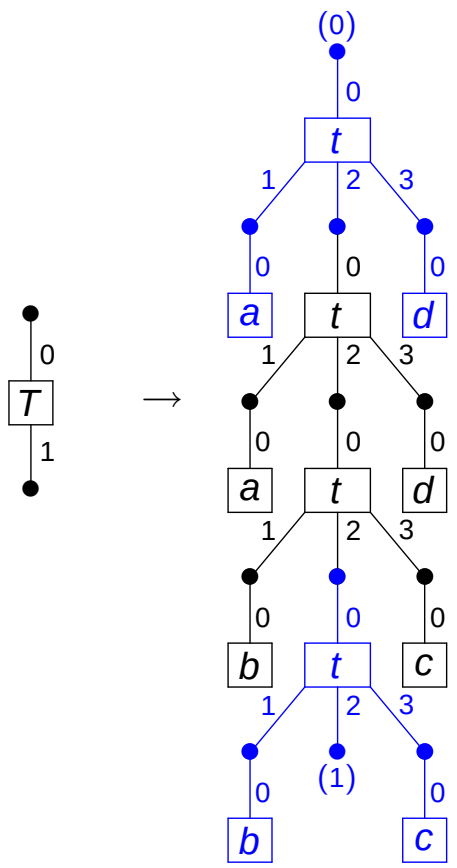




Tree Adjoining Grammars as HR Grammars

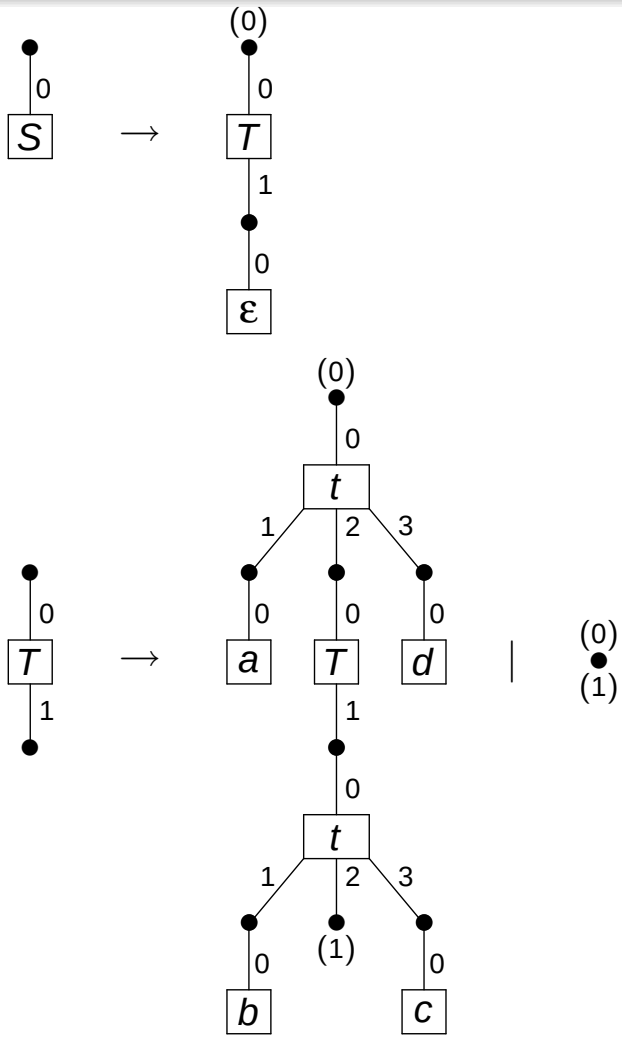
Example: $a^n b^n c^n d^n$

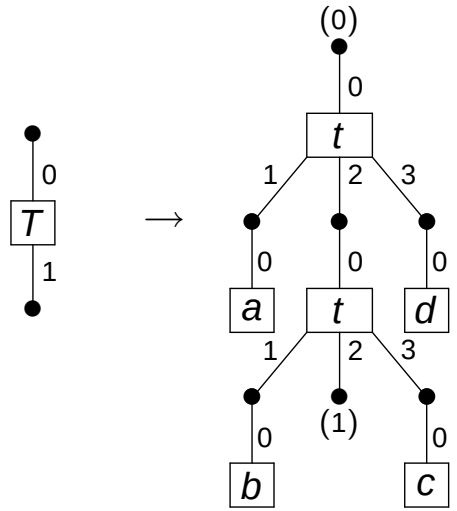




Tree Adjoining Grammars as HR Grammars

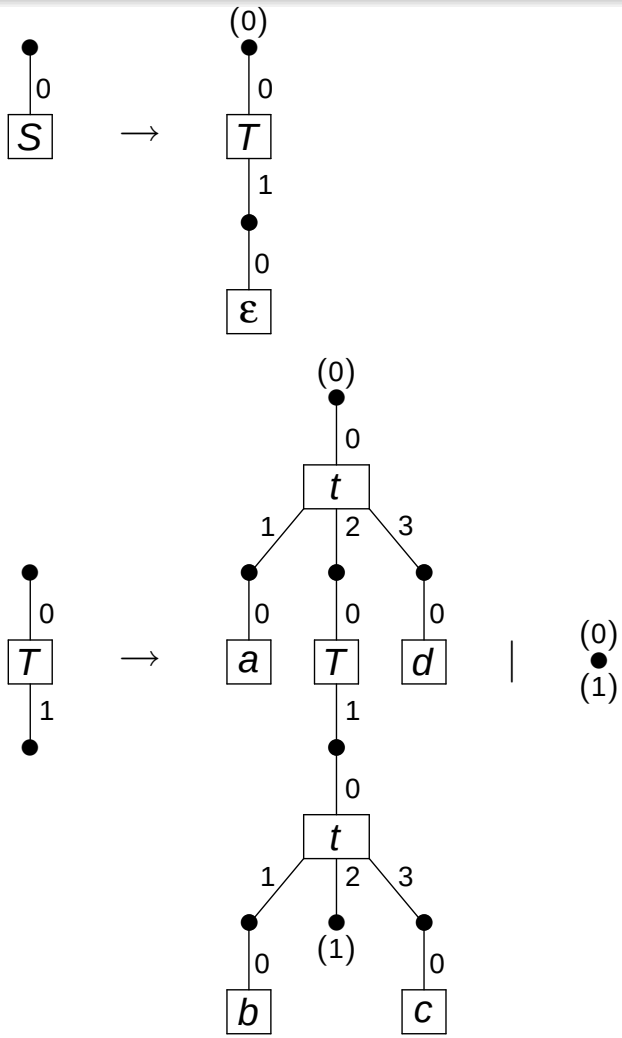
Example: $a^n b^n c^n d^n$

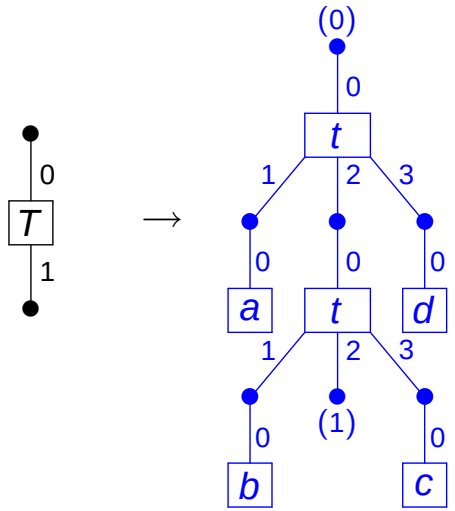




Tree Adjoining Grammars as HR Grammars

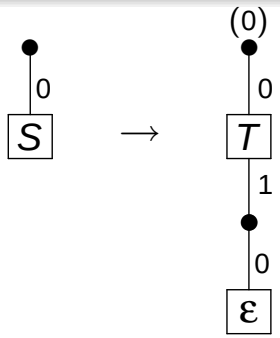
Example: $a^n b^n c^n d^n$

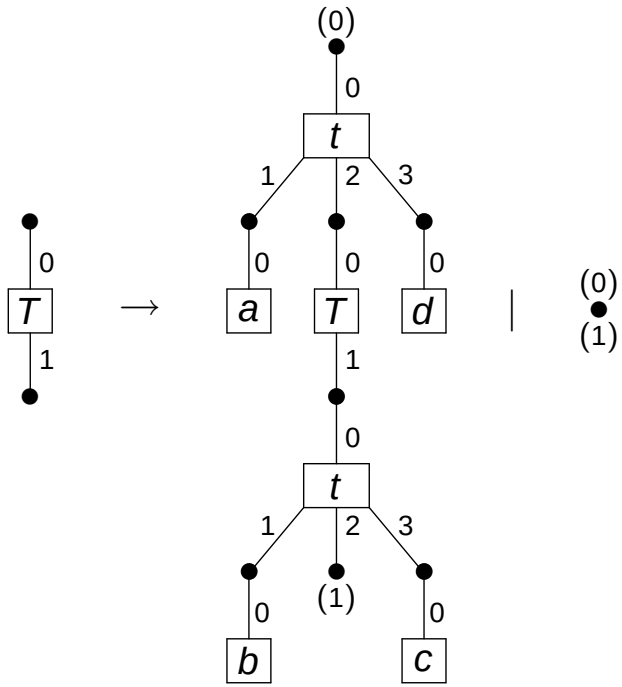


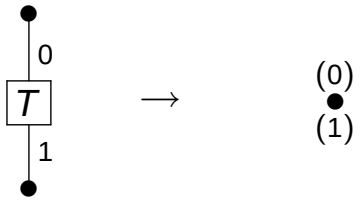


Tree Adjoining Grammars as HR Grammars

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

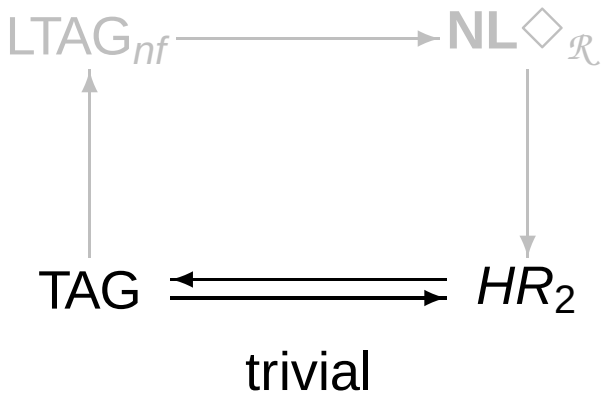






Intermezzo

Relations so far



Tree Adjoining Grammars

Introduction

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

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.

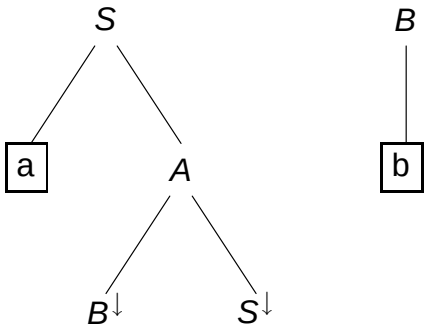
Tree Adjoining Grammars

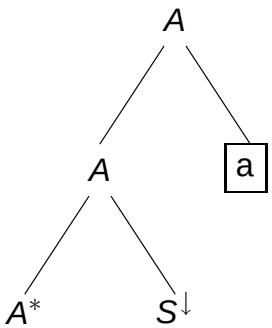
A Normal Form for LTAGs

- 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.
- Transforming an LTAG into a weakly equivalent LTAG in normal form is very similar to transforming an LTAG into spinal form.

Tree Adjoining Grammars

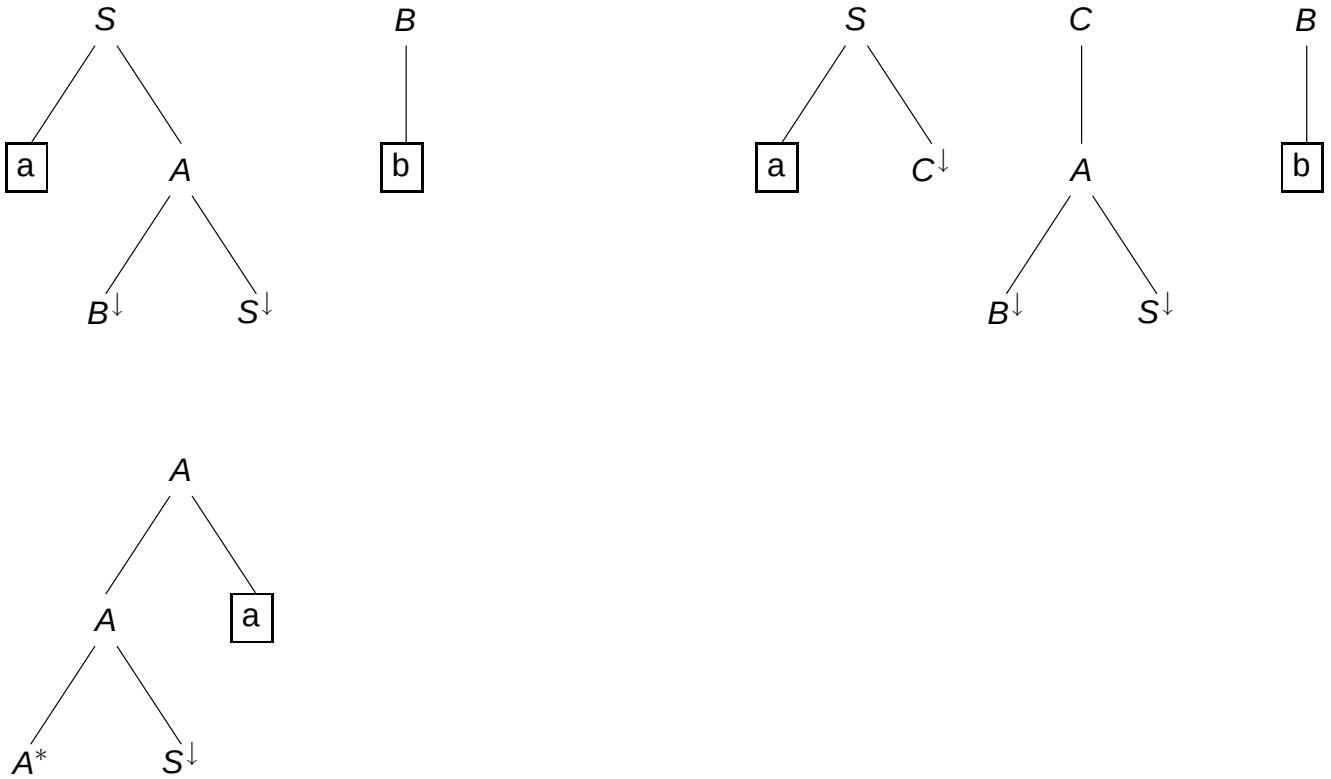
Example: Transforming an LTAG into Normal Form





