# Proof Nets For LG

Richard Moot

LaBRI (CNRS), INRIA Bordeaux & University of Bordeaux

Prelude, 16 January 2008

# Introduction

## Lambek-Grishin: Beyond the non-associative Lambek calculus

- The Lambek-Grishin calculus extends the Lambek calculus by reversing all arrows between derivations.
- Grishin (1983) proposes an number of interaction principles between the (standard) Lambek and the (inverse) Grishin connectives.
- Bernardi & Moortgat (2007) apply these connectives to compute quantifier scope — including scope possiblities unobtainable in the Lambek calculus — as well as their meaning recipes by means of continuations.

# Introduction

## Lambek-Grishin: Proof Nets and Applications

- We would like to have a non-redundant proof system for LG similar to the proof nets for linear logic.
- Adapting the proof nets for the multimodal Lambek calculus by using the symmetries between hypothesis and conclusions is fairly simple.
- I will finish by showing a new application of LG: tree adjoining grammars are translated into LG, showing that LG can handle phenomena beyond context-free grammars.

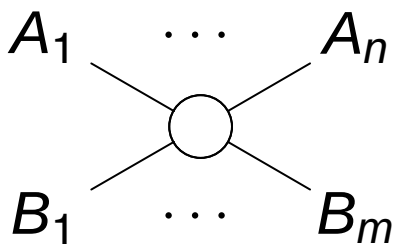
# Introduction
## Proof Nets for Multiplicative Linear Logic

- A *proof structure* is a set of links between formulas. A proof structure has only conclusions.
- From a proof structure we obtain a set of more general structures which are *correction graphs*. Correction graphs are obtained by erasing all formulas and making a binary choice of a single edge for every par 'linear or' link.
- A *proof net* is a proof structure for which all correction graphs are acyclic and connected, ie. they are (unrooted) trees.

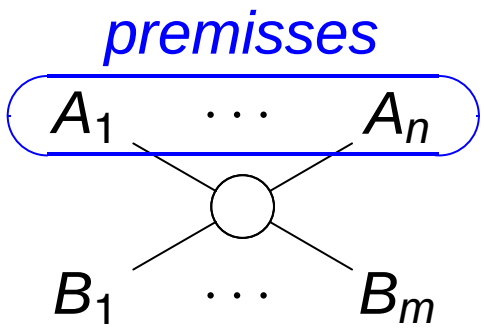
# Introduction
## Proof Nets for NL, LG

- A *proof structure* is a set of links between formulas. A proof structure has *hypotheses* in addition to conclusions.
- From a proof structure we obtain a more general structure which is an *abstract proof structure*. Abstract proof structures are obtained by erasing all formulas *on the internal nodes*.
- A *proof net* is a proof structure for which the abstract proof structure *contracts to a tree*. For NL this tree is a binary rooted tree, for LG it might not be.