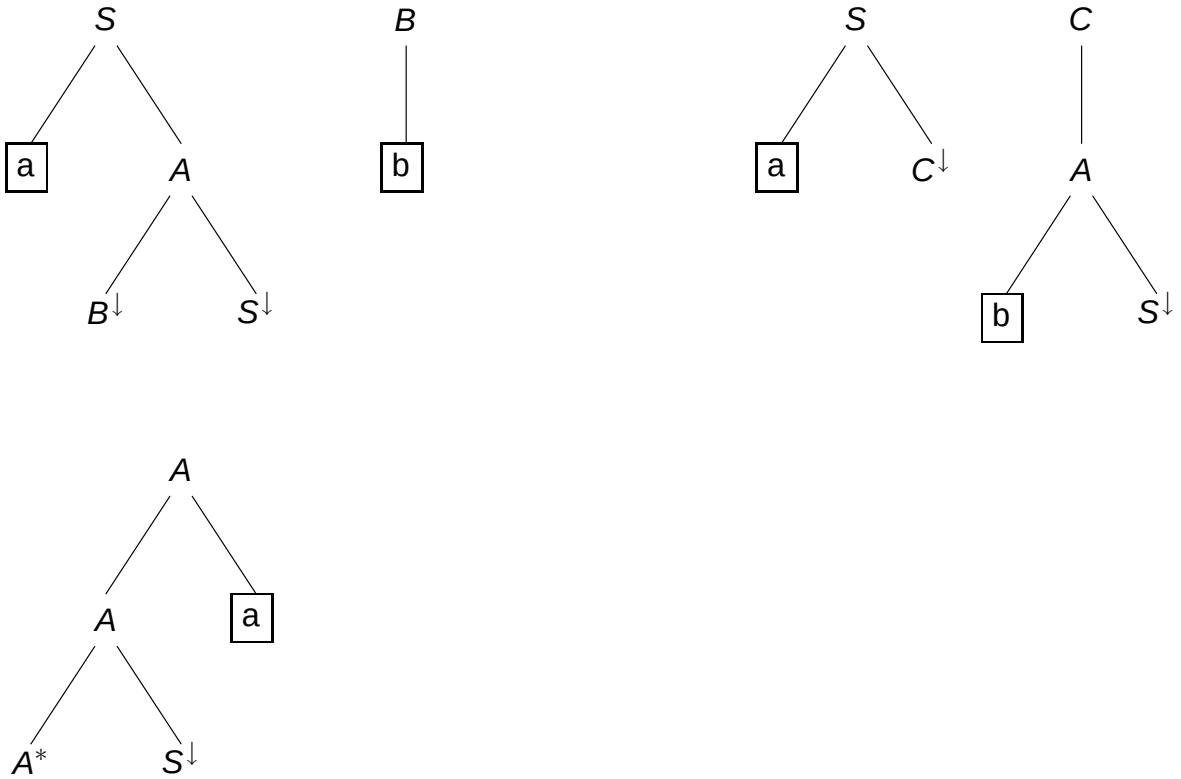
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



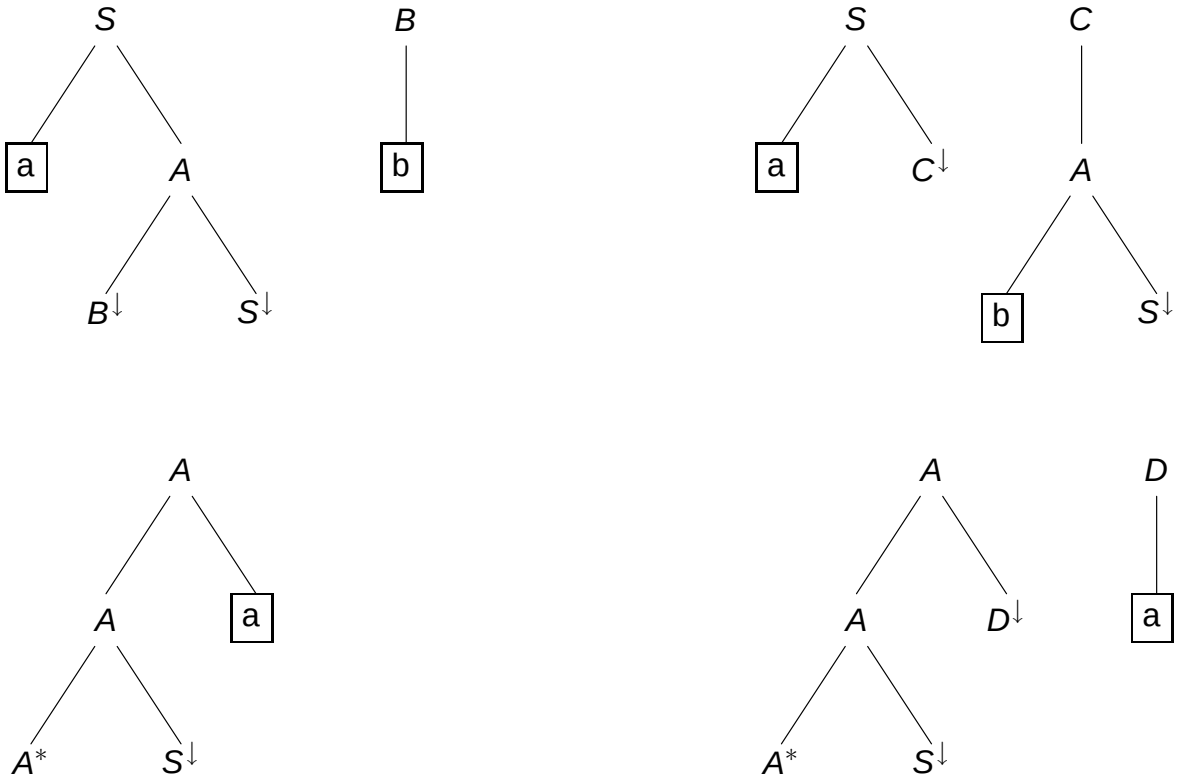
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



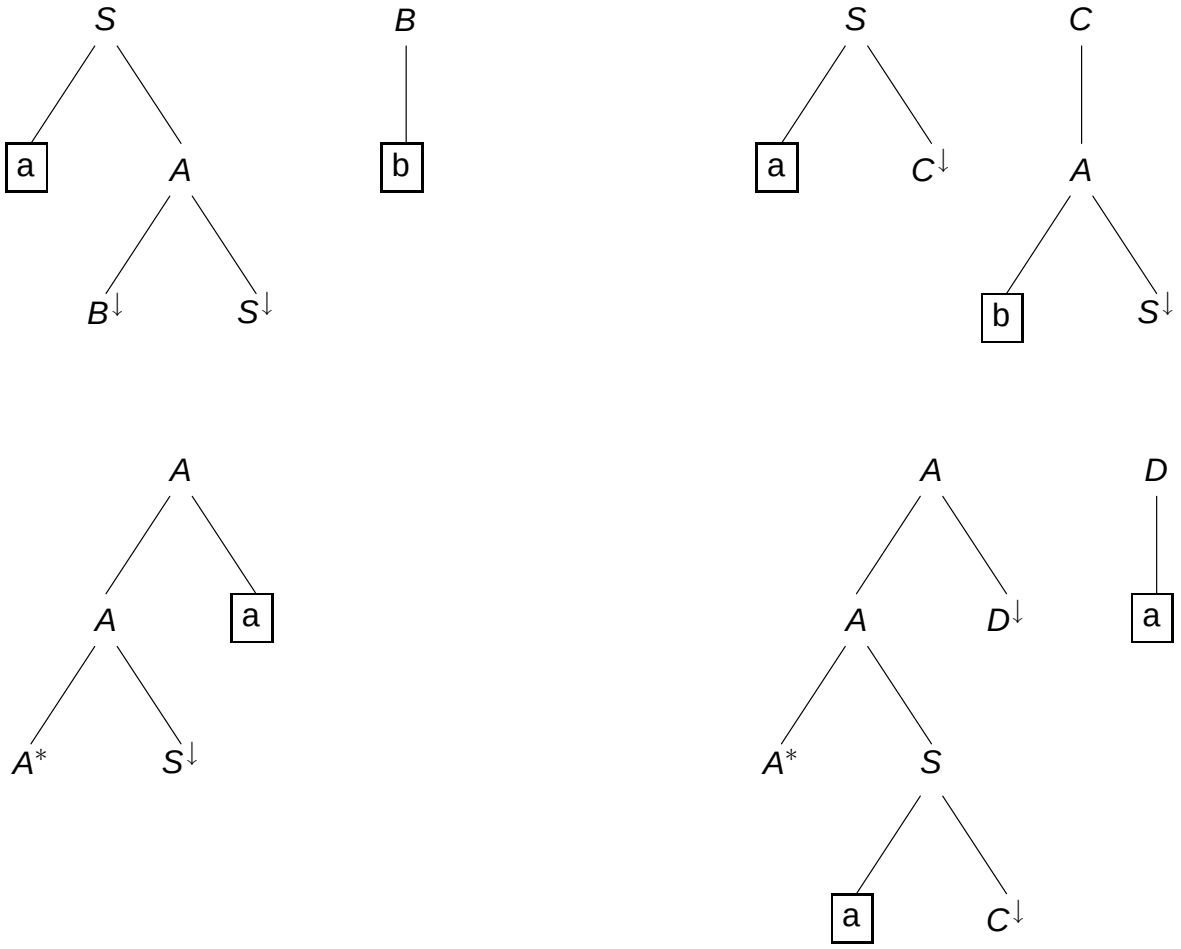
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



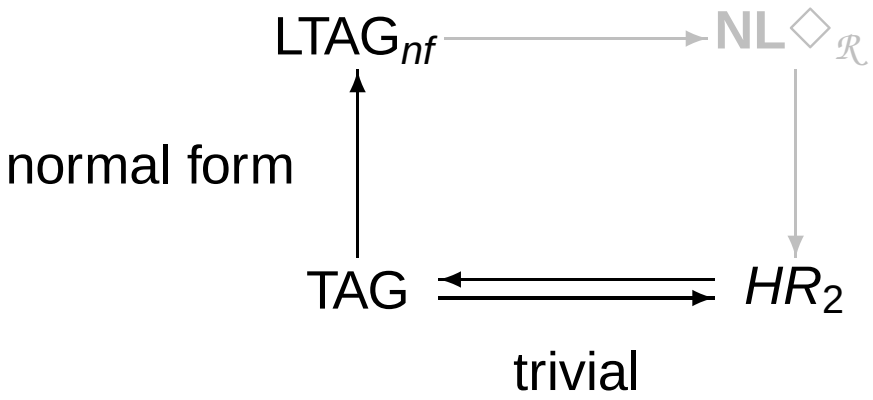
Tree Adjoining Grammars

Example: Transforming an LTAG into Normal Form



Intermezzo

Relations so far



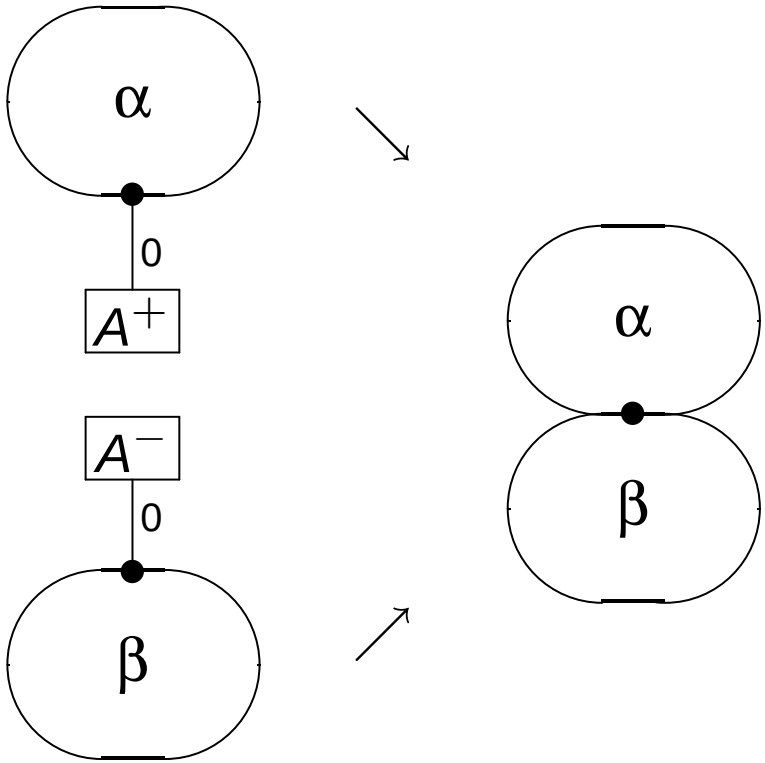
Proof Nets

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).
- The current presentation of proof nets is very close to the partial proof trees of Joshi & Kulick (1997): we unfold lexical entries depending on their lexical formula and combine the resulting trees using the axiom rule (called *application* or *substitution* in the work on partial proof trees).
- Our main difference is in the way we model adjunction or *stretching*.

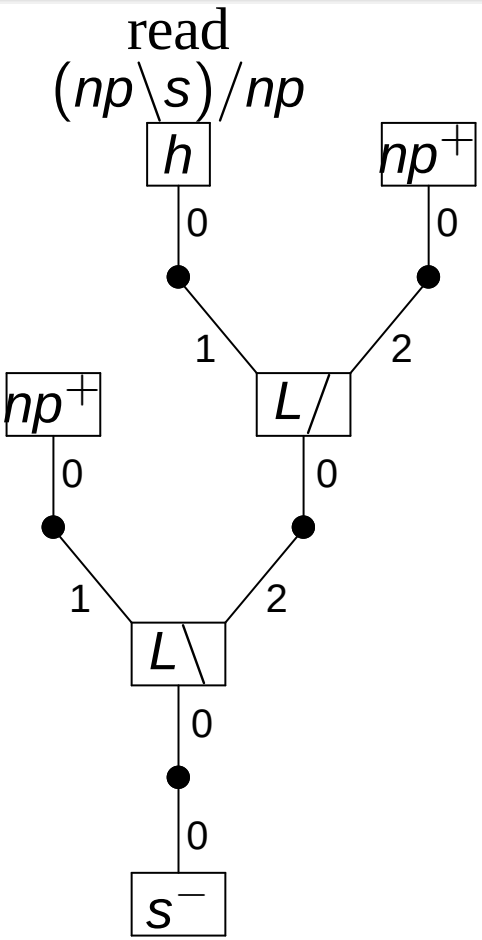
Proof Nets for AB

The Axiom Rule



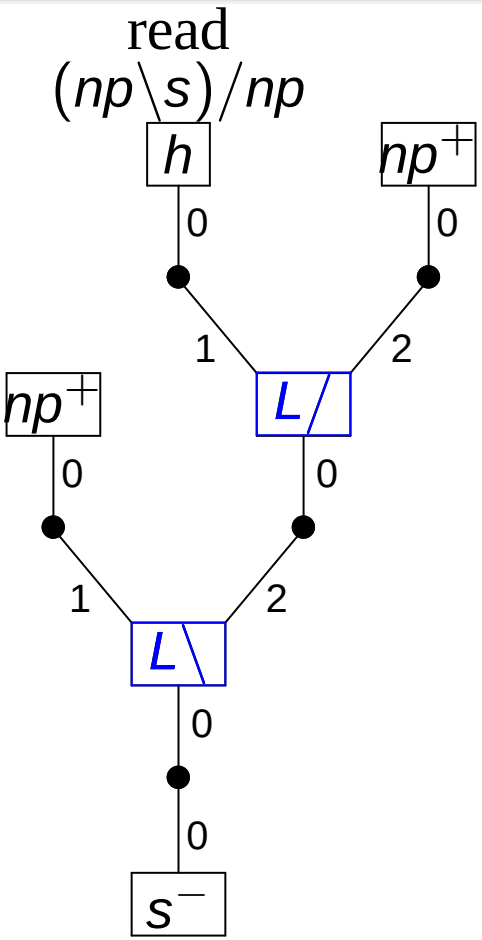
Proof Nets for AB

The Anatomy of a Lexical Entry



Proof Nets for AB

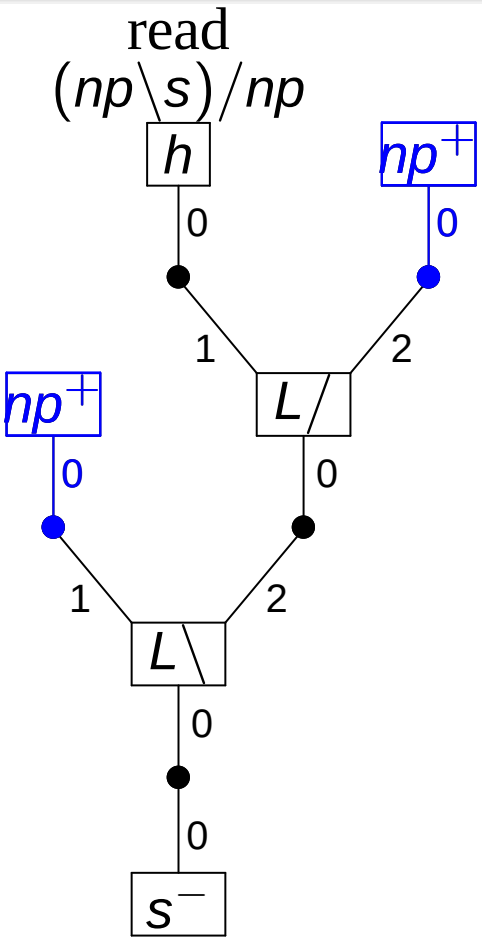
The Anatomy of a Lexical Entry



– ternary hyperedges are labeled by rule names

Proof Nets for AB

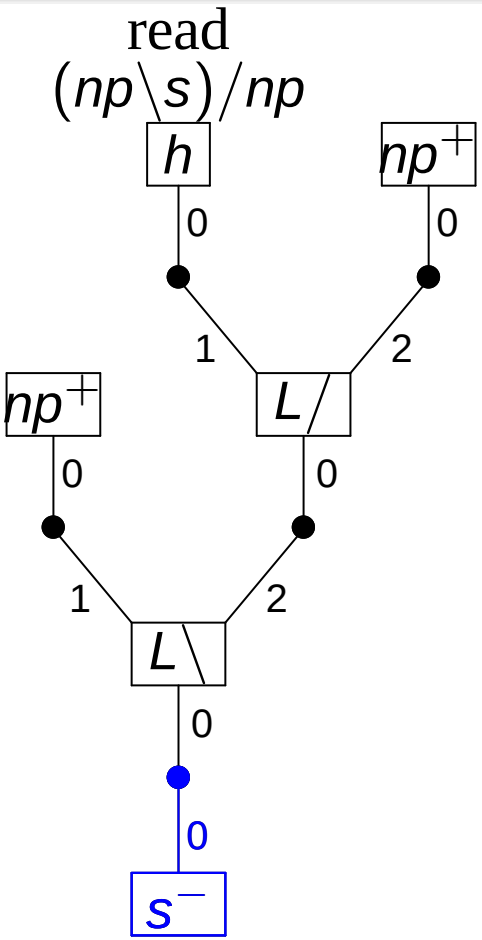
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates a tree with root np^- needs to be substituted here

Proof Nets for AB

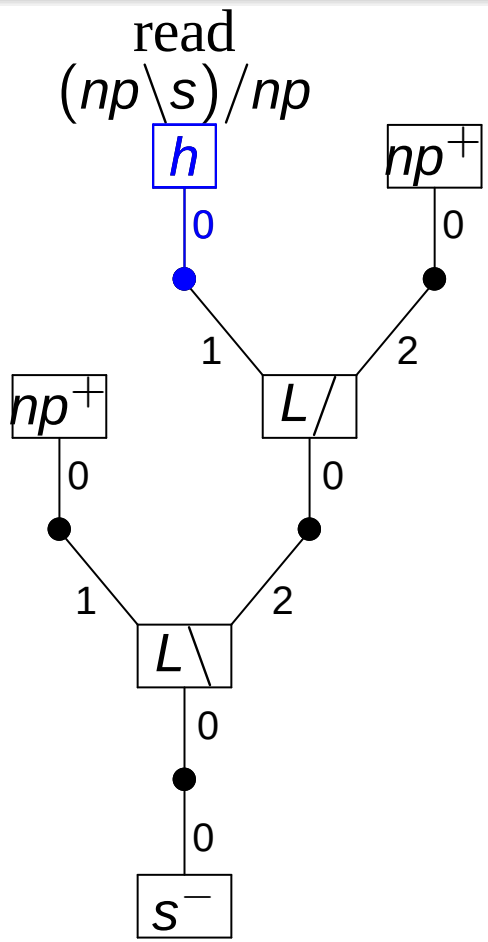
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates a tree with root np^- needs to be substituted here
- inversely, s^- indicates we can subtitute this tree for an s^+

Proof Nets for AB

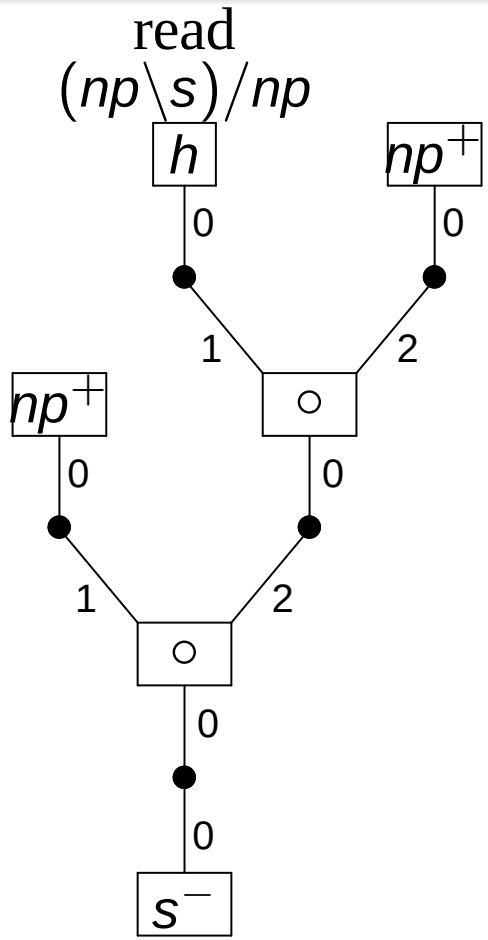
The Anatomy of a Lexical Entry



- ternary hyperedges are labeled by rule names
- np^+ indicates a tree with root np^- needs to be substituted here
- inversely, s^- indicates we can subtitute this tree for an s^+
- h indicates the position of the lexical anchor (hypothesis)