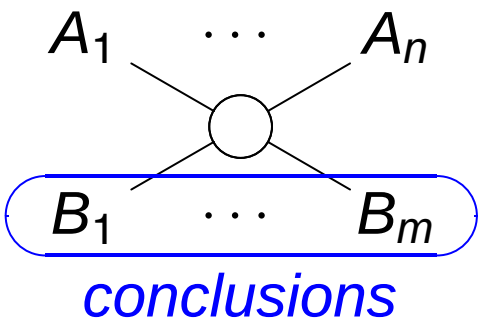
# Links
## The Anatomy of a Link



# Links
## The Anatomy of a Link

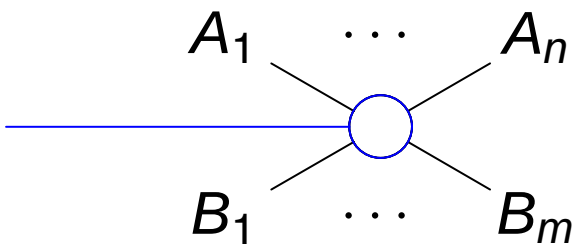


# Links
## The Anatomy of a Link



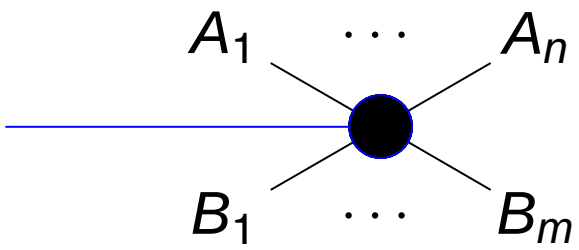
# Links
## The Anatomy of a Link

Link type: tensor

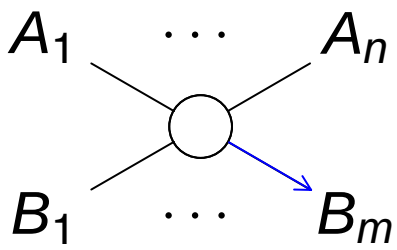


# Links
## The Anatomy of a Link

Link type: par

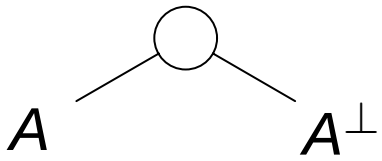


# Links
## The Anatomy of a Link

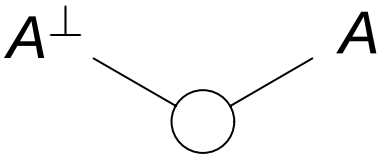



Main formula

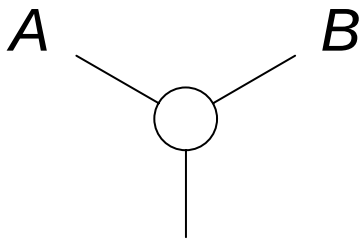
# Links
## Multiplicative Linear Logic




*Axiom*

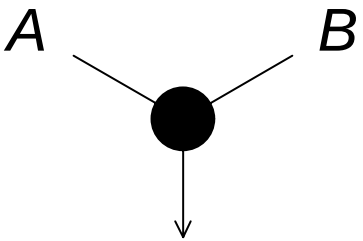



*Cut*




$A \otimes B$

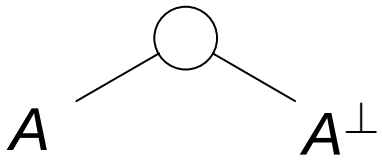
*Tensor*



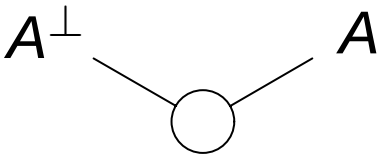

$A \parr B$

*Par*

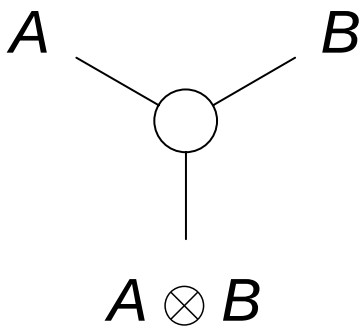
# Links
## Multiplicative Linear Logic



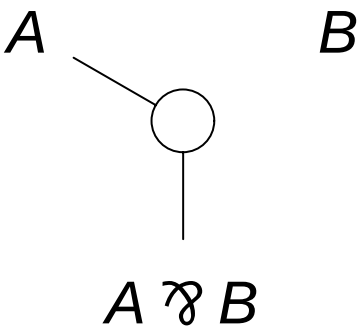

*Axiom*




*Cut*

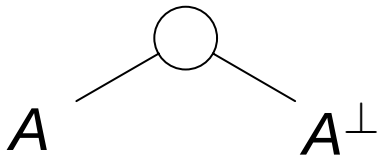



*Tensor*

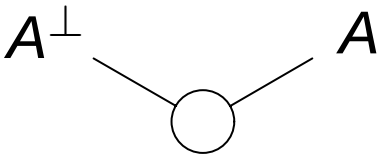



*Par* *(left switching)*

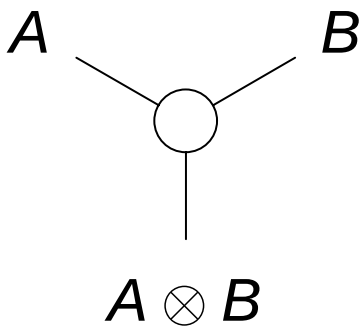
# Links
## Multiplicative Linear Logic



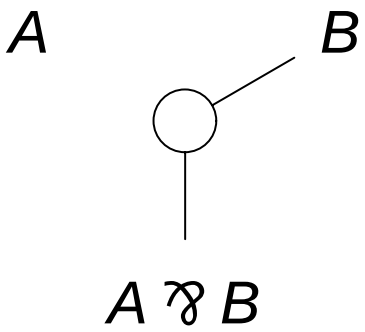

*Axiom*




*Cut*




*Tensor*

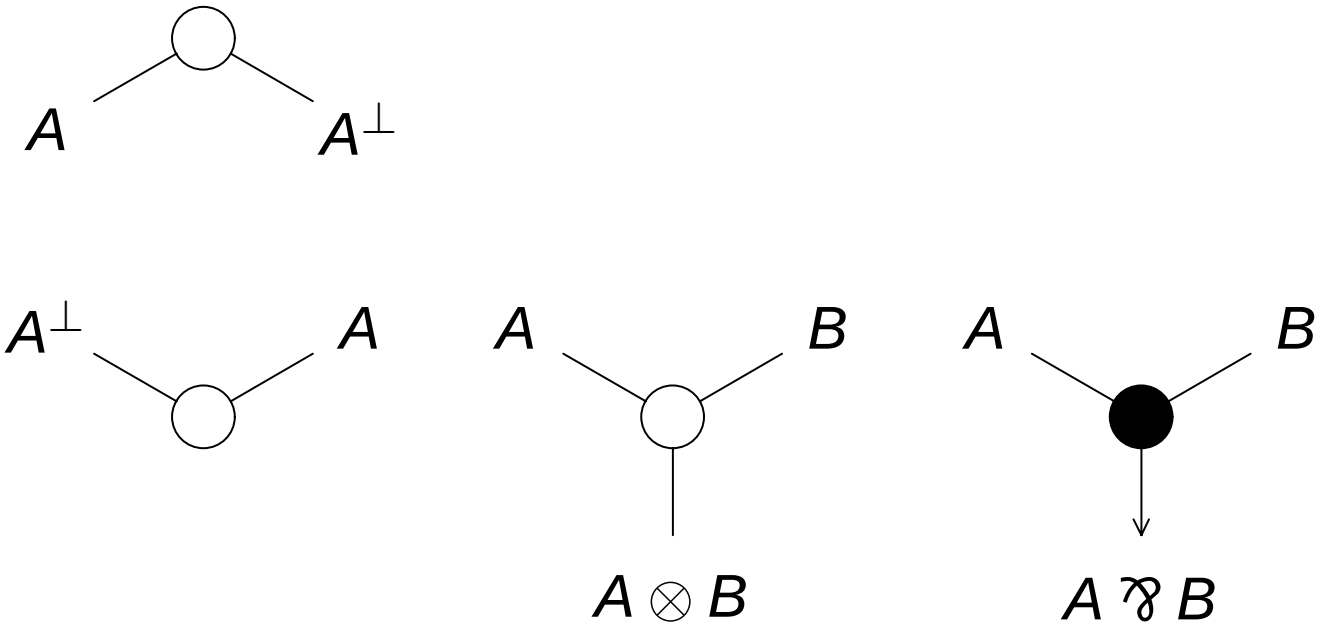



*Par* *(right switching)*

# Links
## Conclusions and Hypotheses

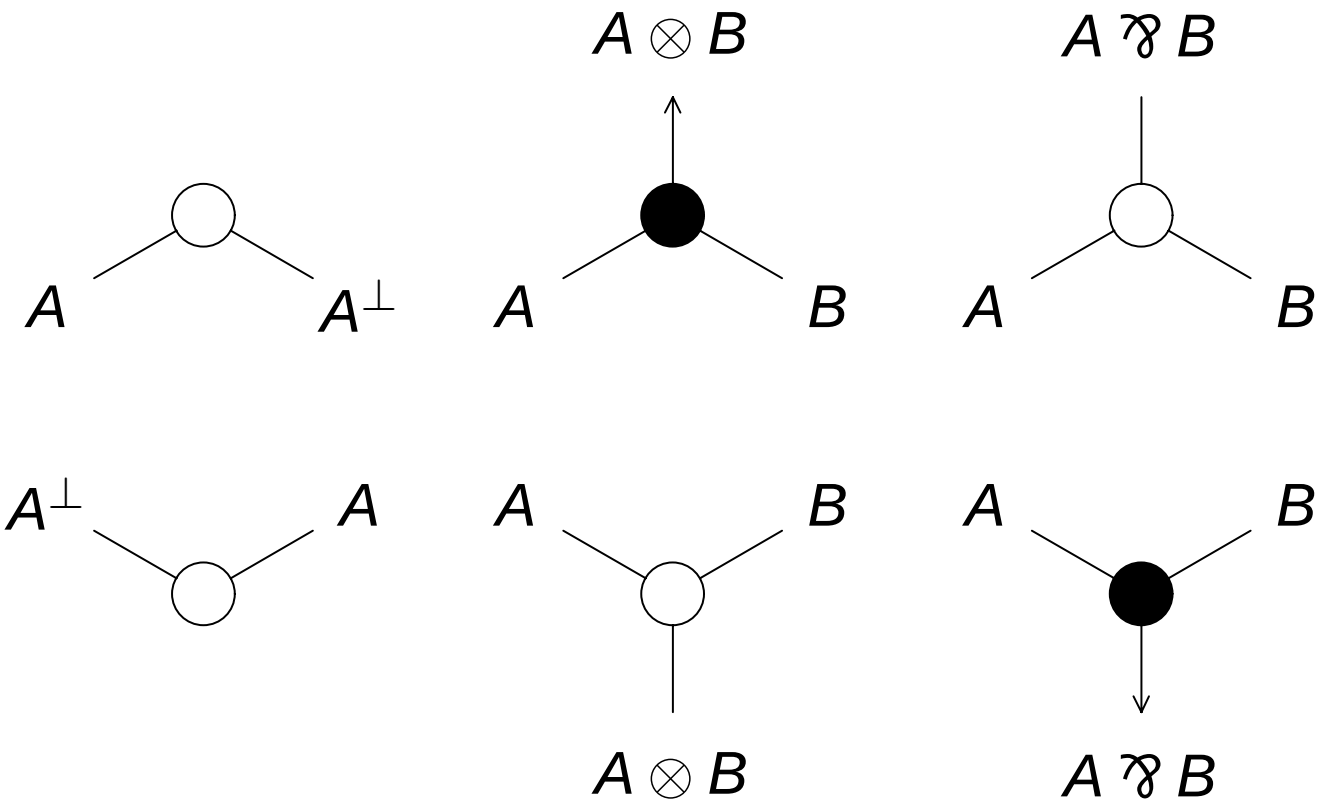
- The links for MLL as given before conflate the two notions of *conclusion* and *main formula*.
- Puite (1998) adds a symmetrical *hypothesis link* — or left link — for every conclusion link — or right link.
- This gives us proof nets (with hypotheses) for two-sided MLL.

# Links
## Two-Sided Multiplicative Linear Logic

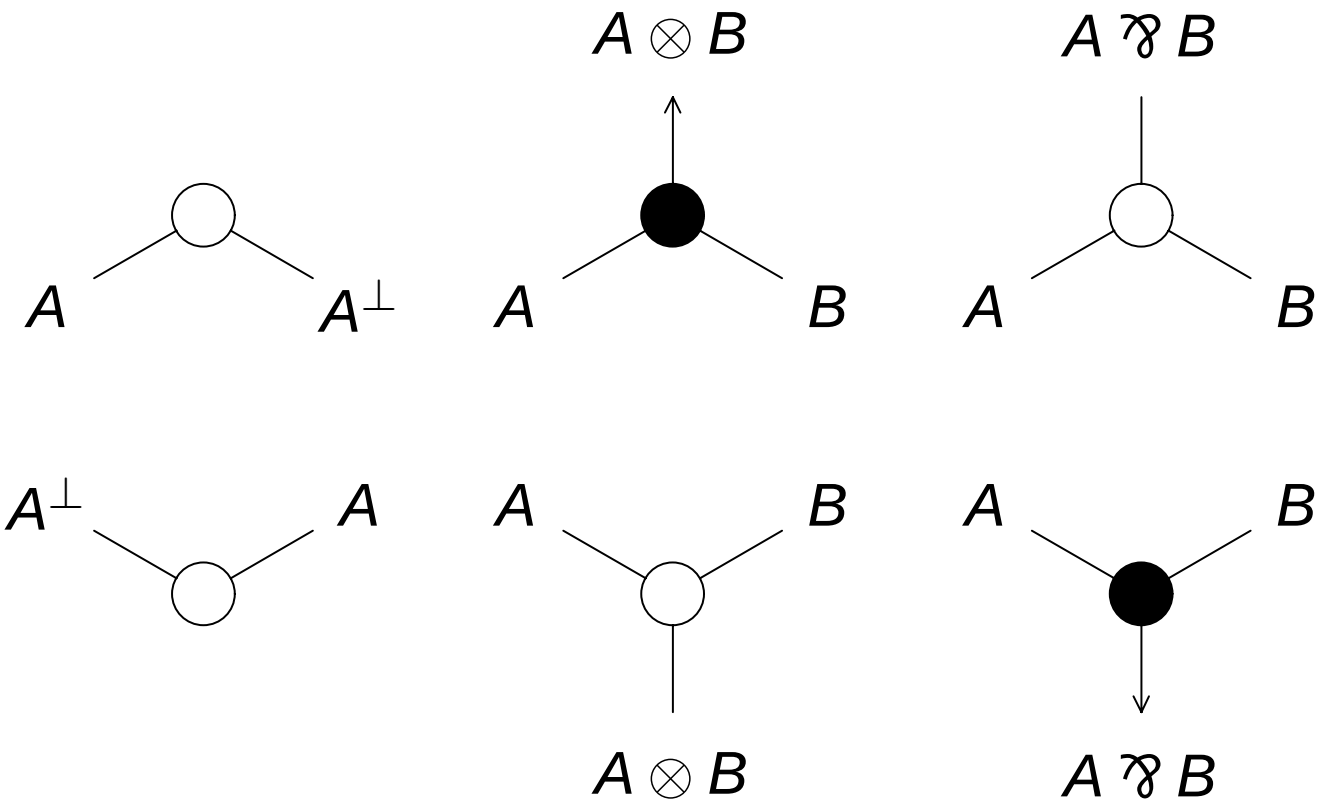


# Links

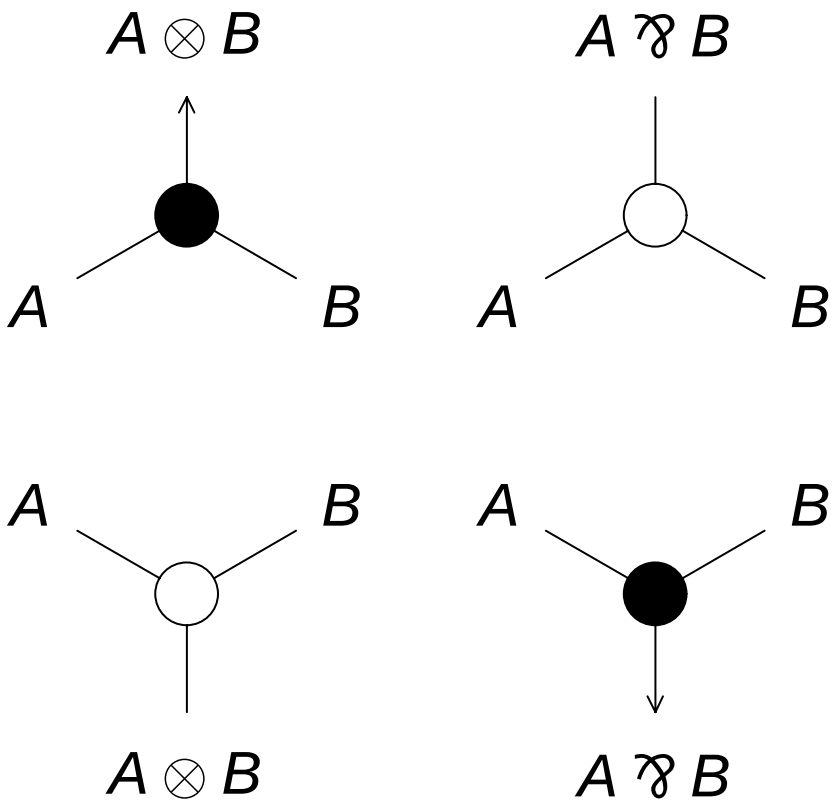
## Two-Sided Multiplicative Linear Logic