Proof Nets for AB

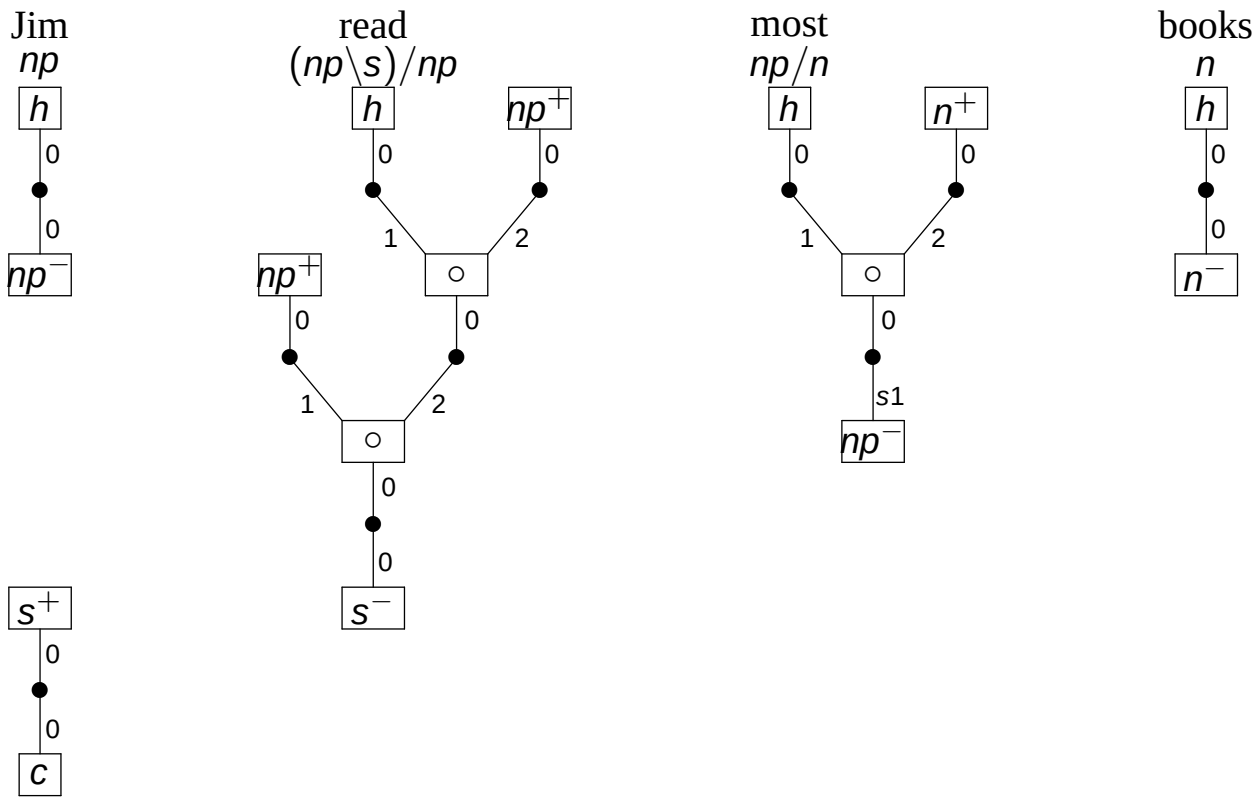
The Anatomy of a Lexical Entry



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

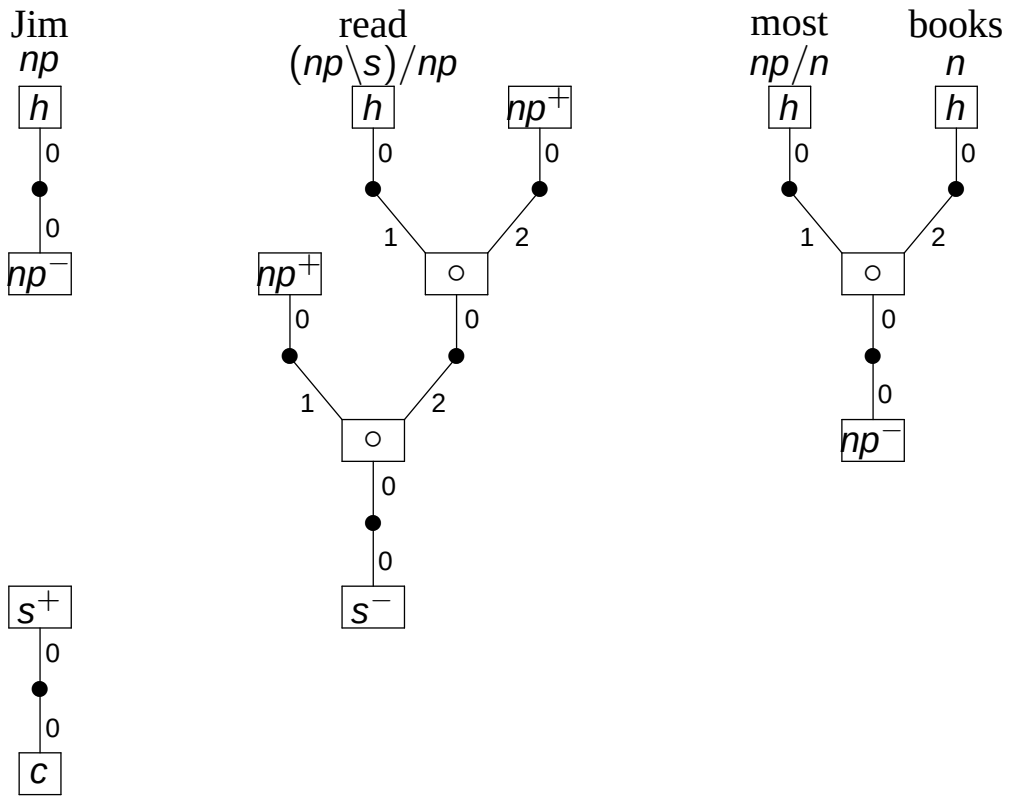
Proof Nets for AB

Example



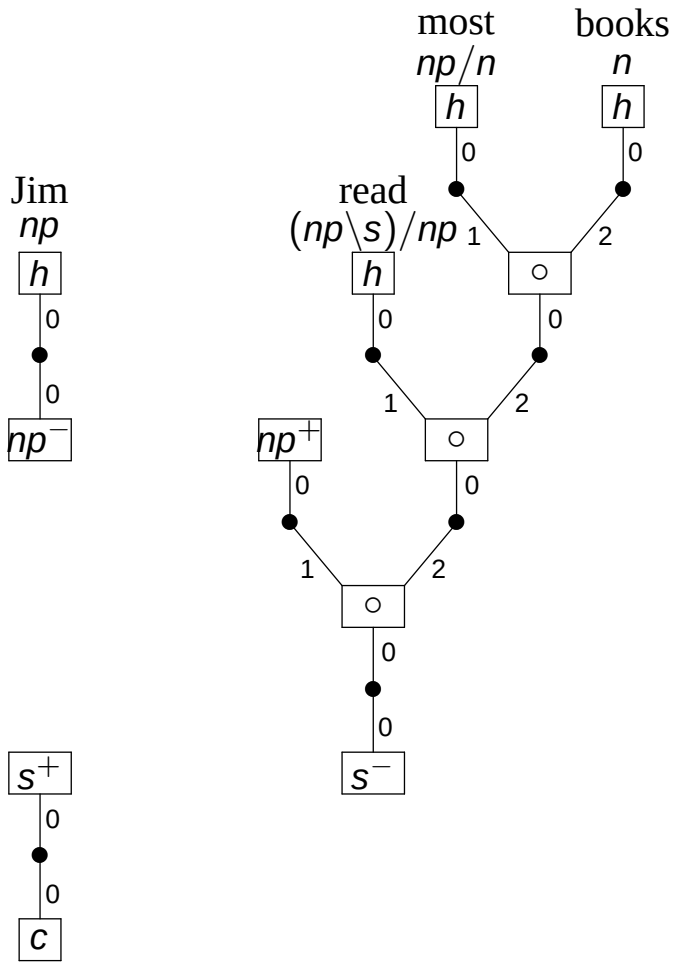
Proof Nets for AB

Example



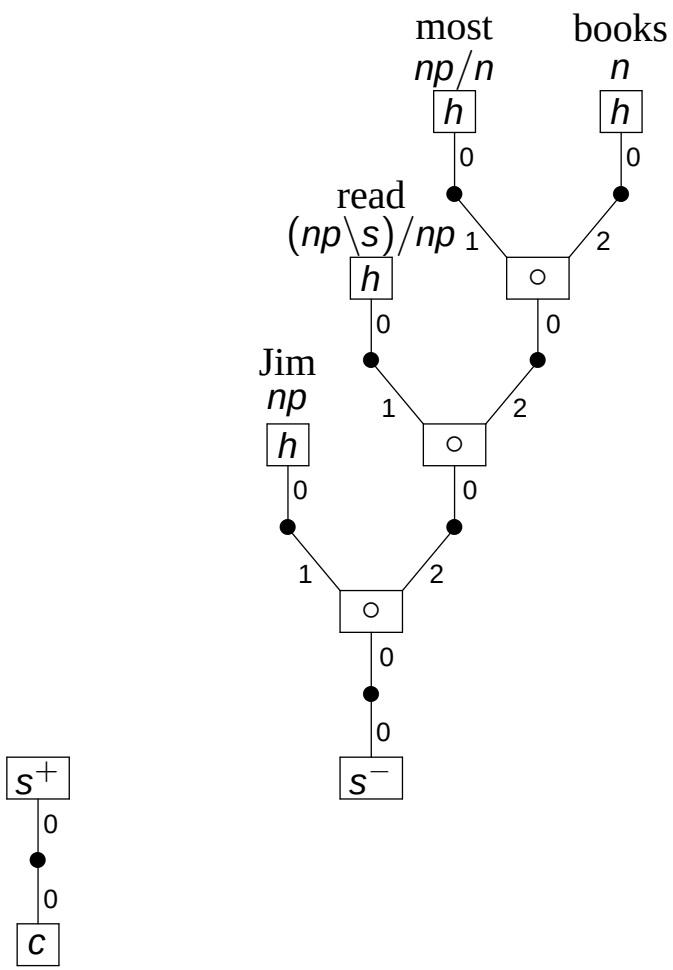
Proof Nets for AB

Example



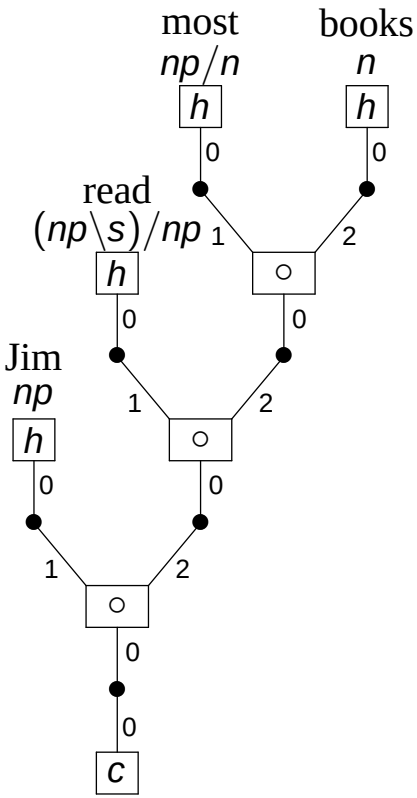
Proof Nets for AB

Example



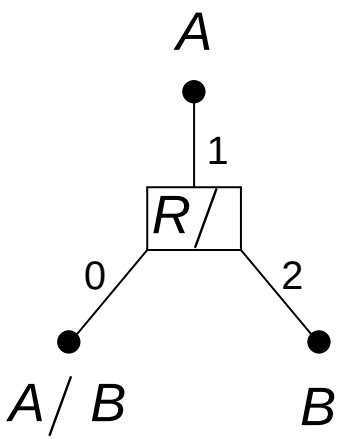
Proof Nets for AB

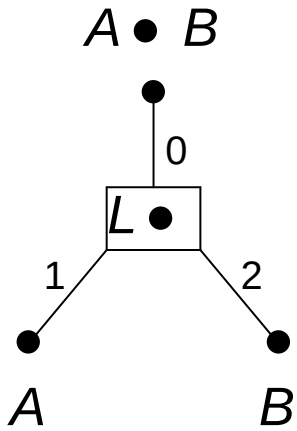
Example

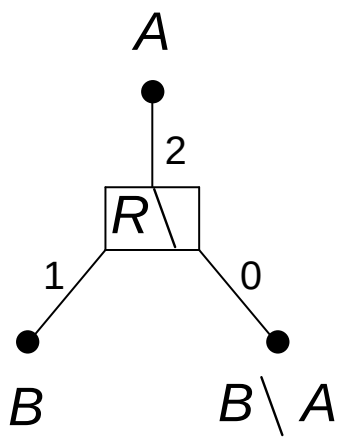


Multimodal Proof Nets

New Links

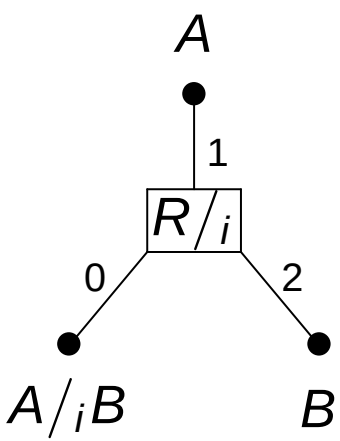


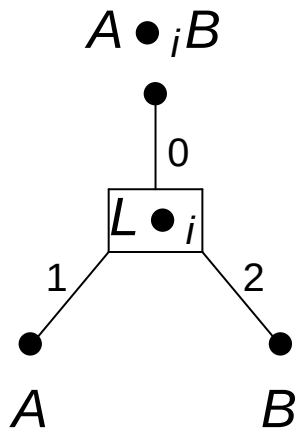


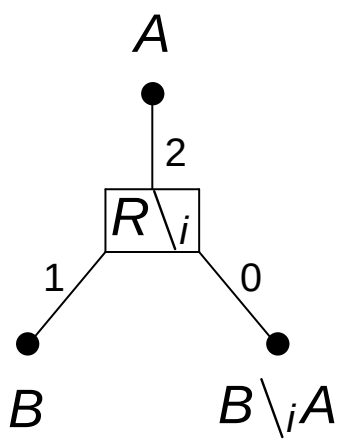


Multimodal Proof Nets

New Links — And Modes

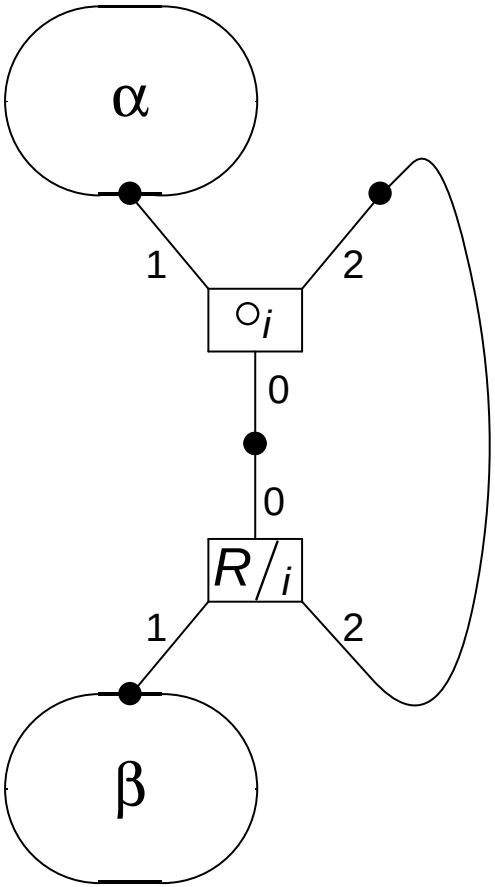


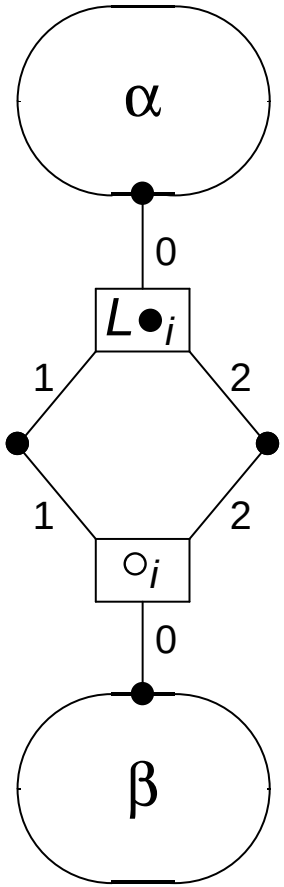


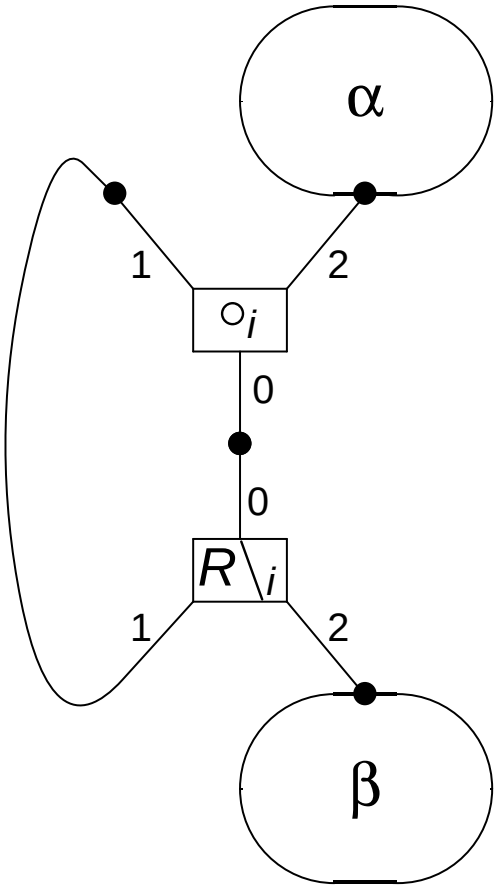


Multimodal Proof Nets

Contractions

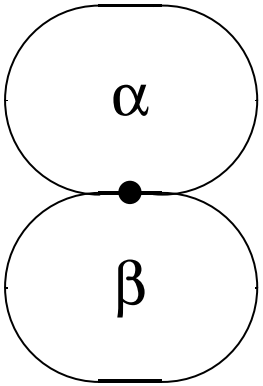


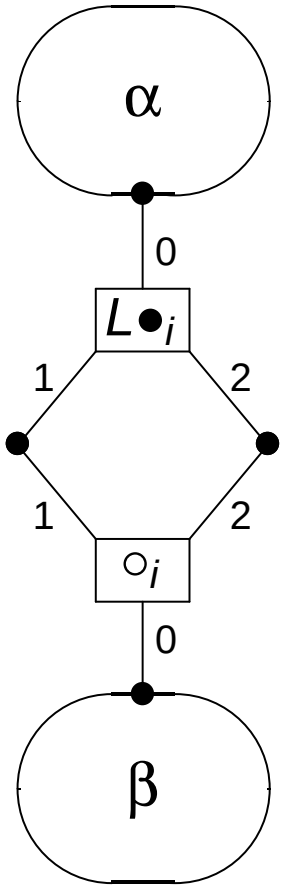


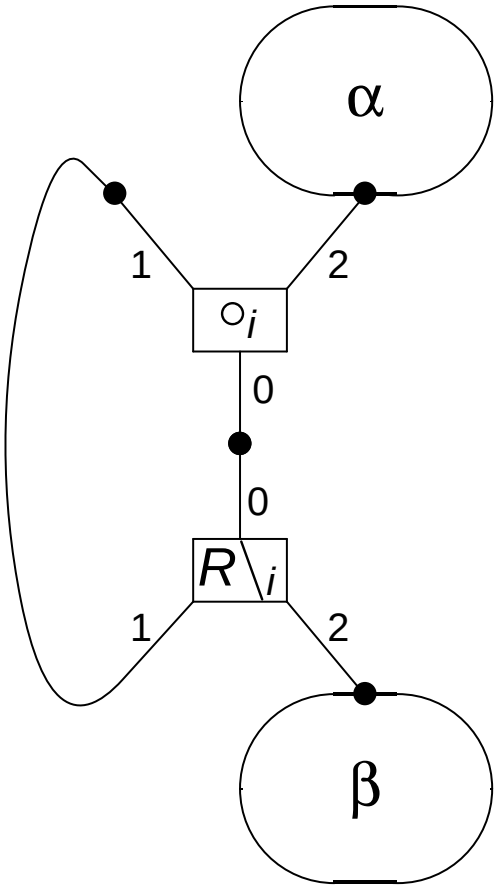