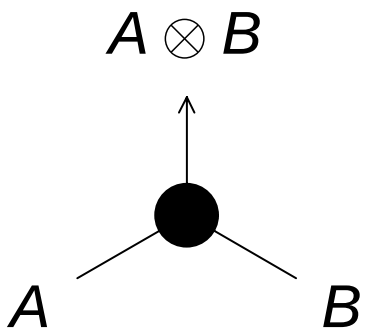
# Links
## From 2-Sided MLL to NL

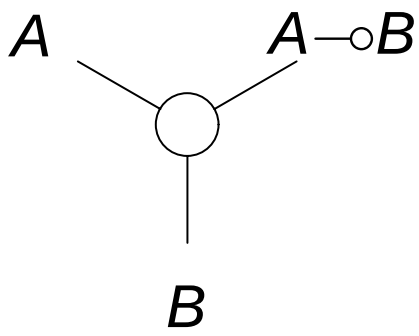


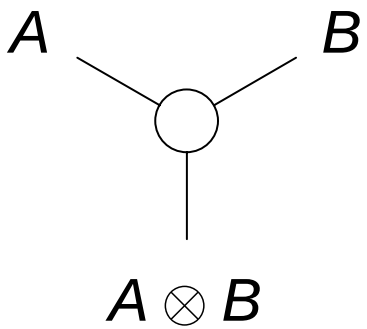
# Links
## From 2-Sided MLL to NL

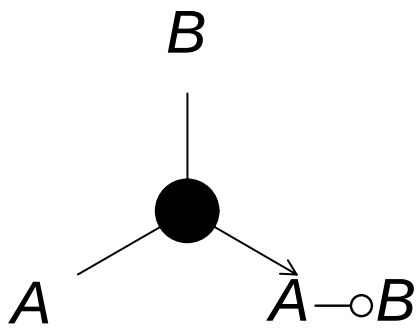


# Links
## From 2-Sided MLL to NL

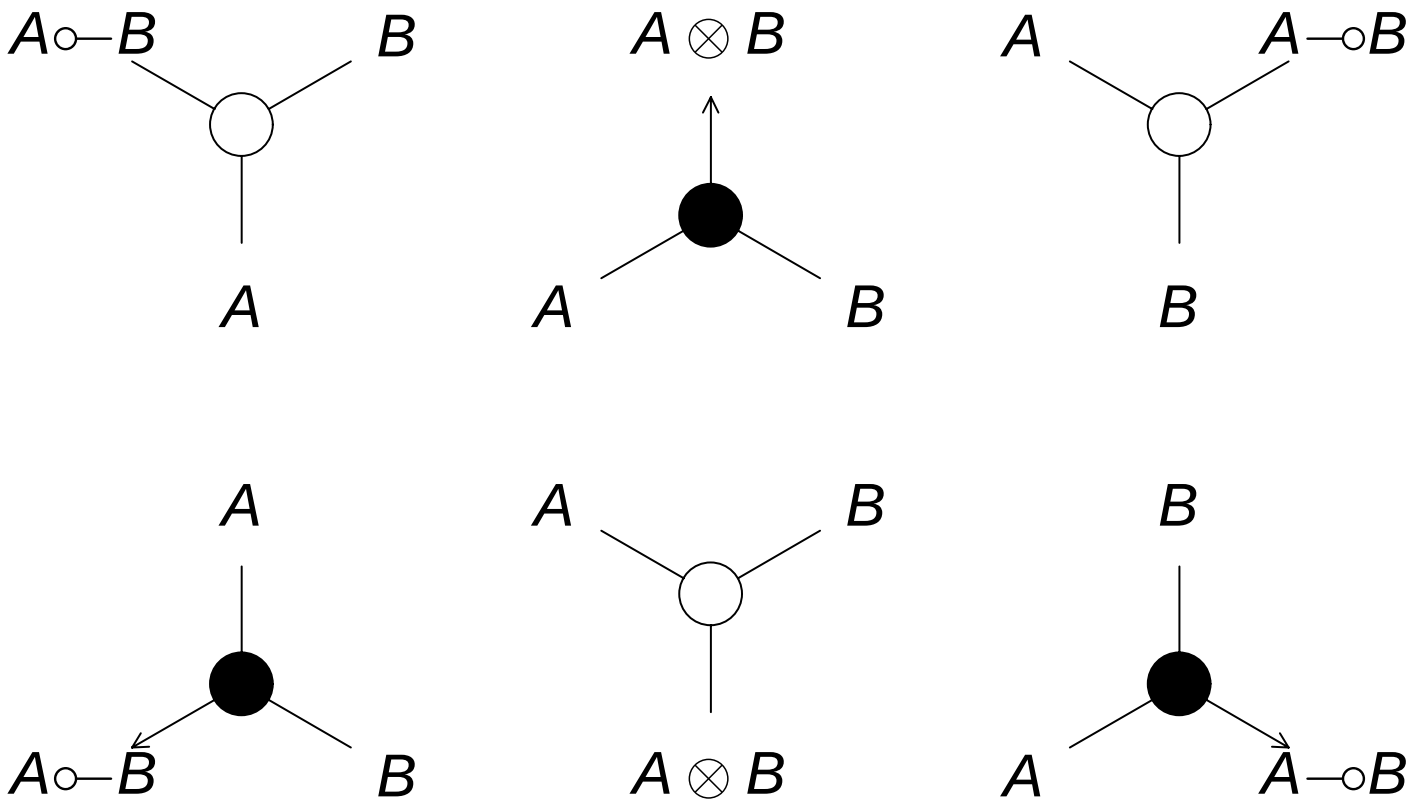


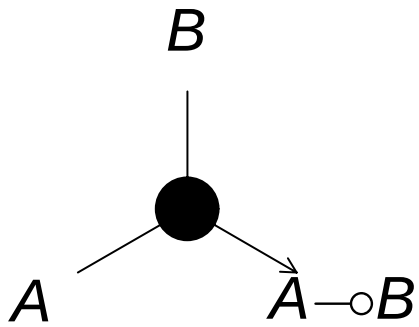



# Links
## From 2-Sided MLL to NL



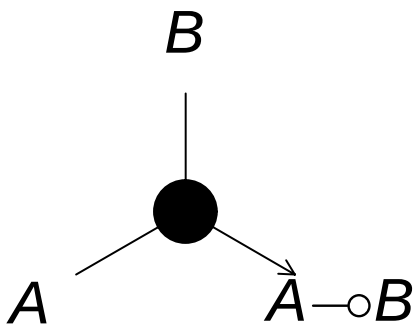
# Links
## From 2-Sided MLL to NL



# Links

## NL — Zoom on the par link for $\multimap$

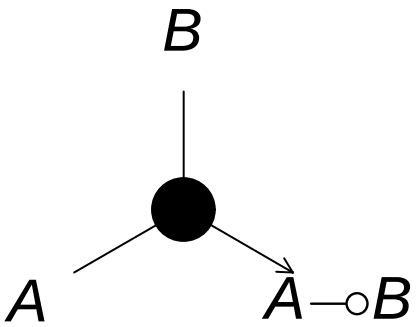
$$A \multimap B = A^{\perp} \parr B$$

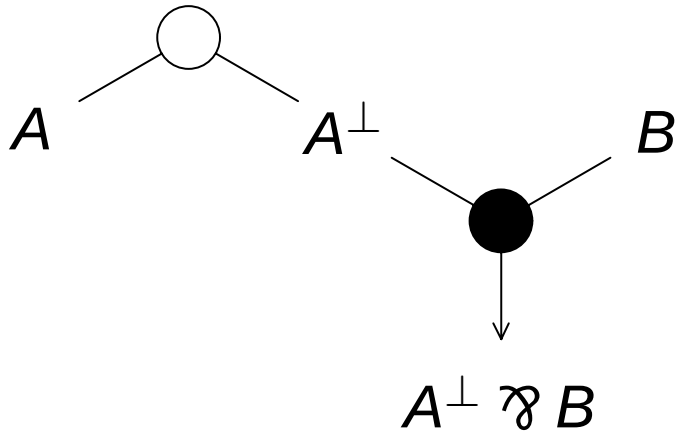


# Links

## NL — Zoom on the par link for $\multimap$

$$A \multimap B = A^{\perp} \parr B$$

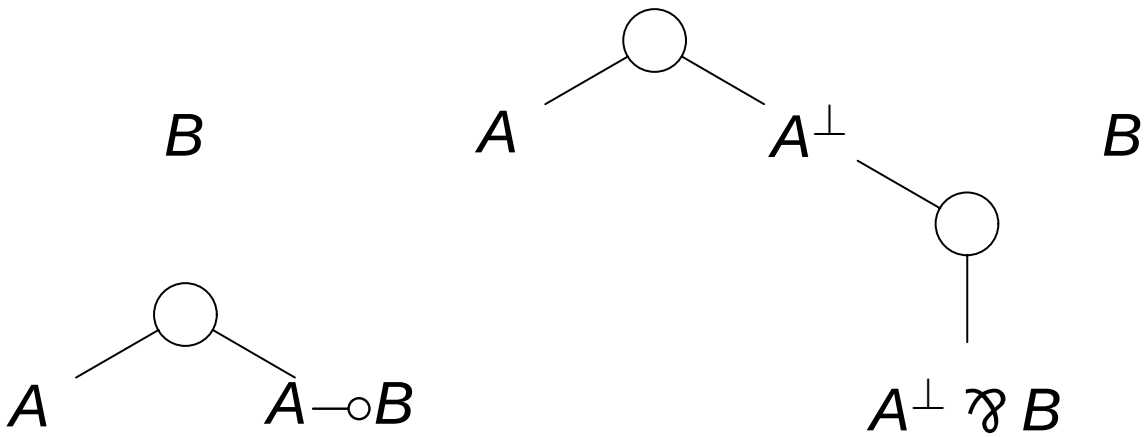


# Links

## NL — Zoom on the par link for $\multimap$

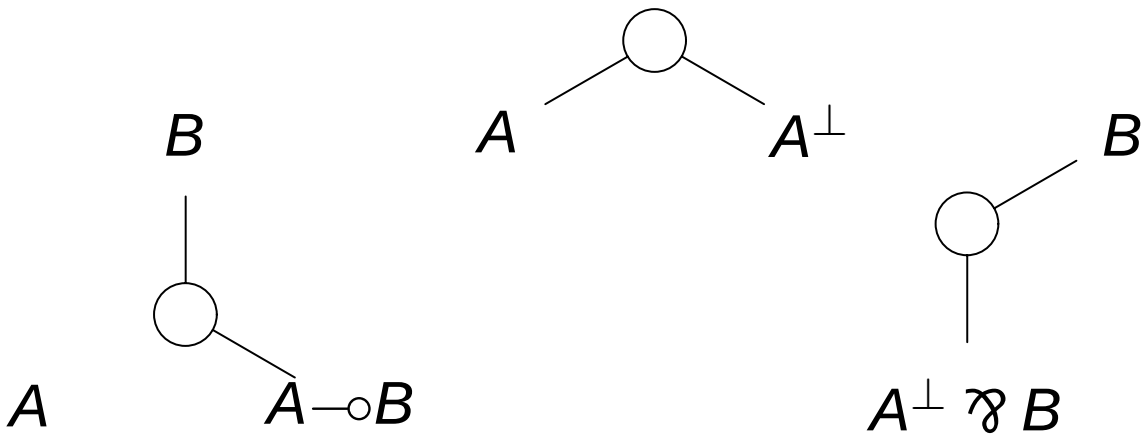
$$A \multimap B = A^{\perp} \parr B$$



# Links

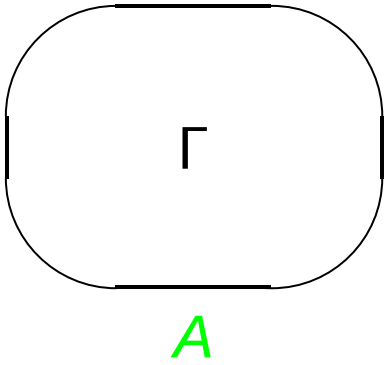
## NL — Zoom on the par link for $\multimap$

$$A \multimap B = A^{\perp} \parr B$$



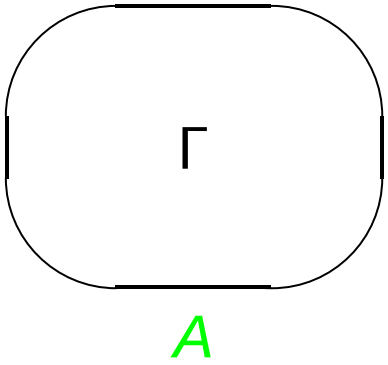
# Links

## NL — Where are the cuts and axioms?



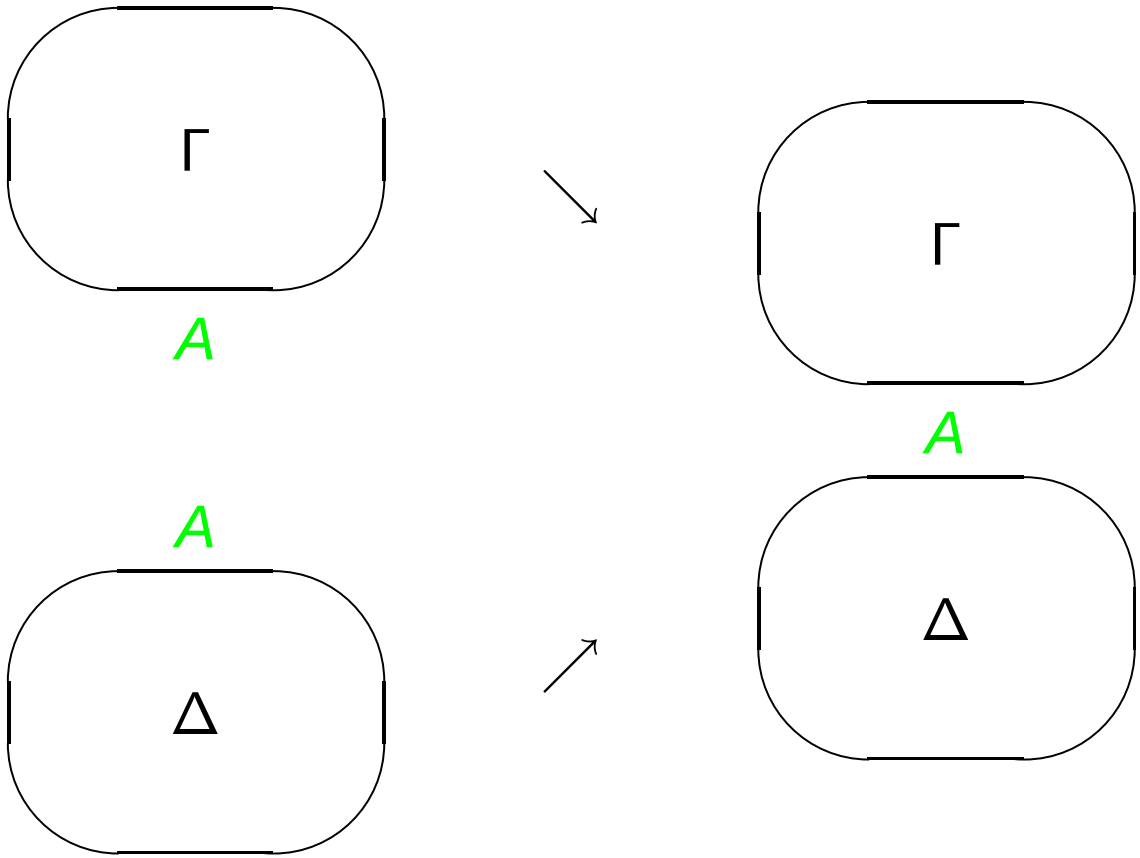
# Links

## NL — Where are the cuts and axioms?

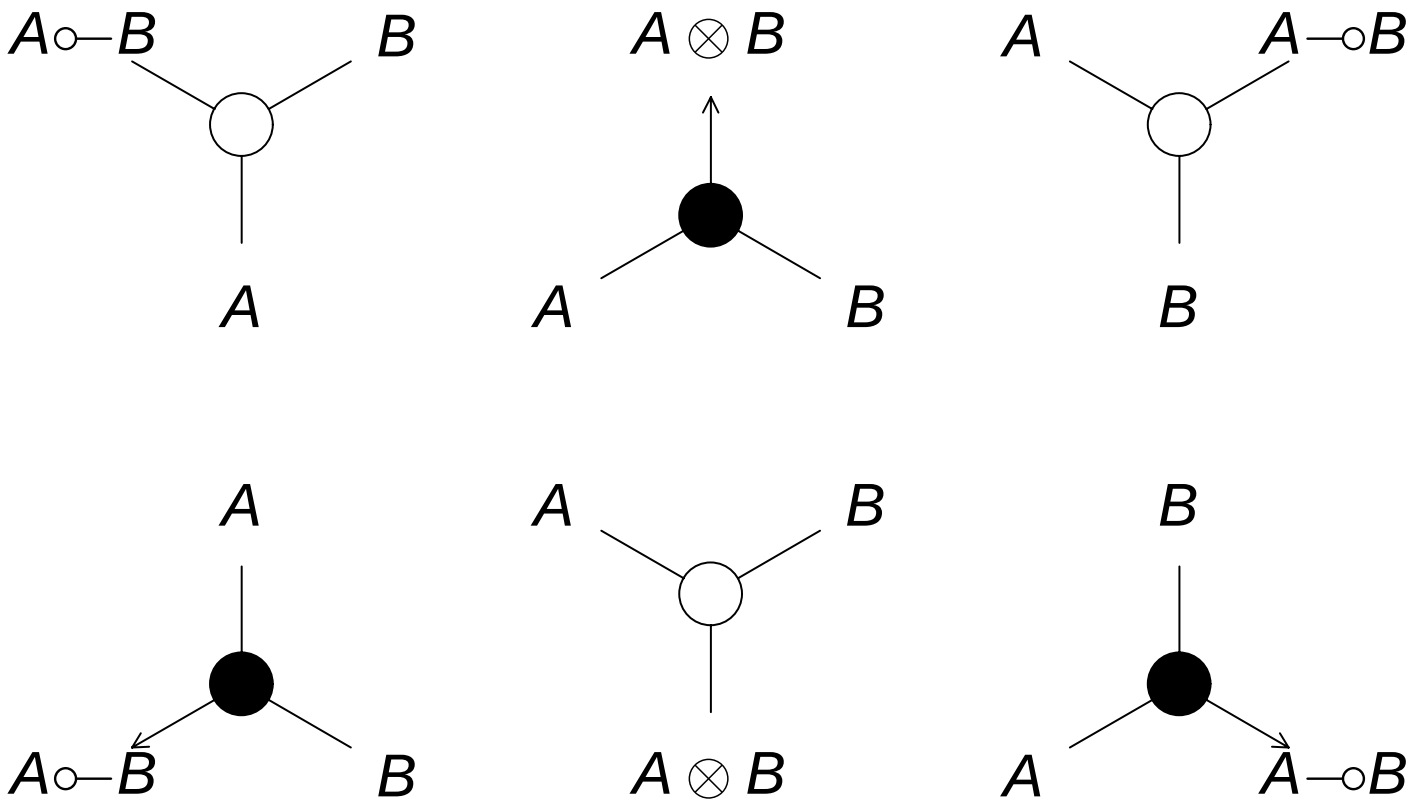


# Links

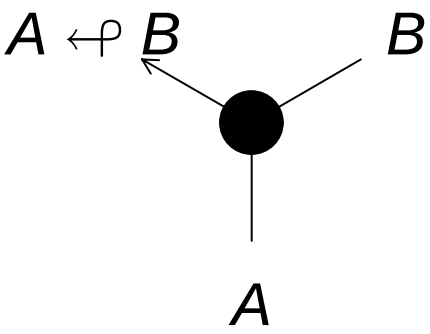
## NL — Where are the cuts and axioms?

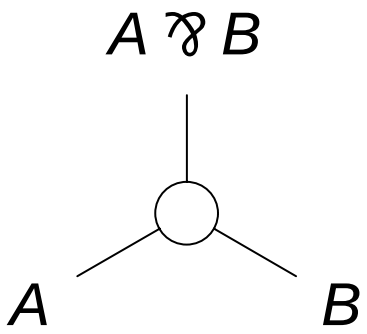


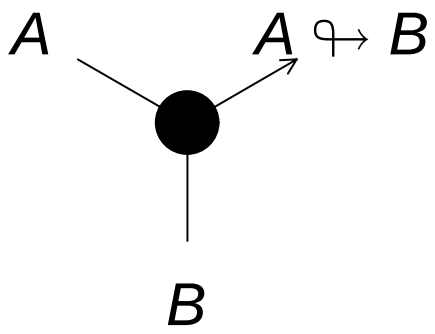
# Links
## LG

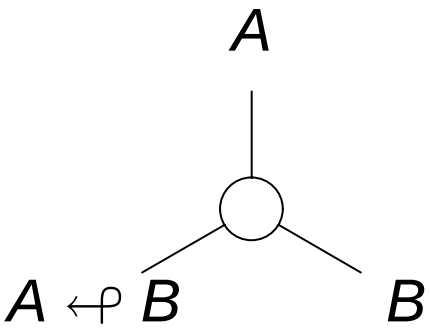


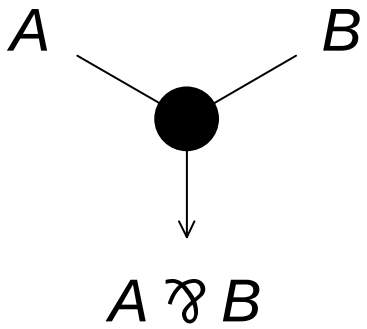
# Links
## LG

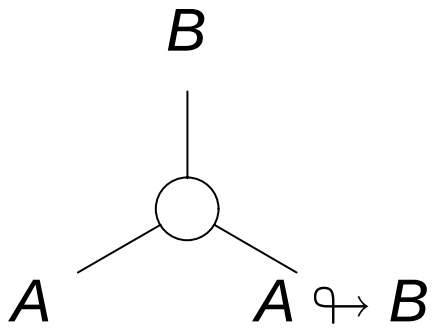


# Proof Structures
## Definition

### Definition

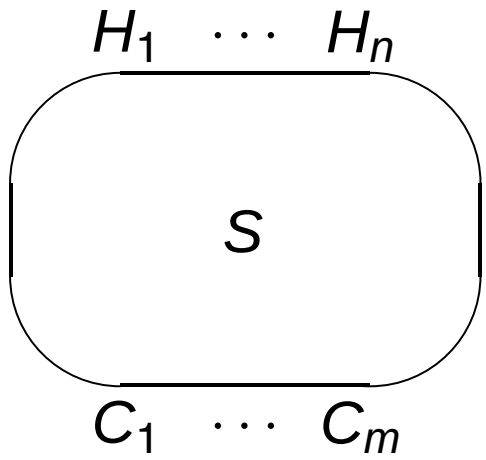
A *proof structure* $\langle S, \mathcal{L} \rangle$ is a finite set of formulas $S$ together with a set of links $\mathcal{L}$ such that.

- every formula of $S$ is at most once the premiss of a link.
- every formula of $S$ is at most once the conclusion of a link.

Formulas which are not the conclusion of any link are the *hypotheses H* of the proof structures, whereas the formulas which are not the premiss of any link are the it *conclusions C*.