Multimodal Proof Nets

Contractions

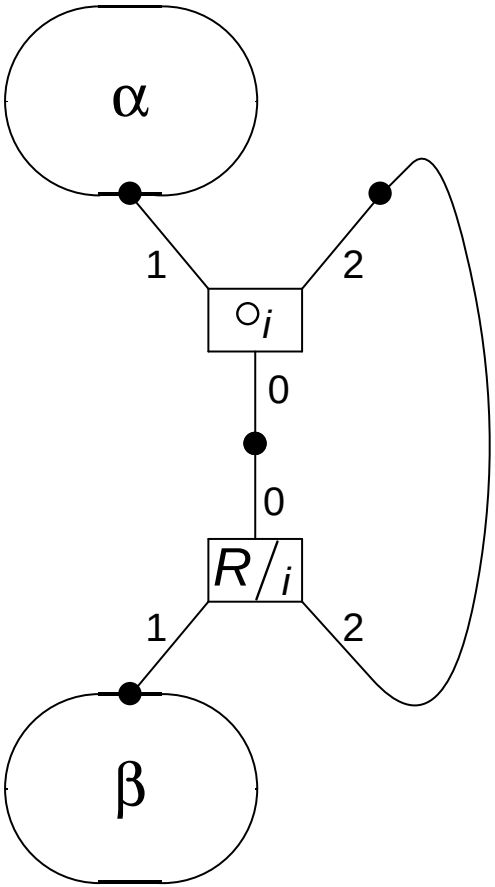


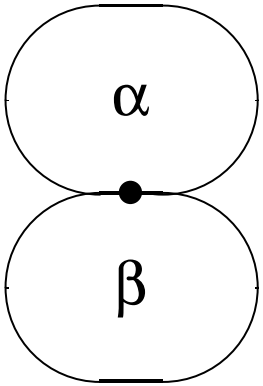


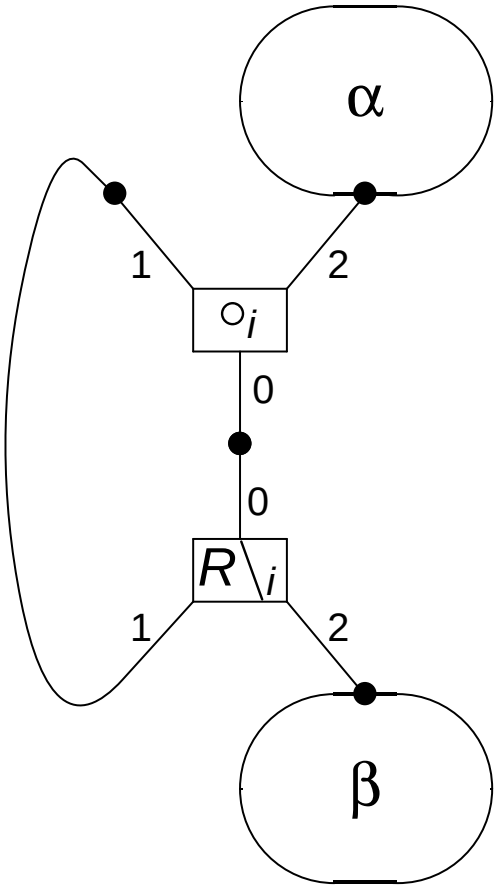


Multimodal Proof Nets

Contractions

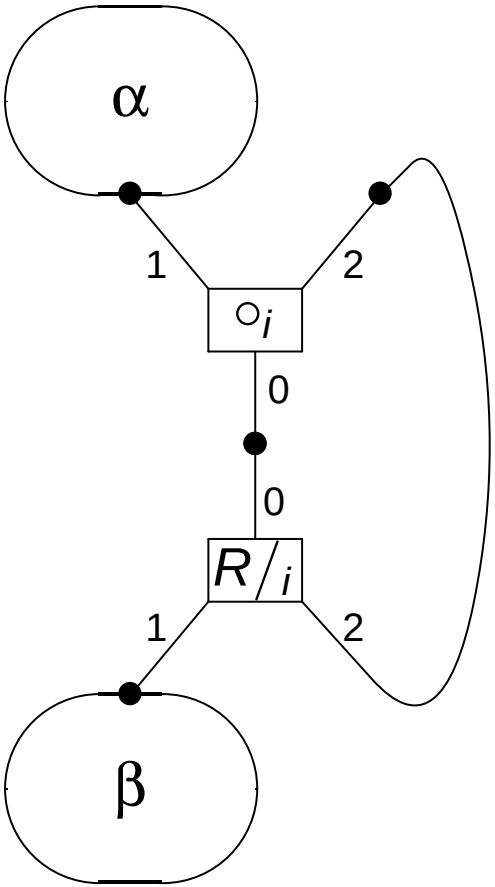


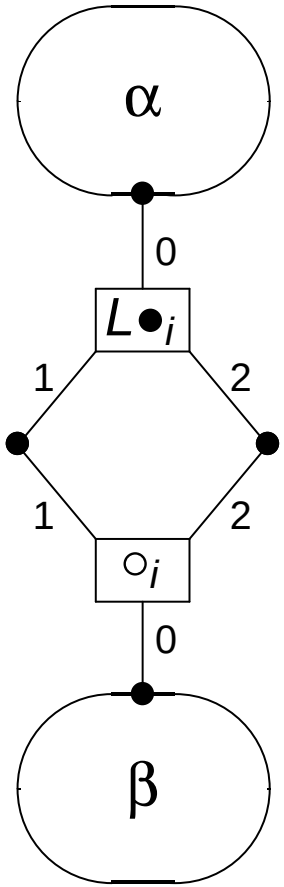


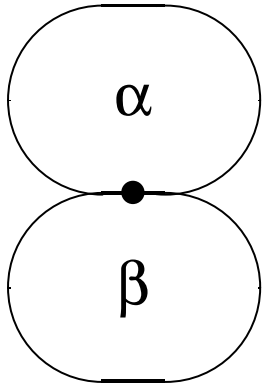


Multimodal Proof Nets

Contractions

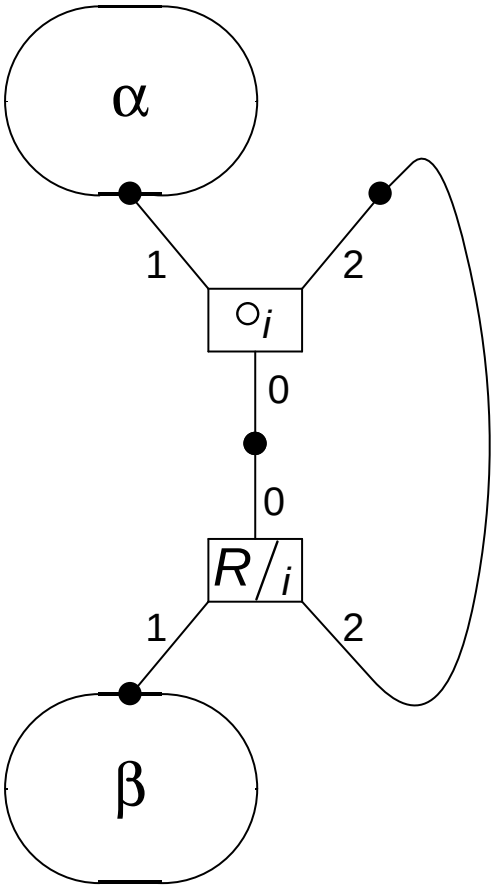






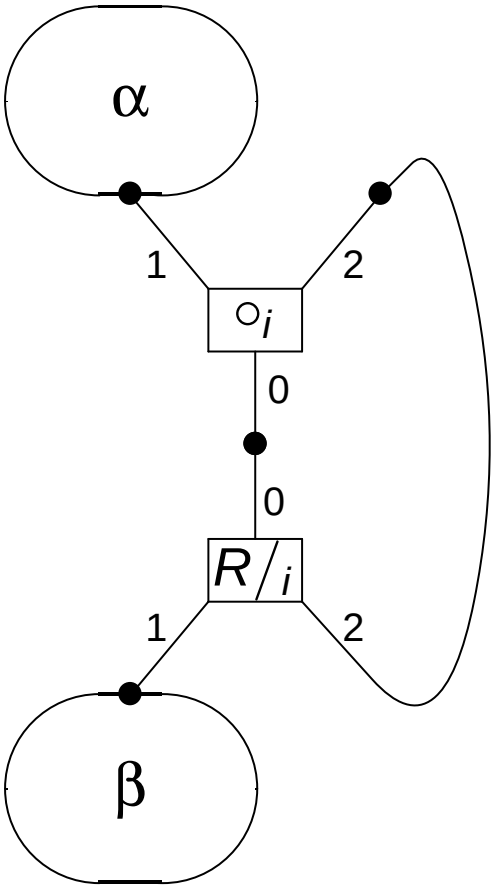
Multimodal Proof Nets

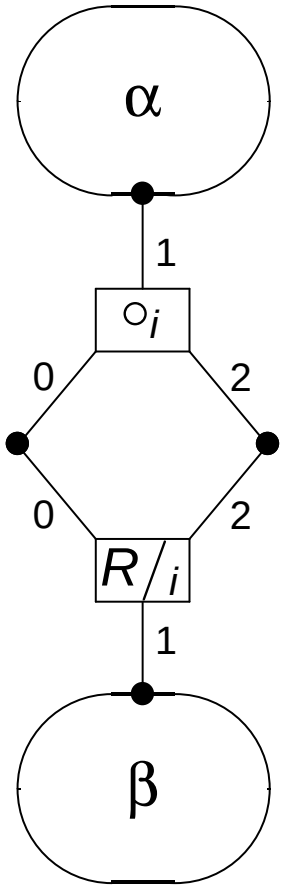
Contractions — Rotation



Multimodal Proof Nets

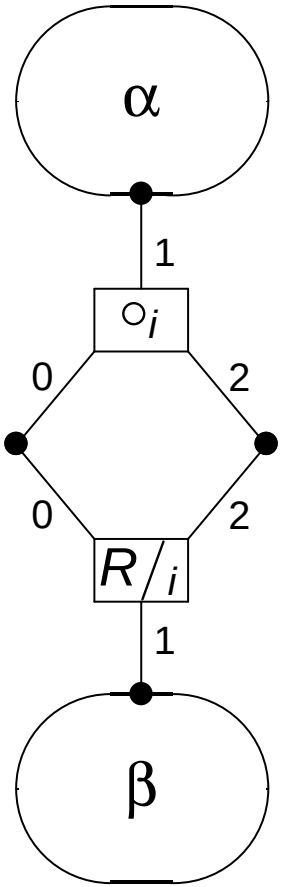
Contractions — Rotation

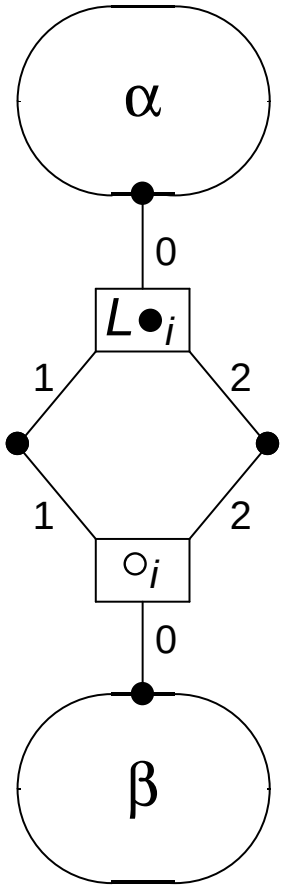


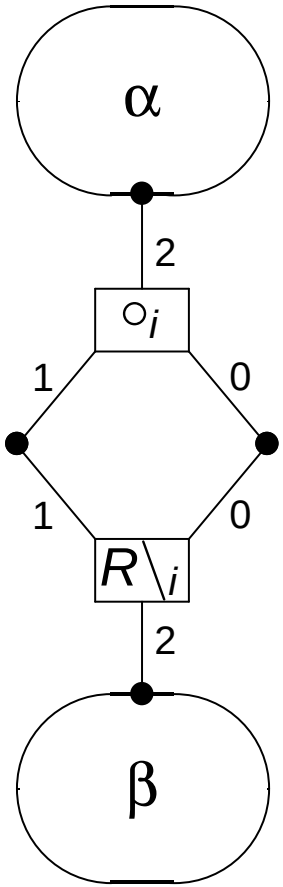


Multimodal Proof Nets

Contractions — Rotated

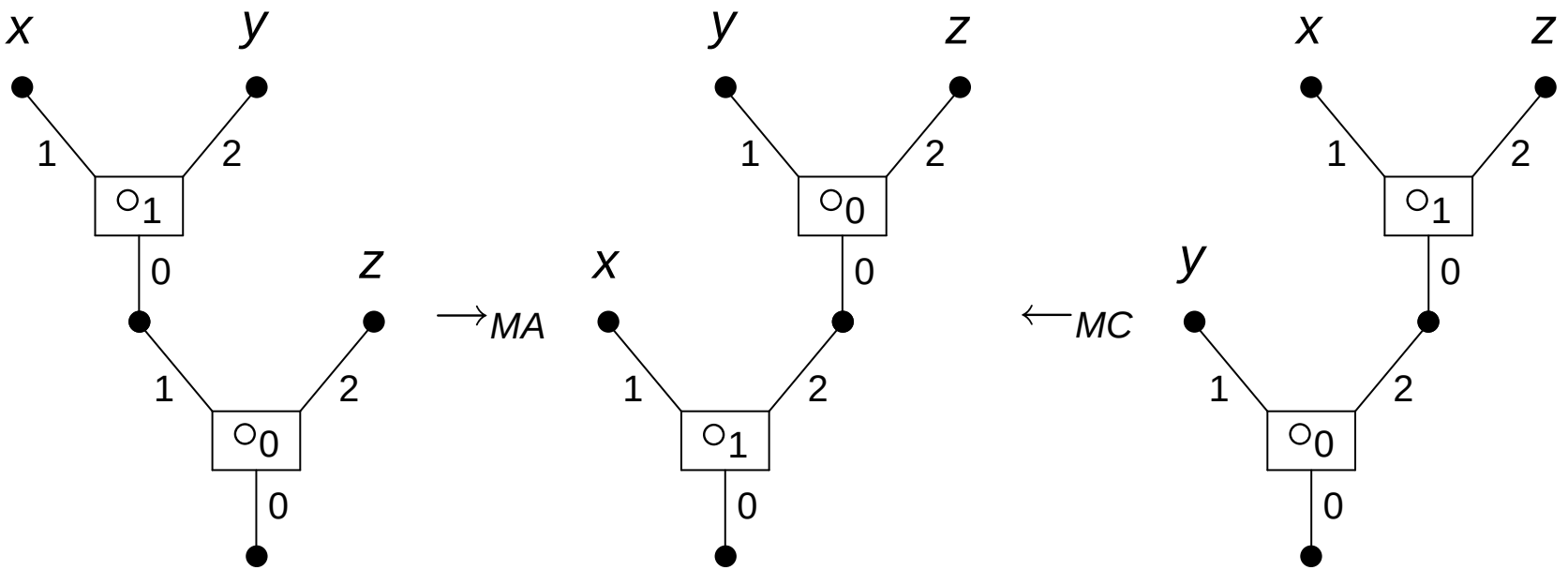






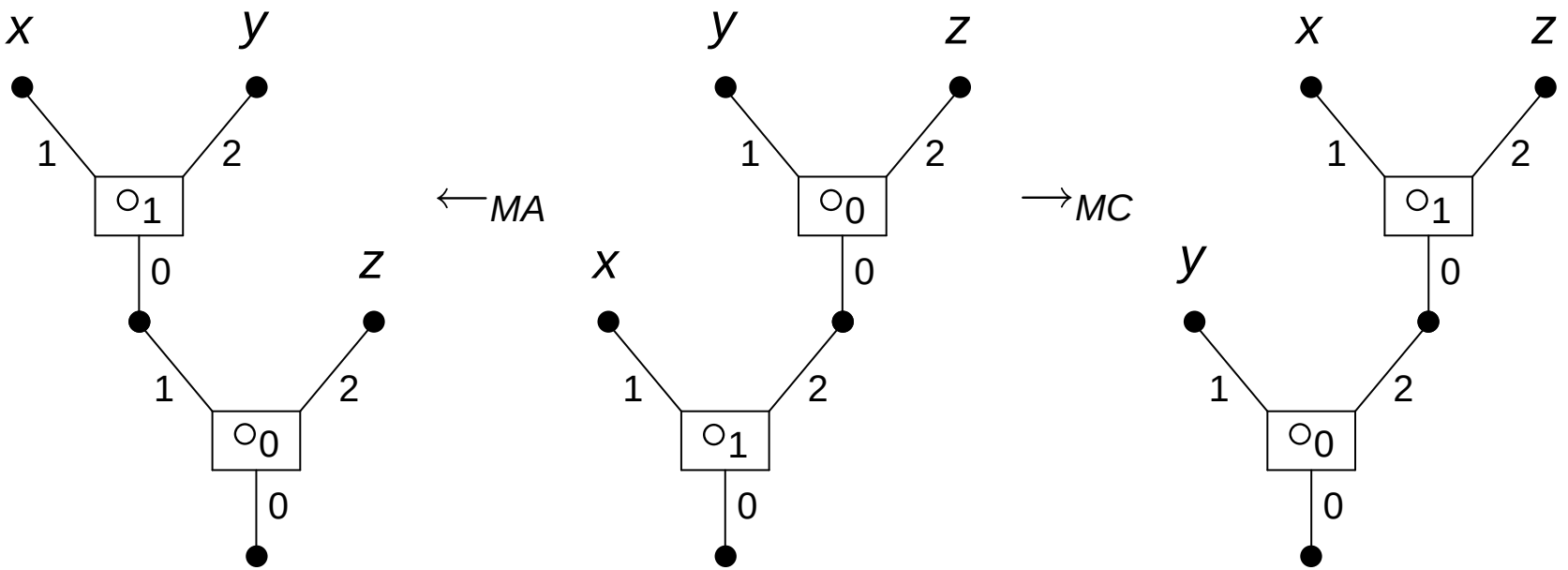
Structural Rules

Mixed Associativity and Commutativity— Extraction



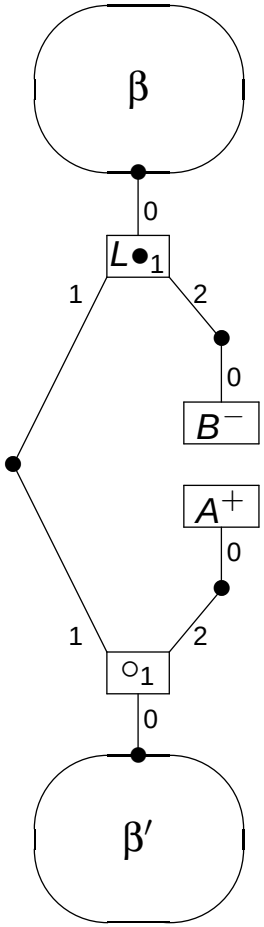
Structural Rules

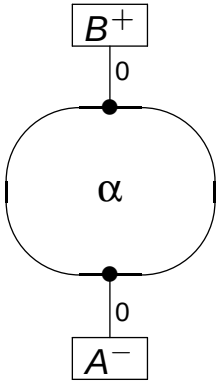
Mixed Associativity and Commutativity — Infixation