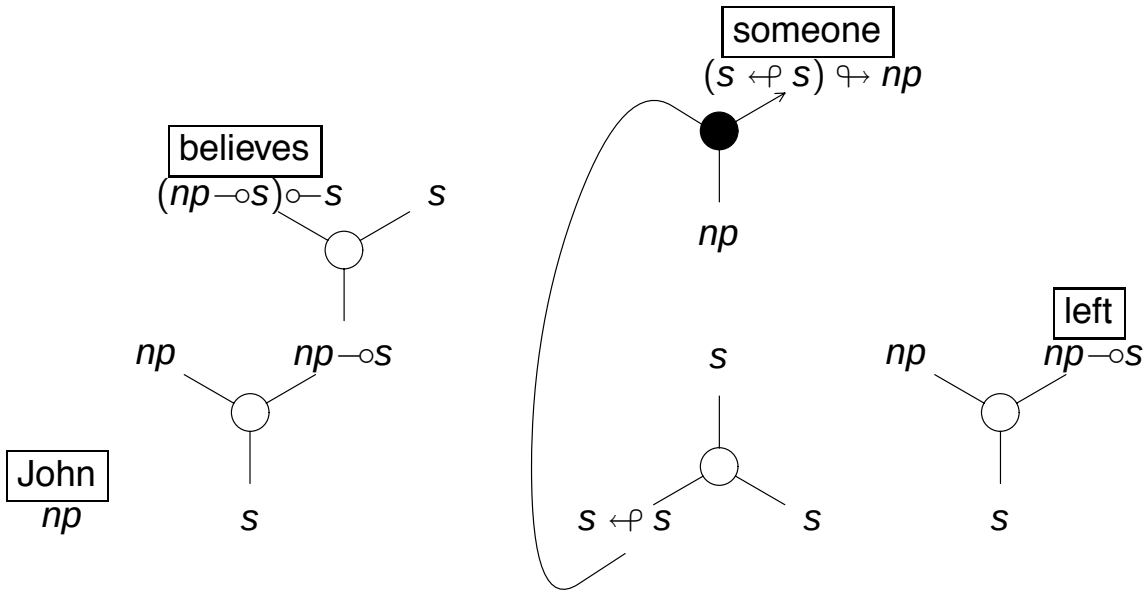
# Proof Structures

## General Form



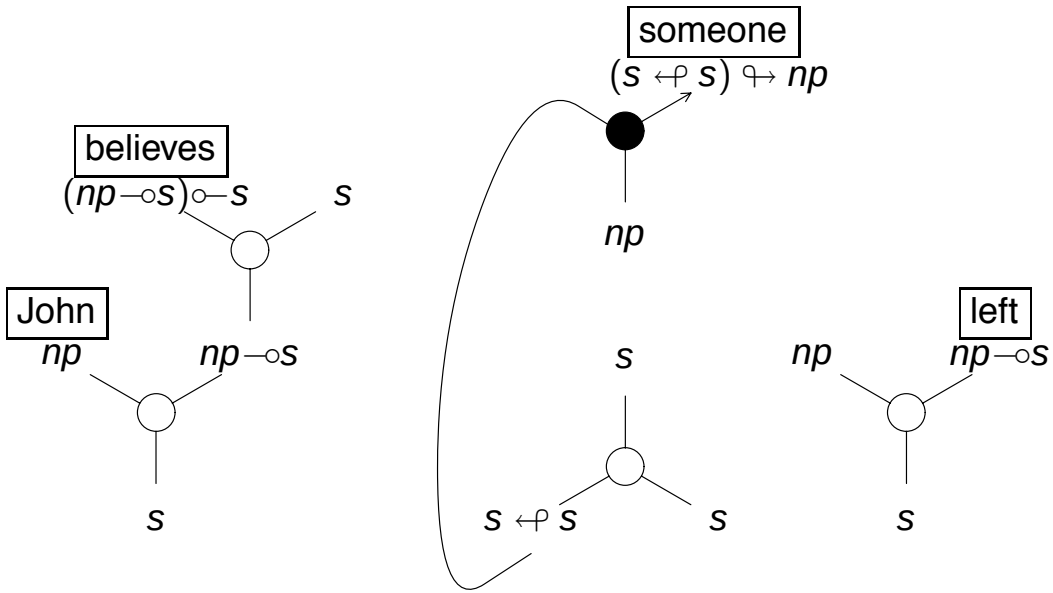
# Proof Structures

## Example: Lexical Proof Structure



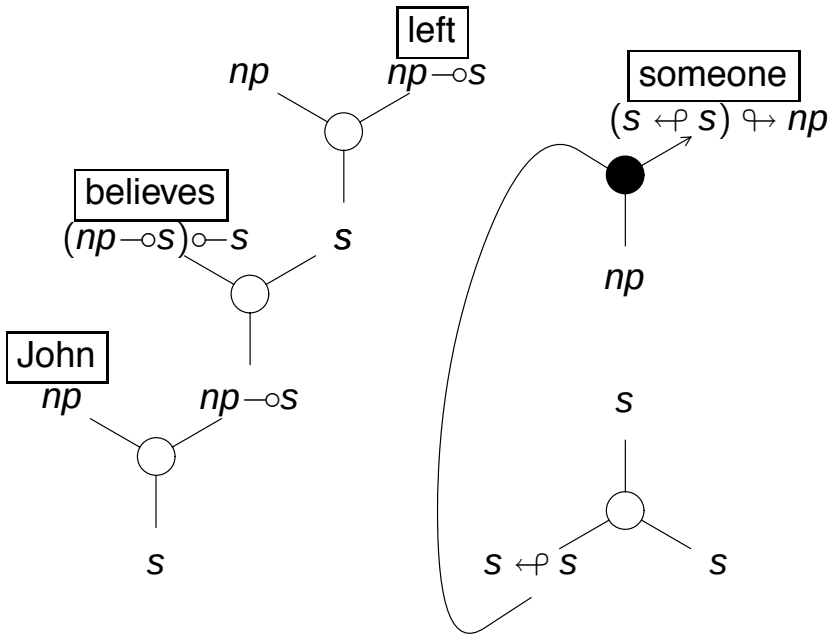
# Proof Structures

## Example: Connections



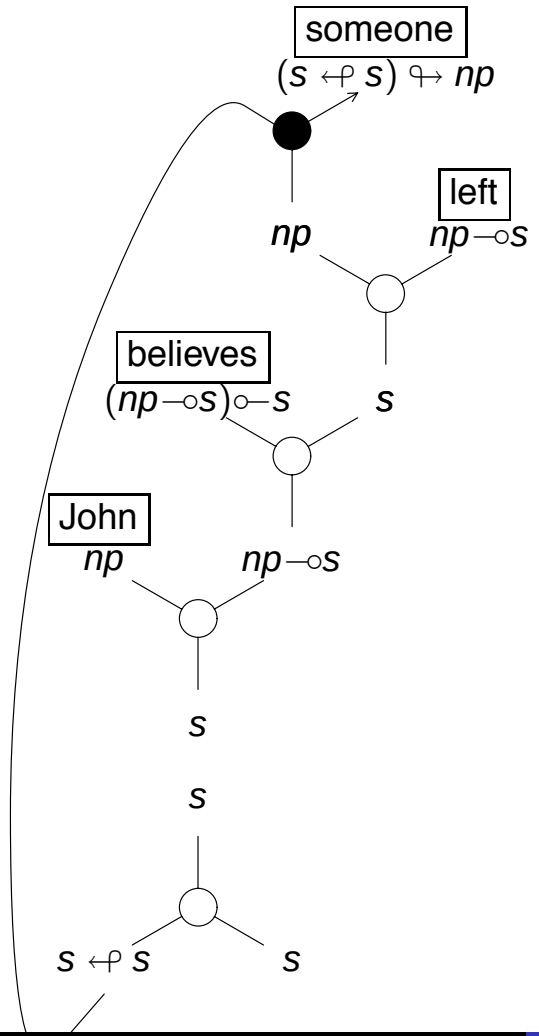
# Proof Structures

## Example: Connections



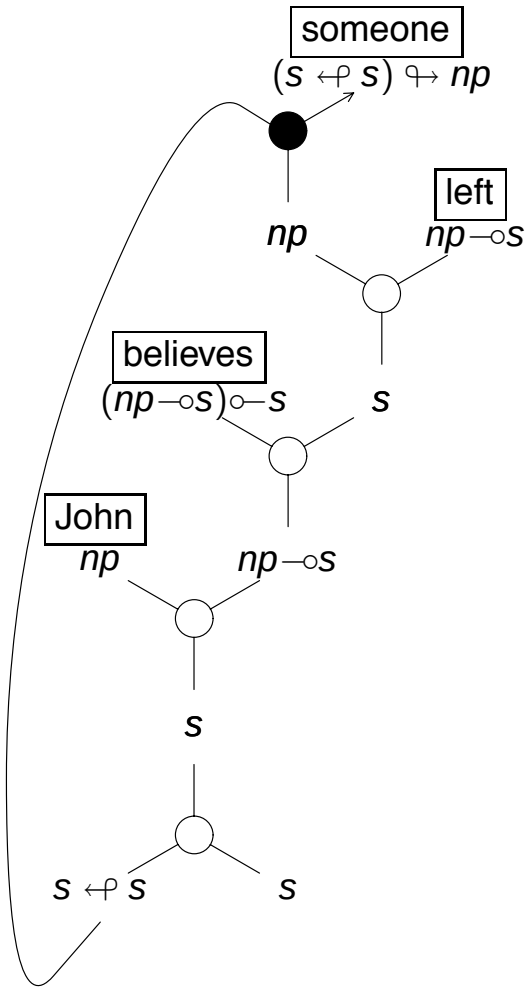
# Proof Structures

## Example: Connections



# Proof Structures

## Example: Connections



# Abstract Proof Structures

## Definition

### Definition

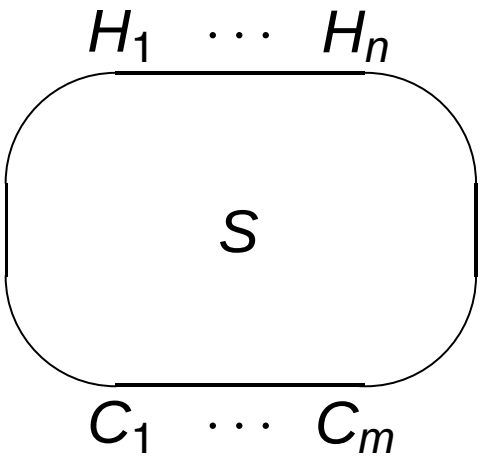
An *abstract proof structure* is a tuple $\langle V, \mathcal{L}, p, q \rangle$ such that.
$V$ is a finite set of vertices,
$\mathcal{L}$ is a set of links such that

- every vertex of $V$ is at most once the premiss of a link,
- every vertex of $V$ is at most once the conclusion of a link

$p$ is a labelling function assigning a formula to the hypotheses of the abstract proof structure, that is, to those formulas which are not the conclusion of any link,
$q$ is a labelling function assigning a formula to the conclusions of the abstract proof structure, that is, to those formulas which are not the premiss of any link.

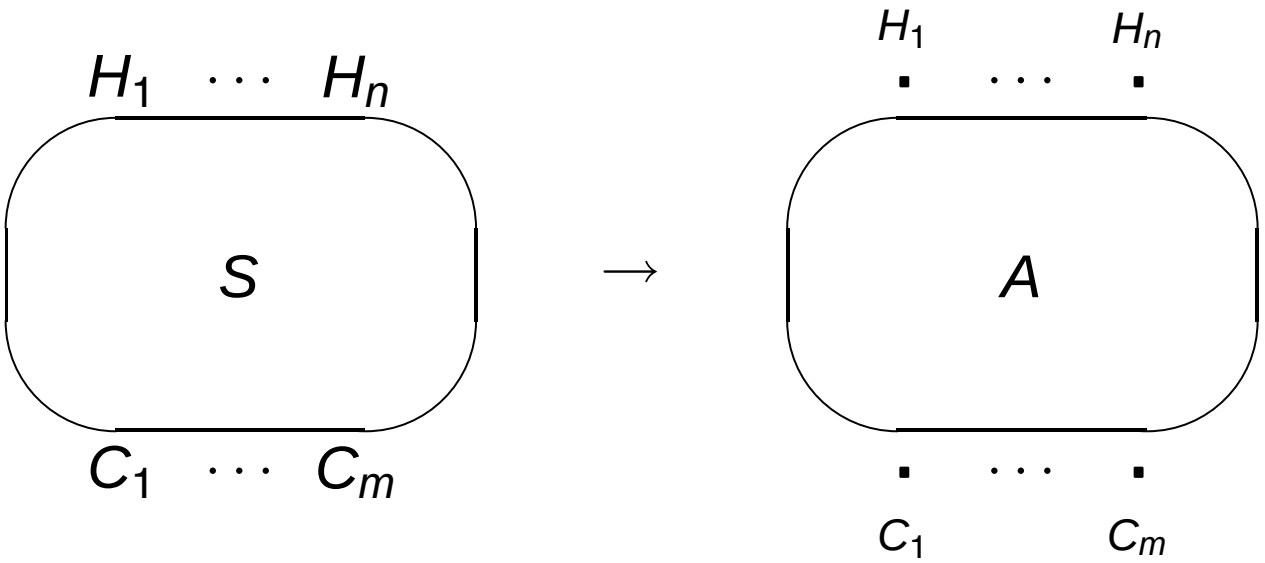
# Abstract Proof Structures

## General Form



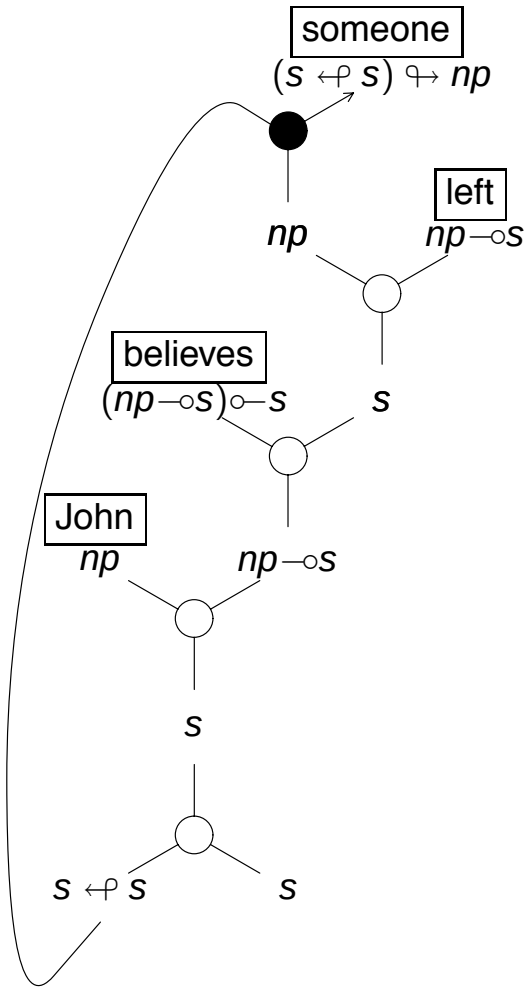
# Abstract Proof Structures

## General Form



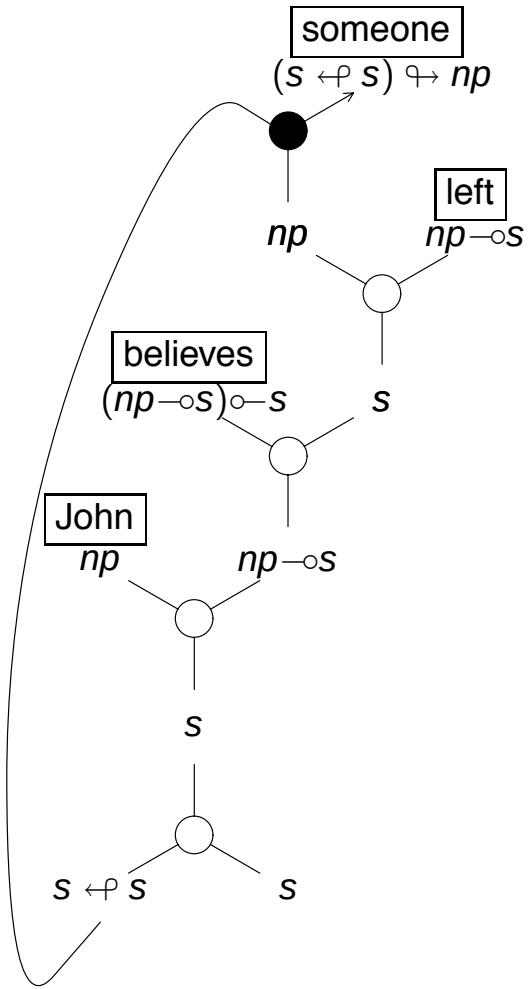
# Abstract Proof Structures

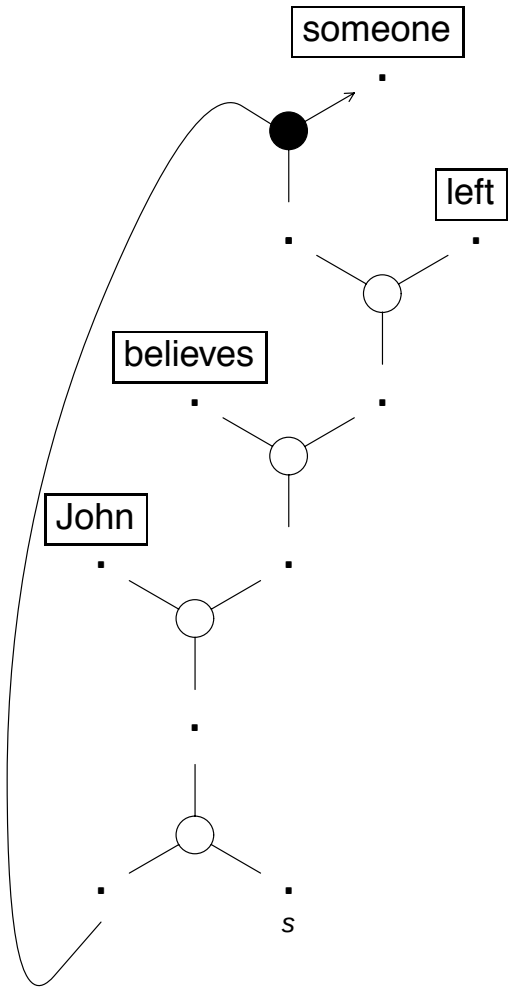
## Example: From Proof Structure to Abstract Proof Structure