Simulating Adjunction

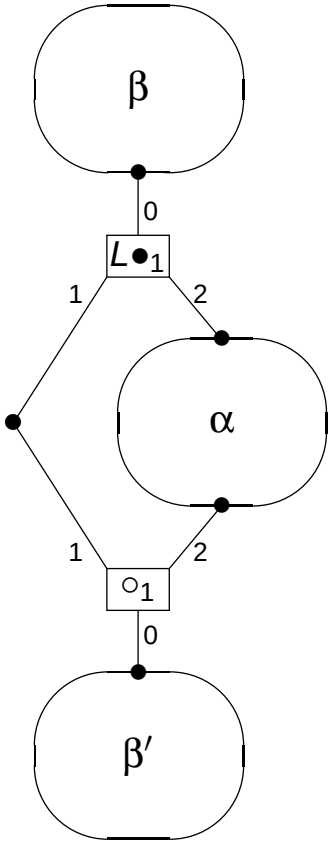
Initial Configuration of Adjunction Point





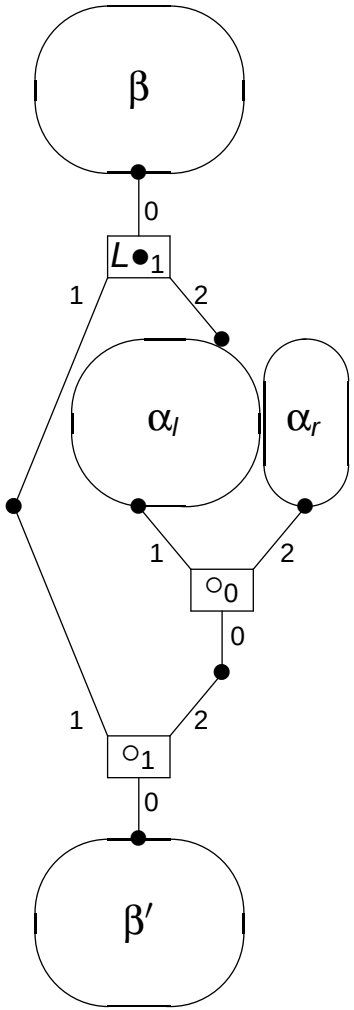
Simulating Adjunction

Axioms



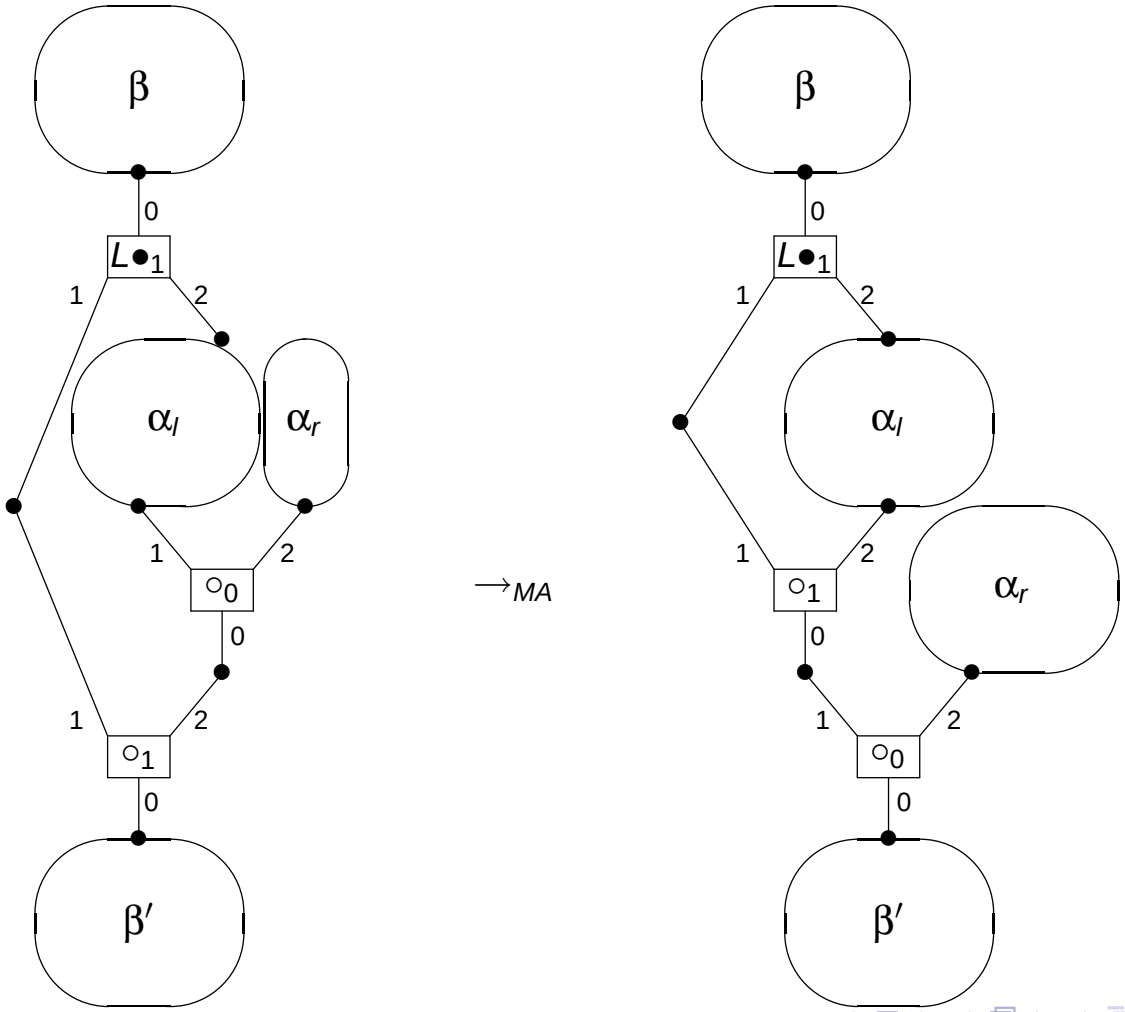
Simulating Adjunction

Structural Rules — Left



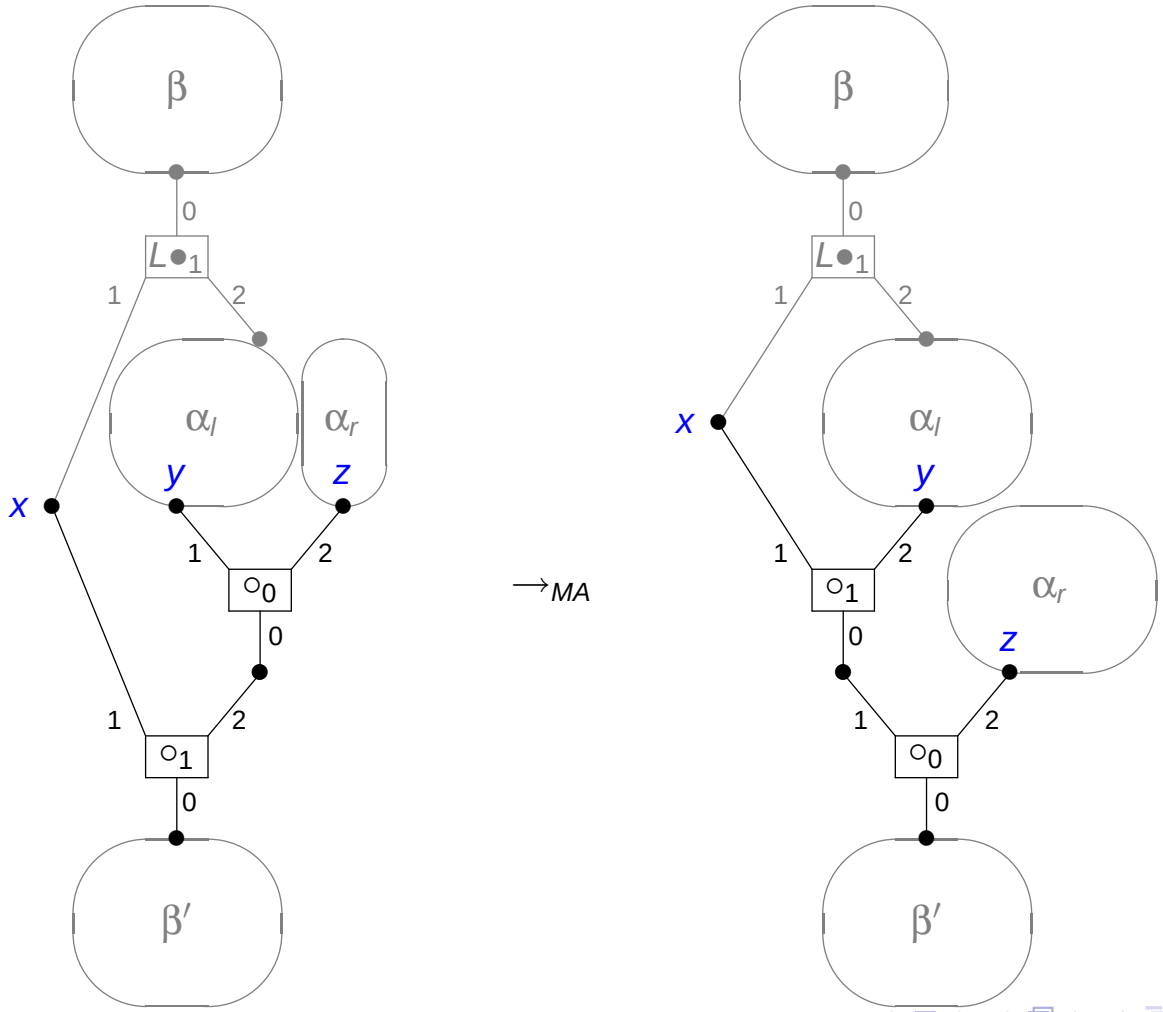
Simulating Adjunction

Structural Rules — Left



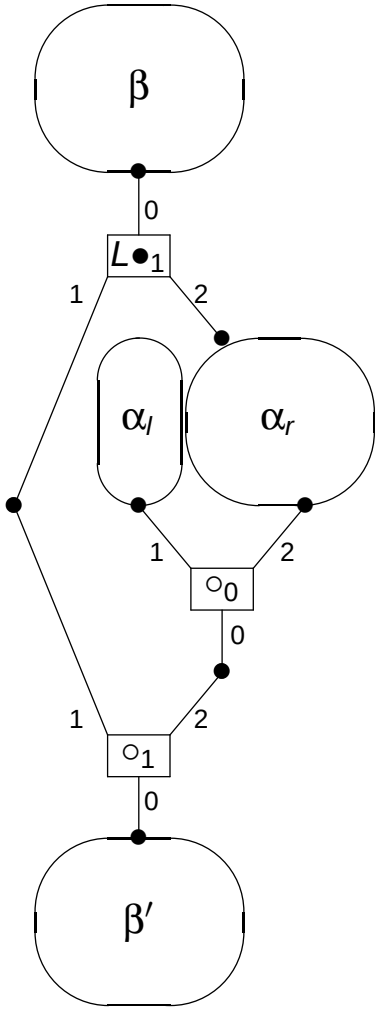
Simulating Adjunction

Structural Rules — Left



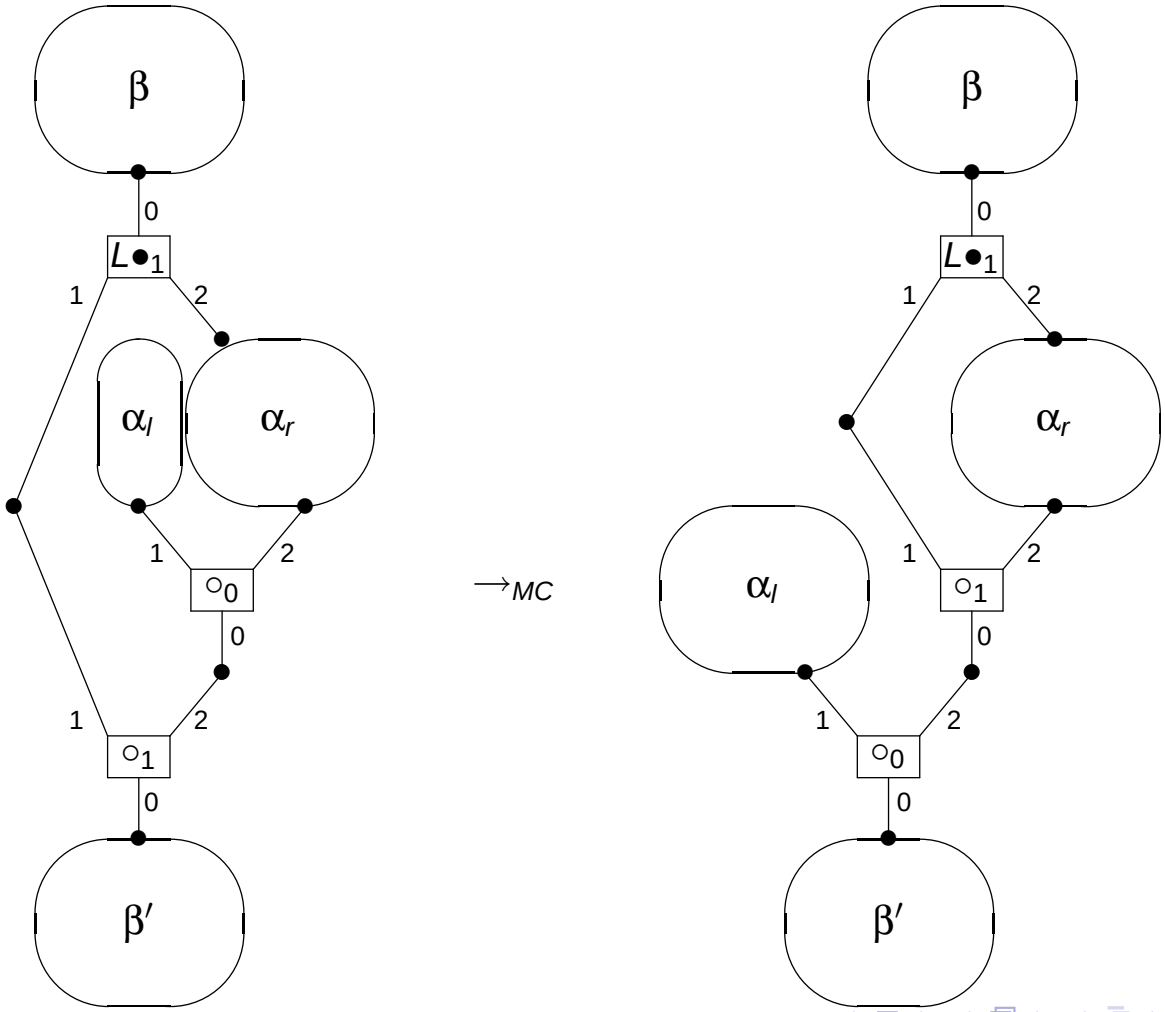
Simulating Adjunction

Structural Rules — Right



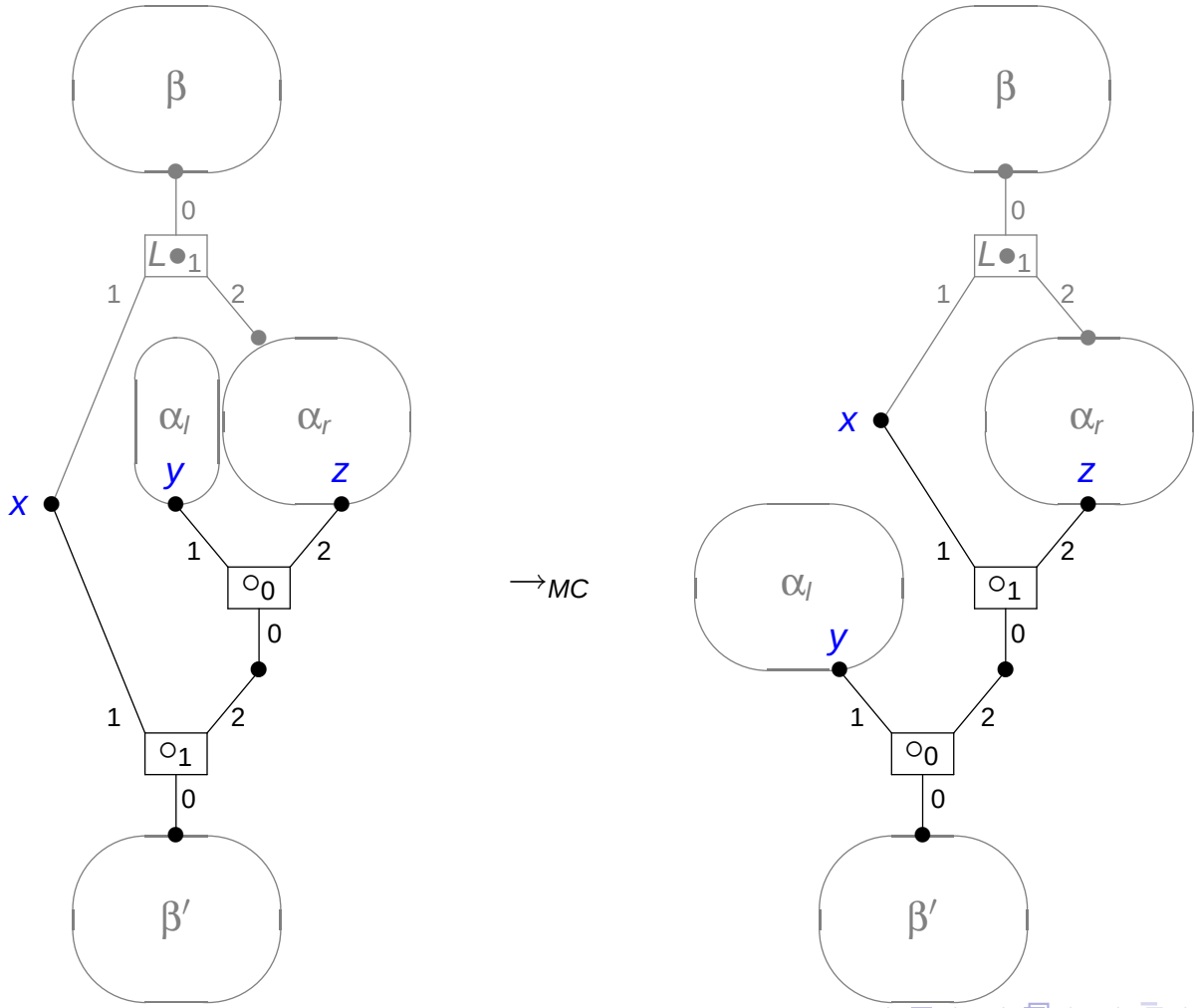
Simulating Adjunction

Structural Rules — Right



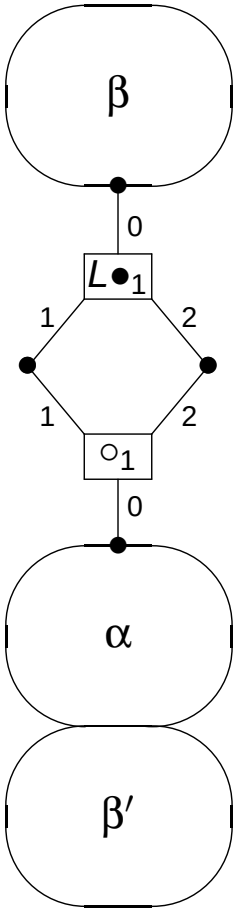
Simulating Adjunction

Structural Rules — Right



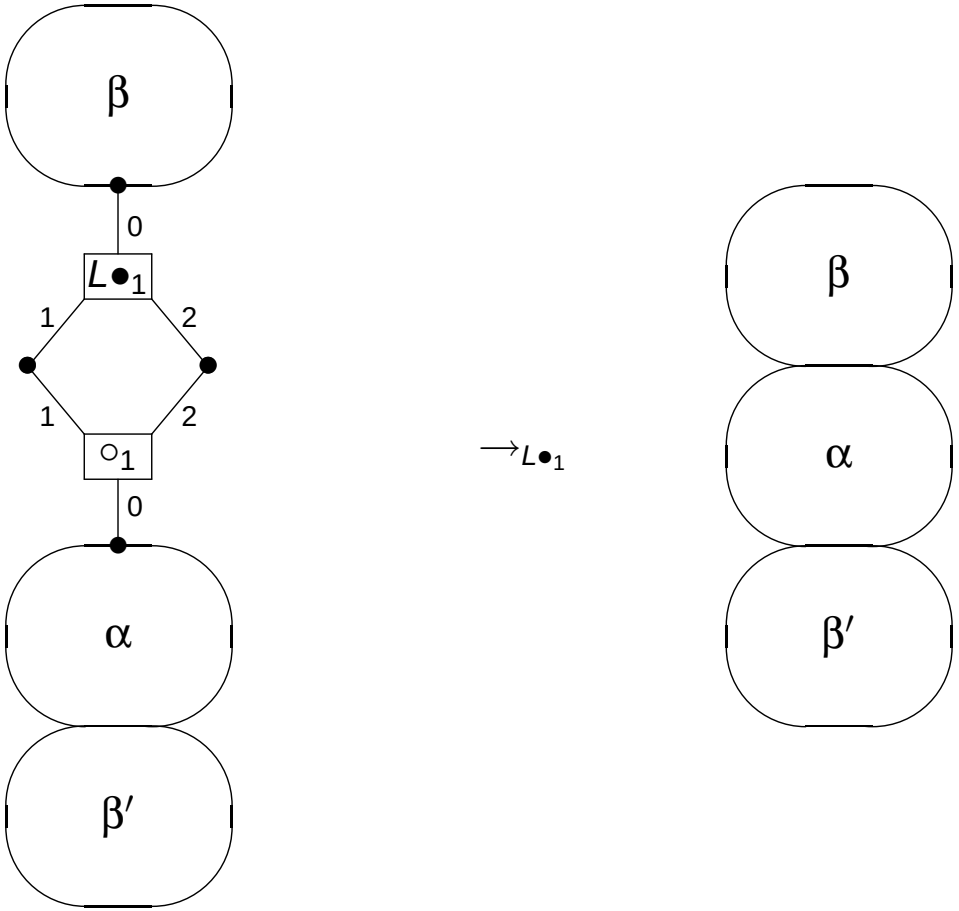