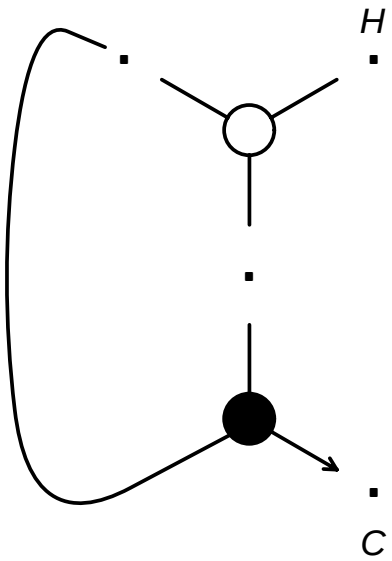
# Abstract Proof Structures

## Example: From Proof Structure to Abstract Proof Structure

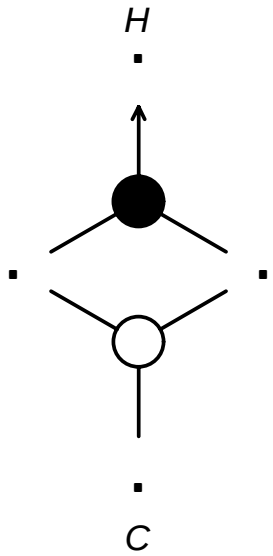


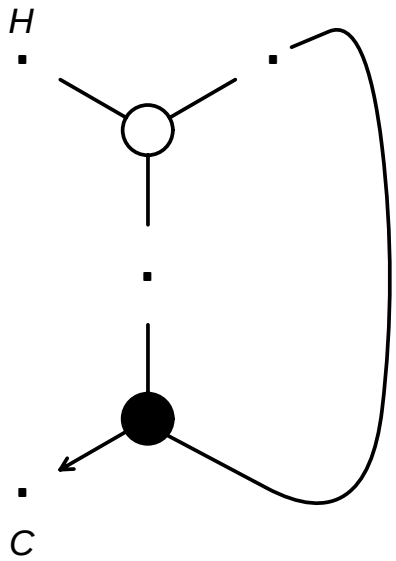
# Contractions
## Binary Residuated




$[R{\multimap}]$




$[L{\otimes}]$

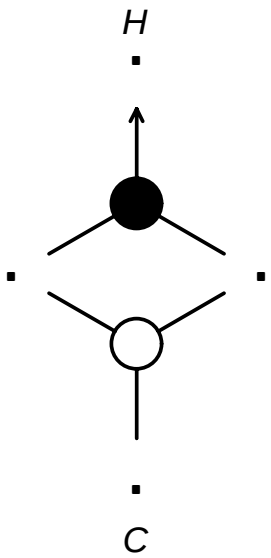



$[R{\circ\!\!-}]$

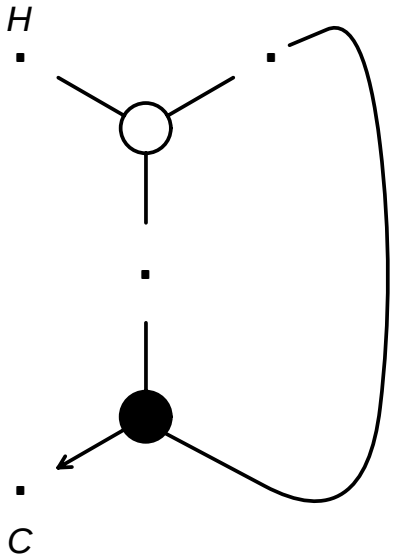
# Contractions

## Binary Residuated


$[R{\multimap}]$

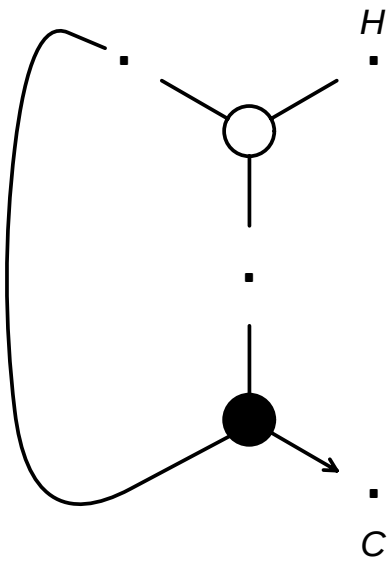



$[L{\otimes}]$



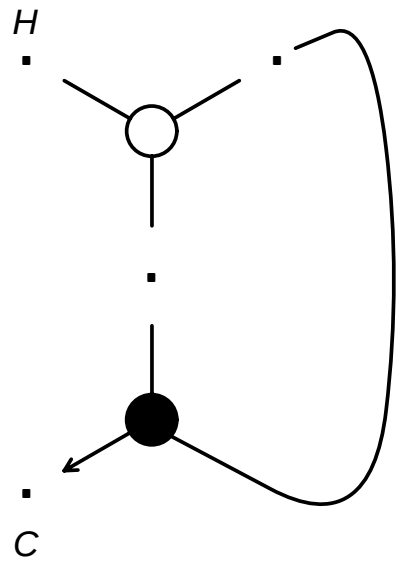

$[R{\circ\!\!-}]$

# Contractions
## Binary Residuated



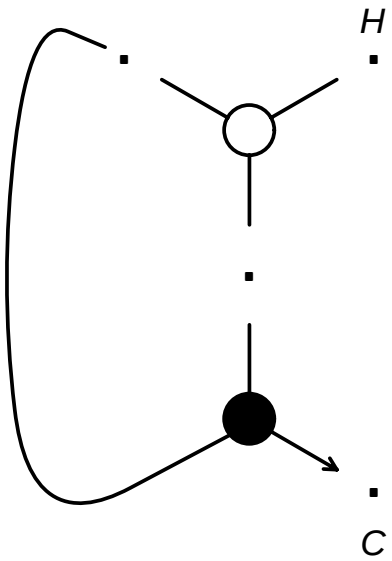

$[R{\multimap}]$


$[L{\otimes}]$

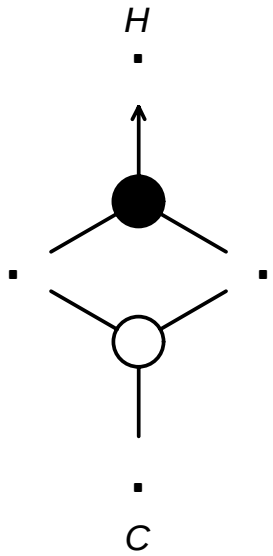



$[R{\circ\!\!-}]$

# Contractions
## Binary Residuated



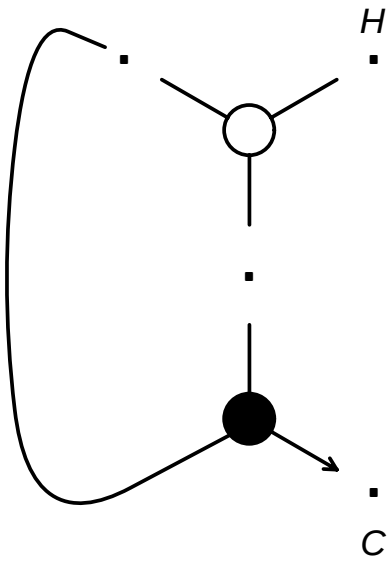

$[R{\multimap}]$



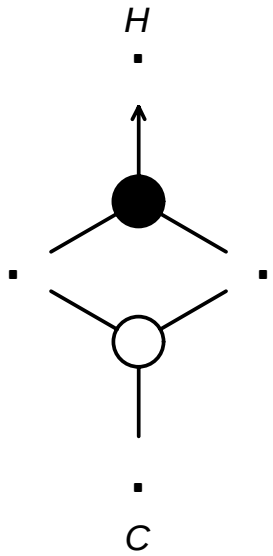

$[L{\otimes}]$


$[R{\circ\!\!-}]$

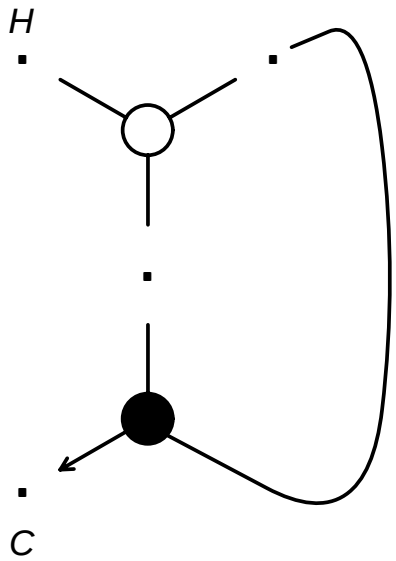
# Contractions
## Binary Residuated




$[R{\multimap}]$



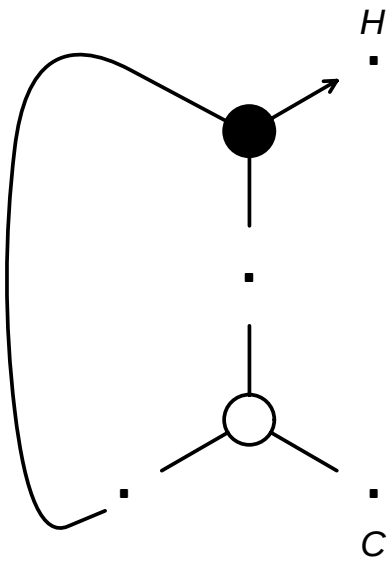

$[L{\otimes}]$



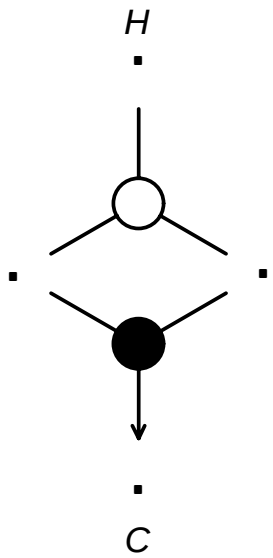

$[R{\circ\!\!-}]$

# Contractions

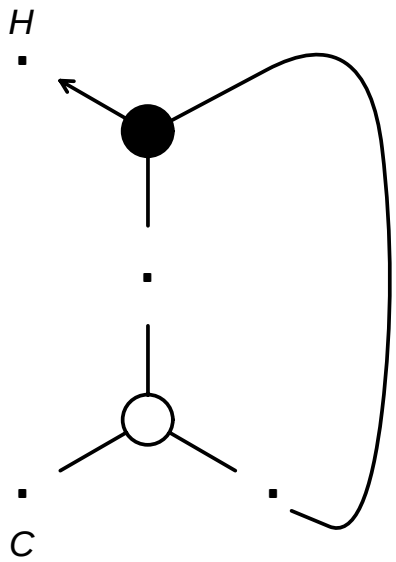
## Binary Dual Residuated




$[L \looparrowright]$



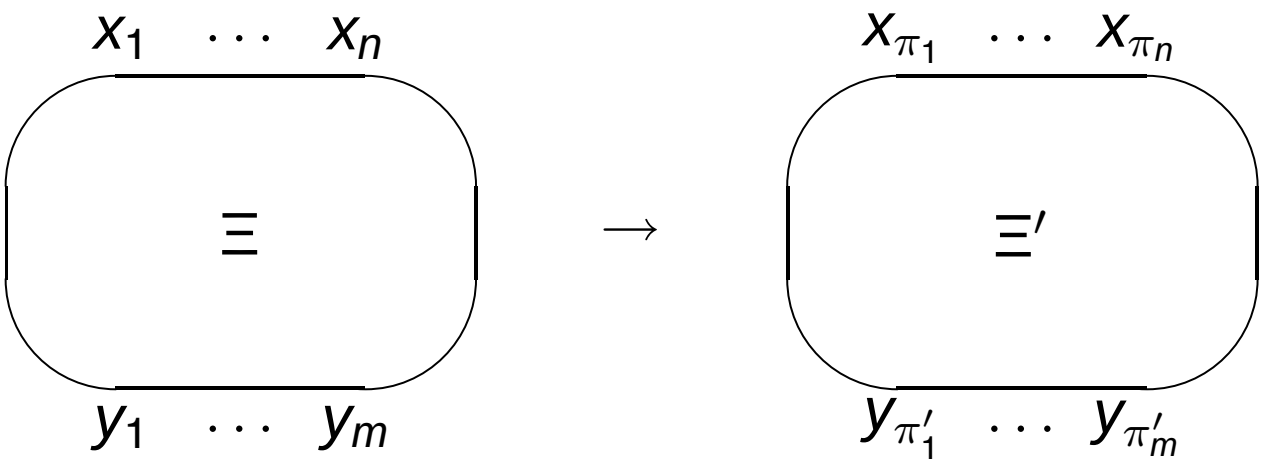

$[R\invamp]$




$[L \hookleftarrow]$

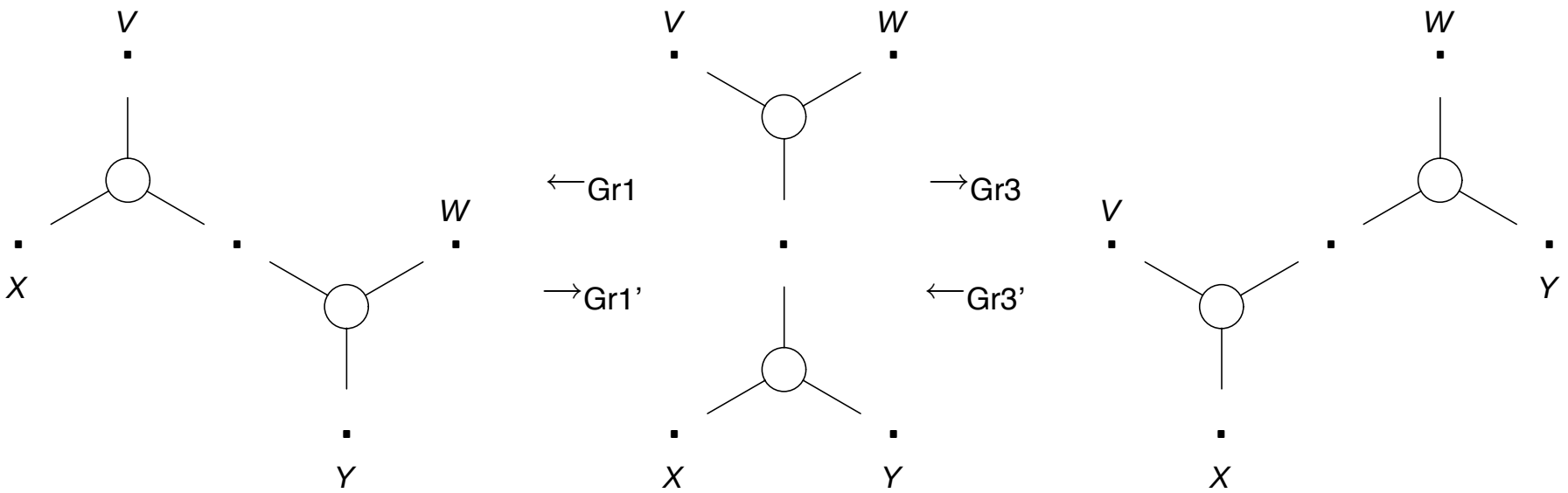
# Structural Rules

## Schematic Form



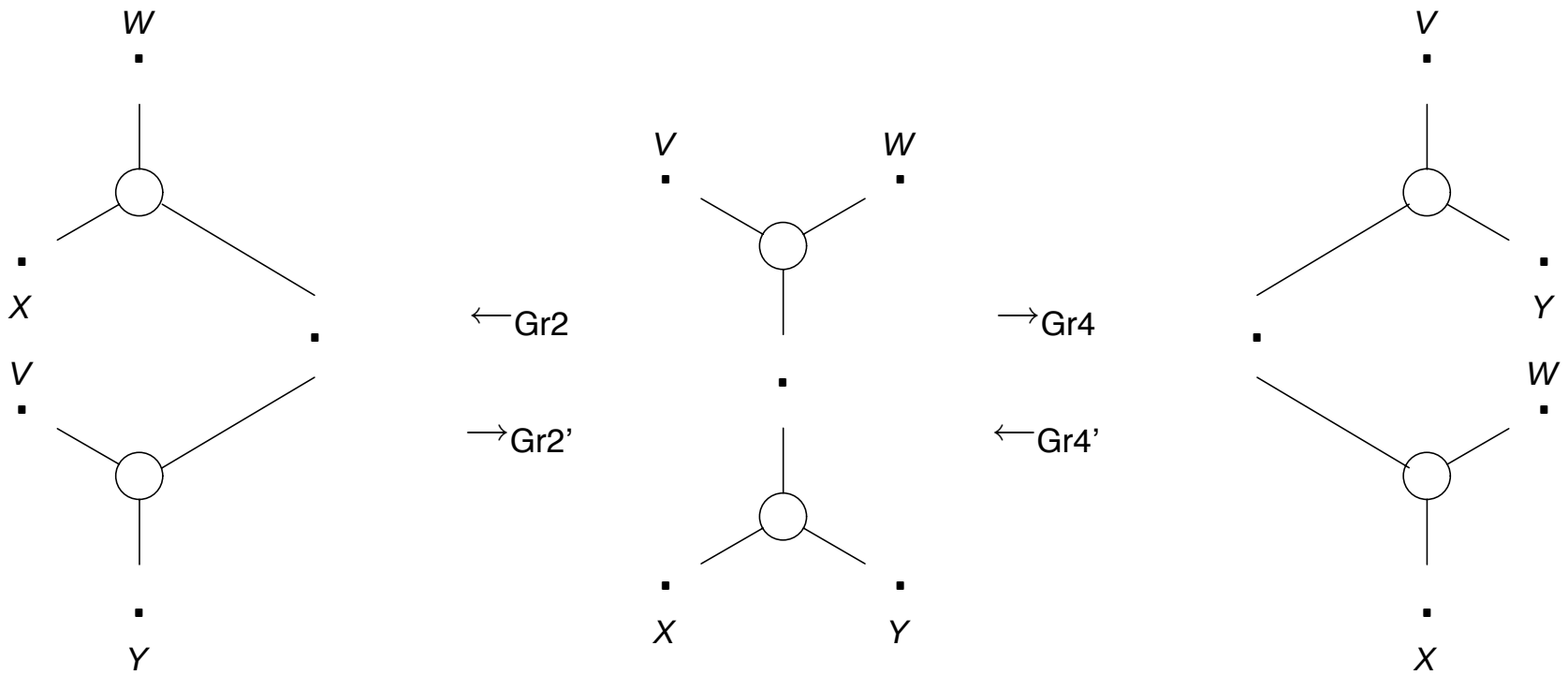
# Structural Rules

## Examples: Grishin



# Structural Rules

## Examples: Grishin (contd.)



# Proof Nets
## Definition

**Definition**

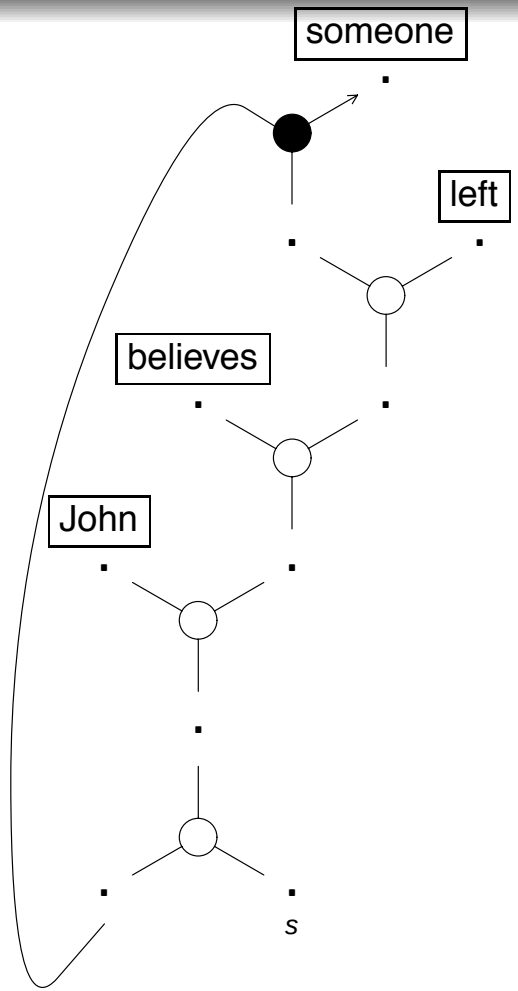
A *tensor tree* is an acyclic connected abstract proof structure containing only tensor links.