Simulating Adjunction

Structural Rules — Contraction and Final Result



Simulating Adjunction

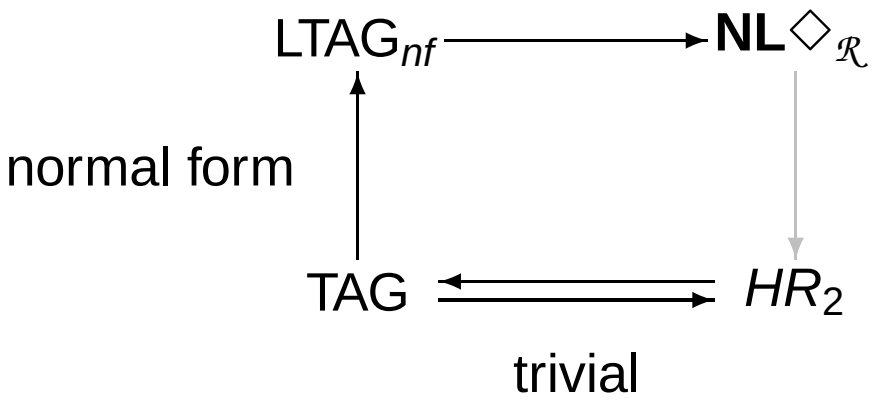
Structural Rules — Contraction and Final Result



Intermezzo

Relations so far

adjunction as contraction
and structural rules



HRG and Proof Nets

Nonterminal Symbols

- S (start),

HRG and Proof Nets

Nonterminal Symbols

- S (start),
- T_{00} (tree, cut),
- T_{01} (tree, flow down),
- T_{10} (tree, flow up),
- T_{11} (tree, axiom),

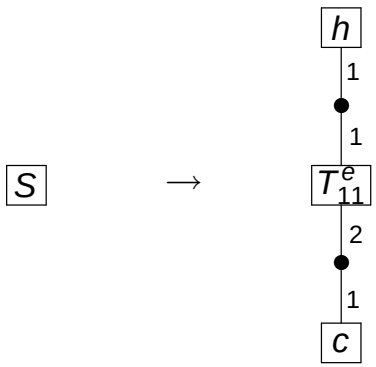
HRG and Proof Nets

Nonterminal Symbols

- S (start),
- T_{00} (tree, cut),
- T_{01} (tree, flow down),
- T_{10} (tree, flow up),
- T_{11} (tree, axiom),
- V_{00} (vertex, cut),
- V_{01} (vertex, flow down),
- V_{10} (vertex, flow up),
- V_{11} (vertex, axiom).

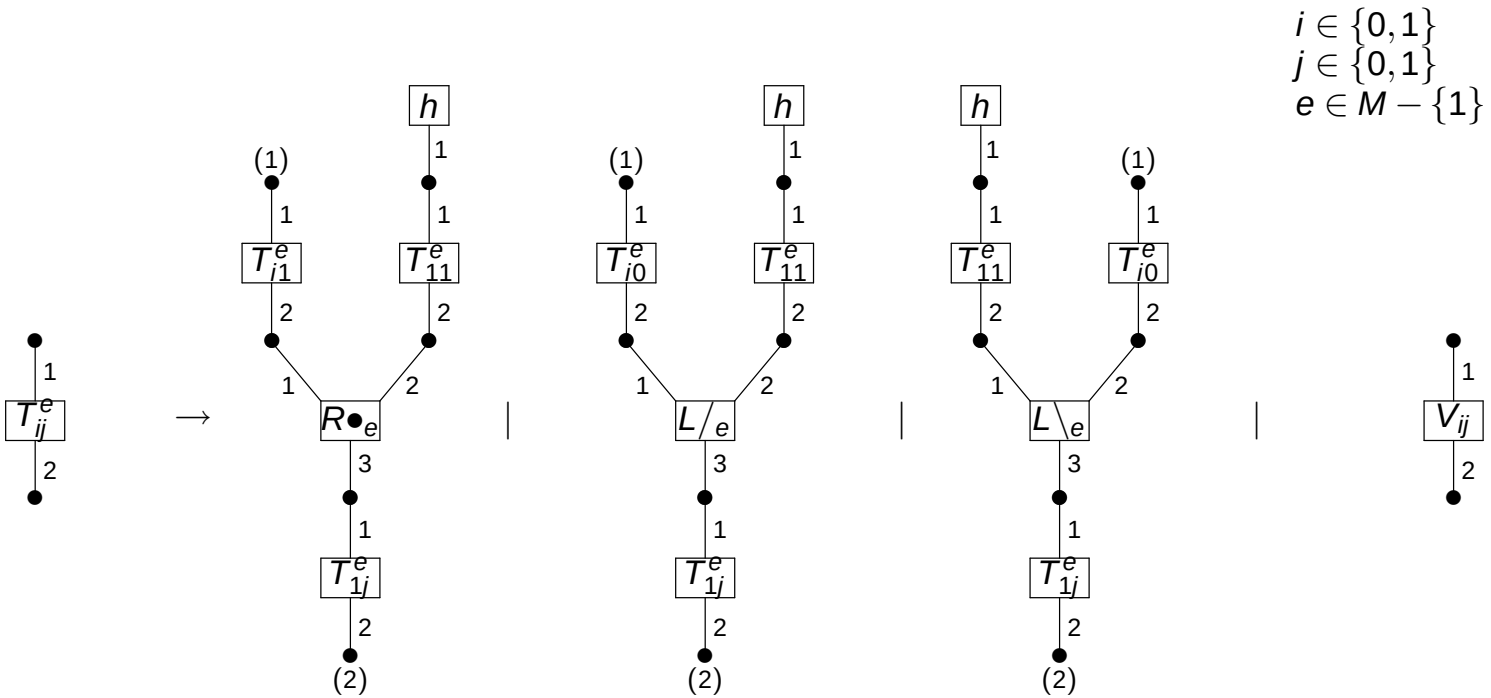
HRG and Proof Nets

Initial Axiom



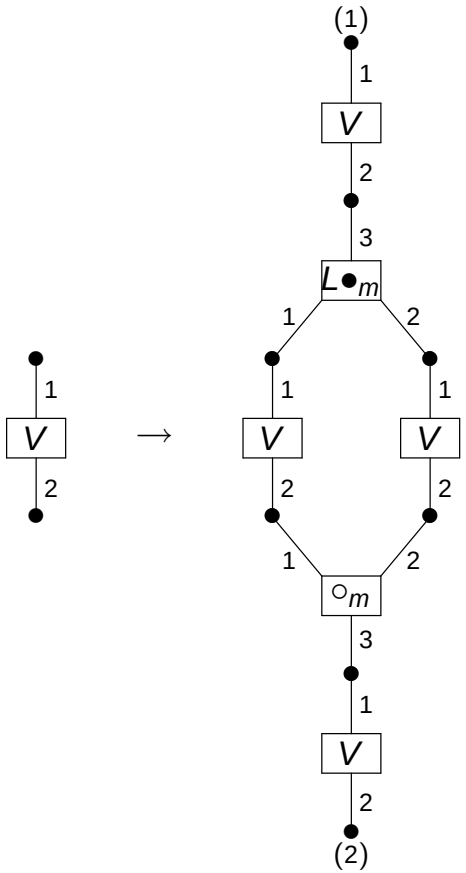
HRG and Proof Nets

Tensor Trees



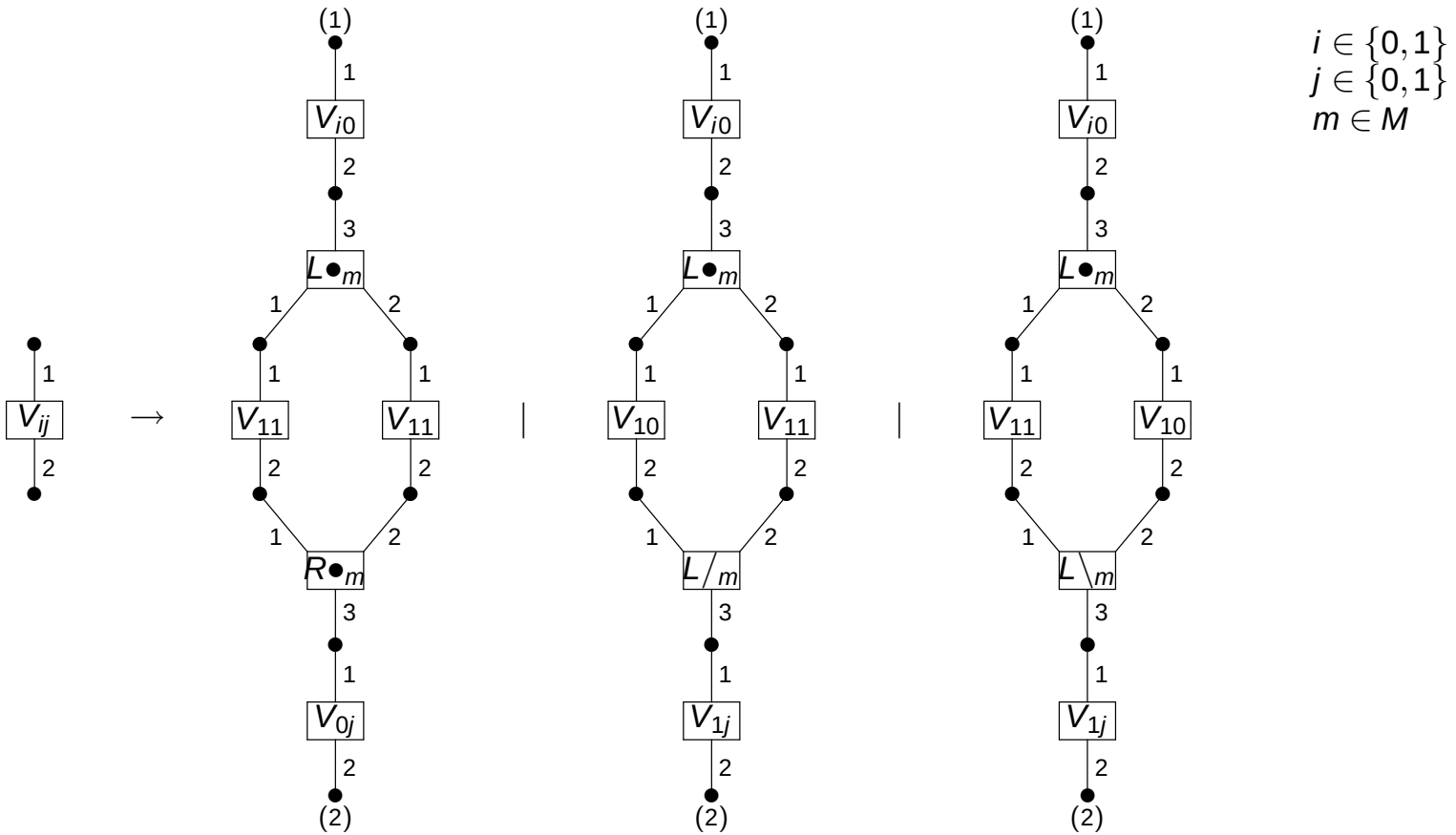
HRG and Proof Nets

Contractions: L• — Simplified



HRG and Proof Nets

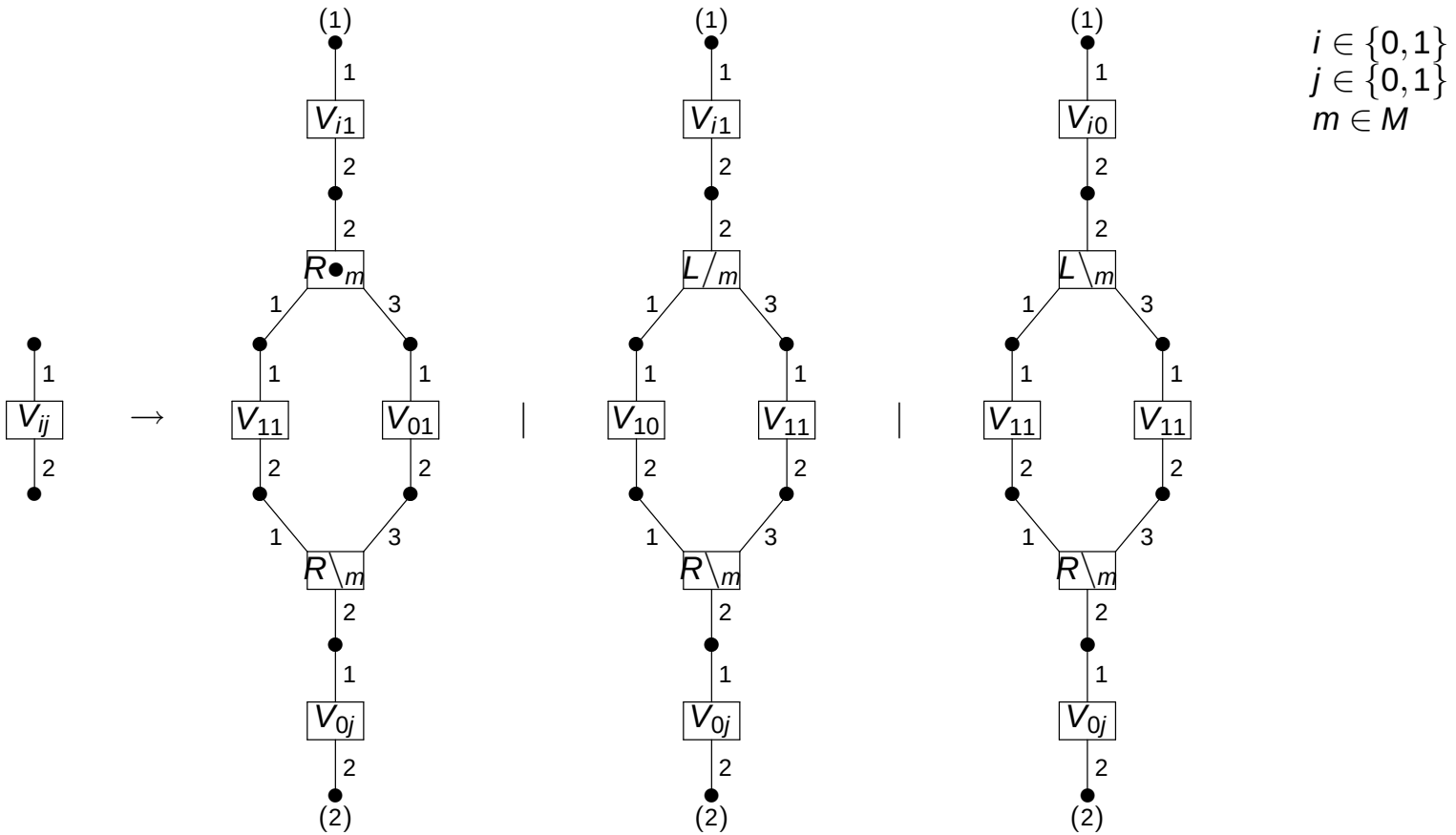
Contractions: L•



$i \in \{0,1\}$
$j \in \{0,1\}$
$m \in M$

HRG and Proof Nets

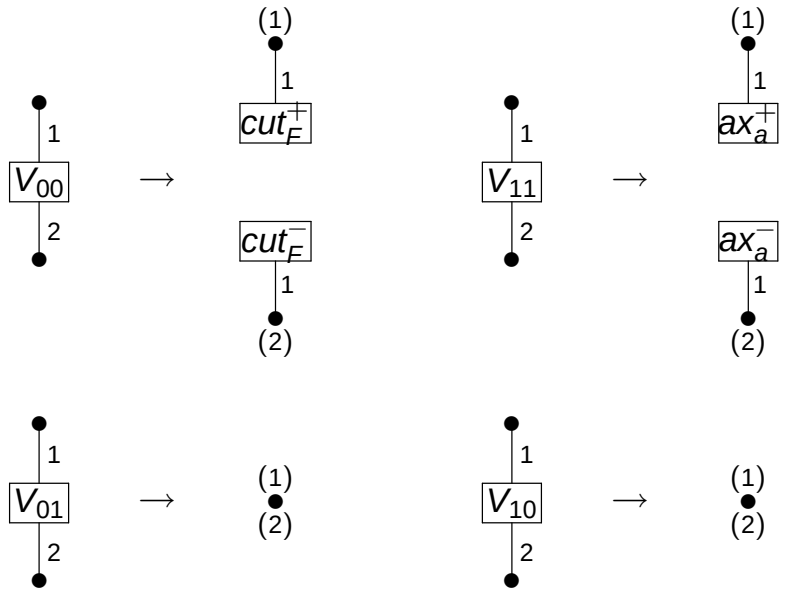
Contractions: R\



$i \in \{0,1\}$
$j \in \{0,1\}$
$m \in M$

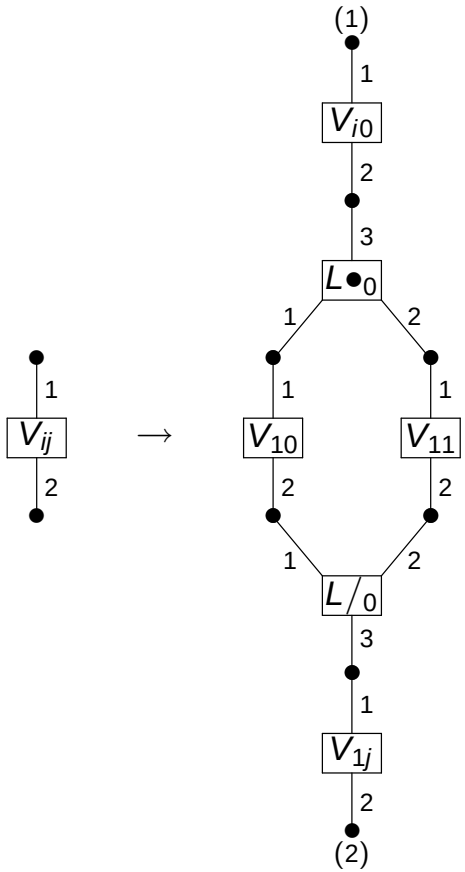
HRG and Proof Nets

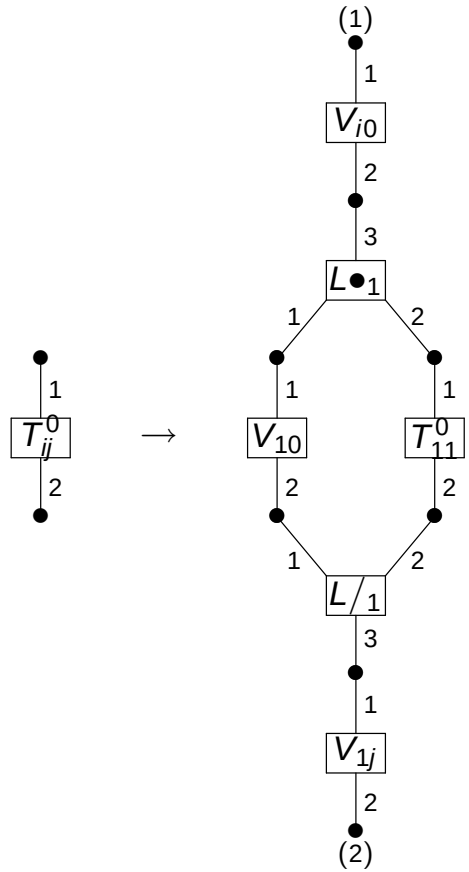
Cut, Flow, Axiom



HRG and Proof Nets

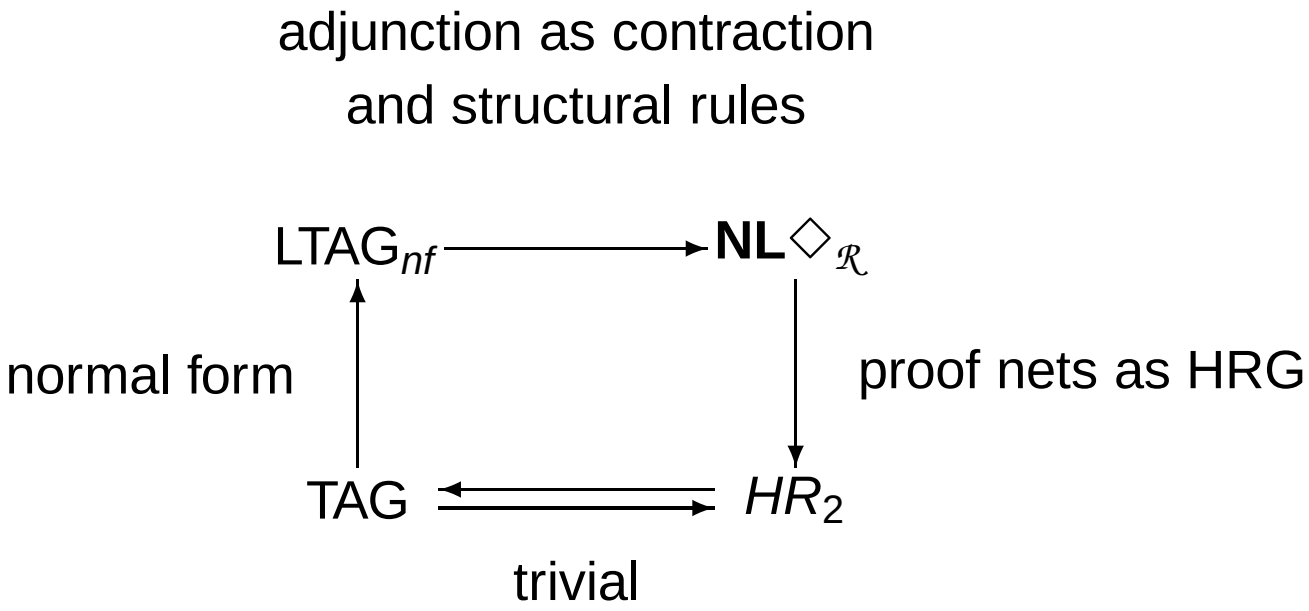
Structural Rules





Final Intermezzo

Relations between formalisms



Conclusions...

- Hyperedge replacement grammars, tree adjoining grammars and proof nets for **NL**$\diamond$ (with mixed associativity and mixed commutativity) all generate the same class of mildly context-sensitive string languages.
- As a consequence, this restricted class of categorial grammars is polynomially parseable.
- Thanks to the simplicity of their basic operations, hyperedge replacement grammars and tree adjoining grammars establishing these correspondences has been fairly easy.

Conclusions...

And Future Work!

- Enforce the links between the different formalisms. For parsing, *apex grammars* and graph algorithms for graphs of *bounded treewidth* seem to offer tantalizing possibilities.
- More classes of structural rules seem to permit a treatment by hyperedge replacement grammars. Identify as many as possible.
- 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

Bauderon, 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.

Joshi, A. & Kulick, S. (1997), 'Partial proof trees as building blocks for a categorial grammar', *Linguistics and Philosophy* **20**(6), 637–667.

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

Moortgat, M. & Oehrle, 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.

Vijay-Shanker, K. & Weir, D. (1994), 'The equivalence of four extensions of context free grammars', *Mathematical Systems Theory* **27**(6), 511–546.

Weir, 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.