**Definition**

A proof structure is a *proof net* iff its abstract proof structure converts to a tensor tree using the structural conversions and the contractions.

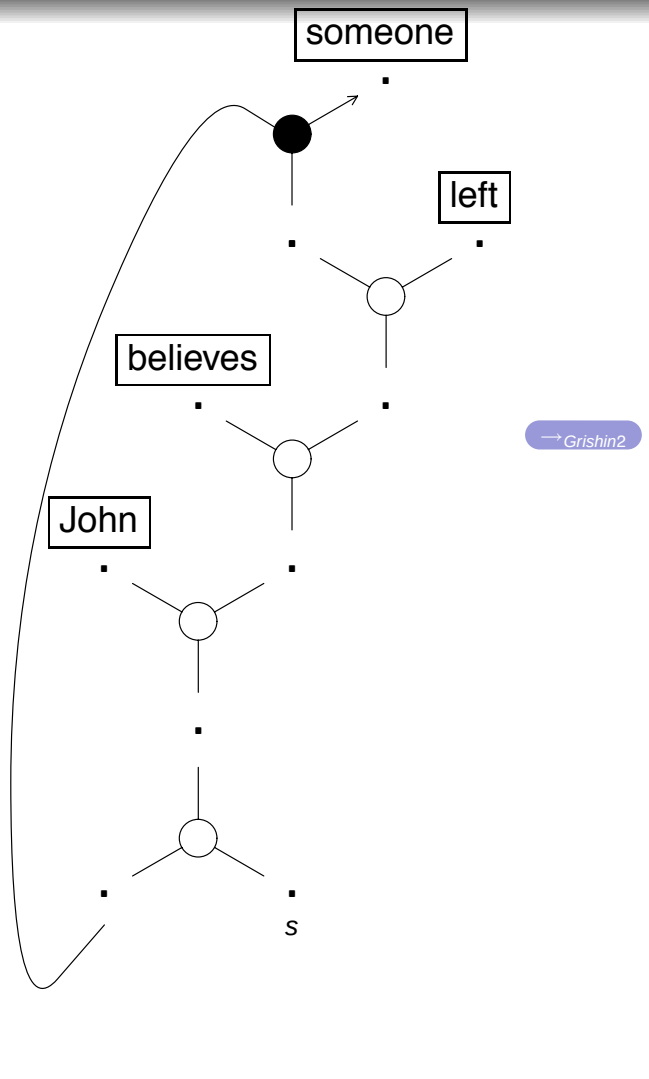
# Proof Nets

## Example: Contractions and Structural Rules

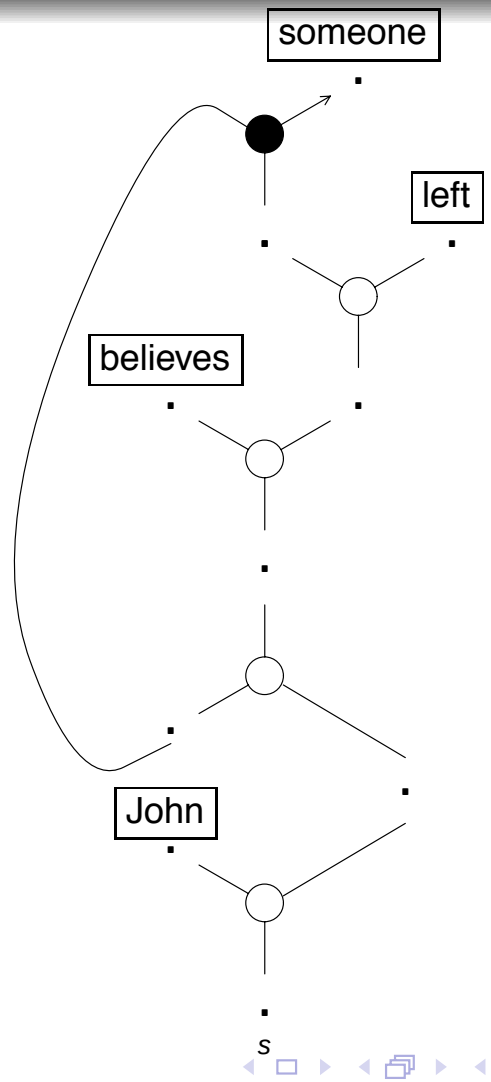


# Proof Nets

## Example: Contractions and Structural Rules

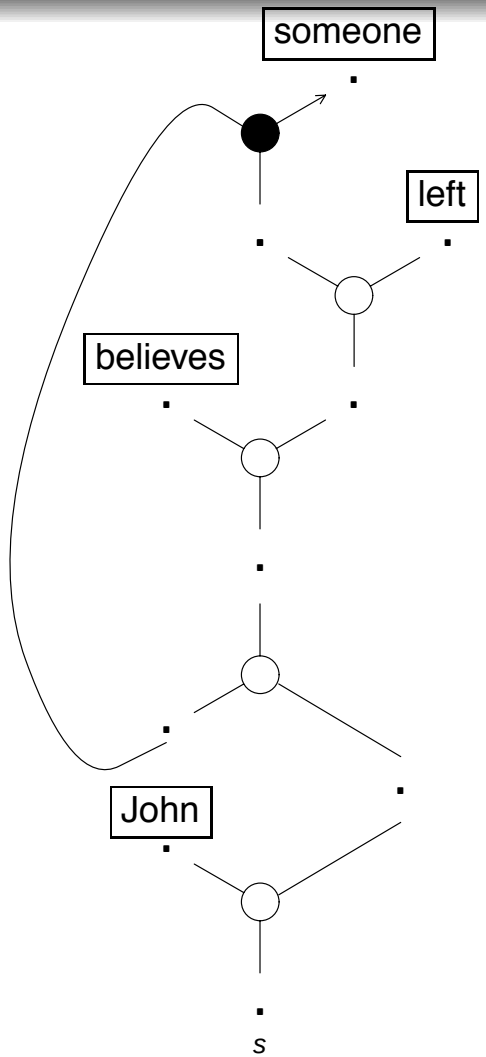



→Grishin2



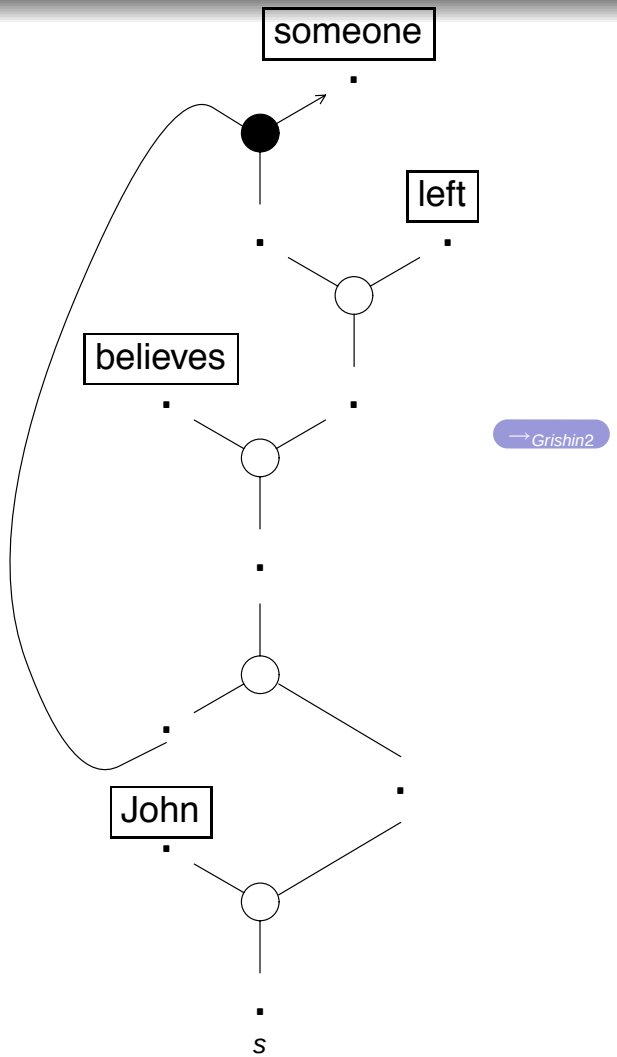
# Proof Nets

## Example: Contractions and Structural Rules

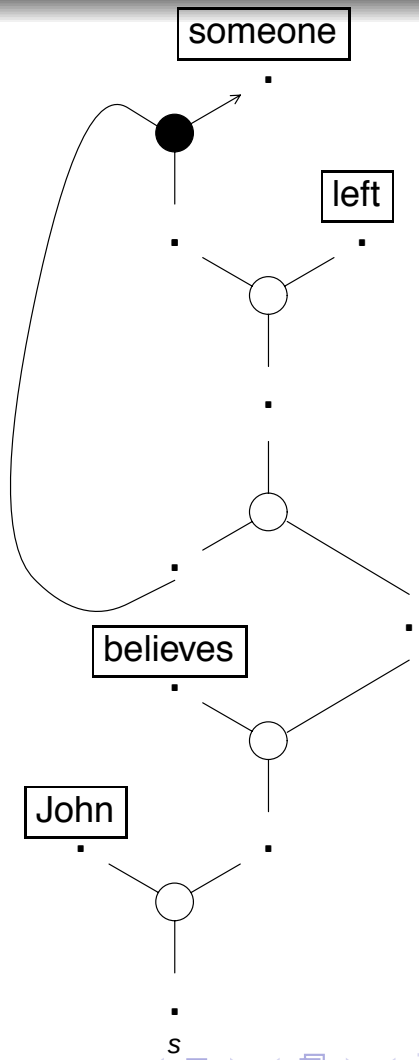


# Proof Nets

## Example: Contractions and Structural Rules

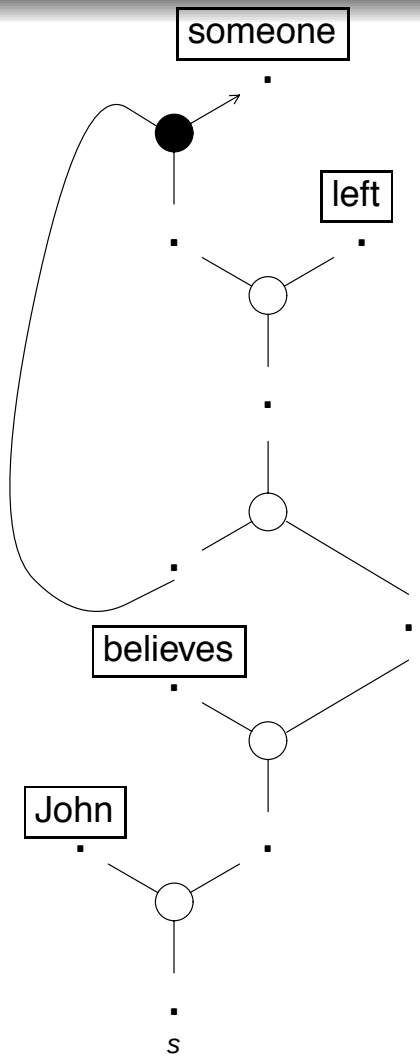



→Grishin2



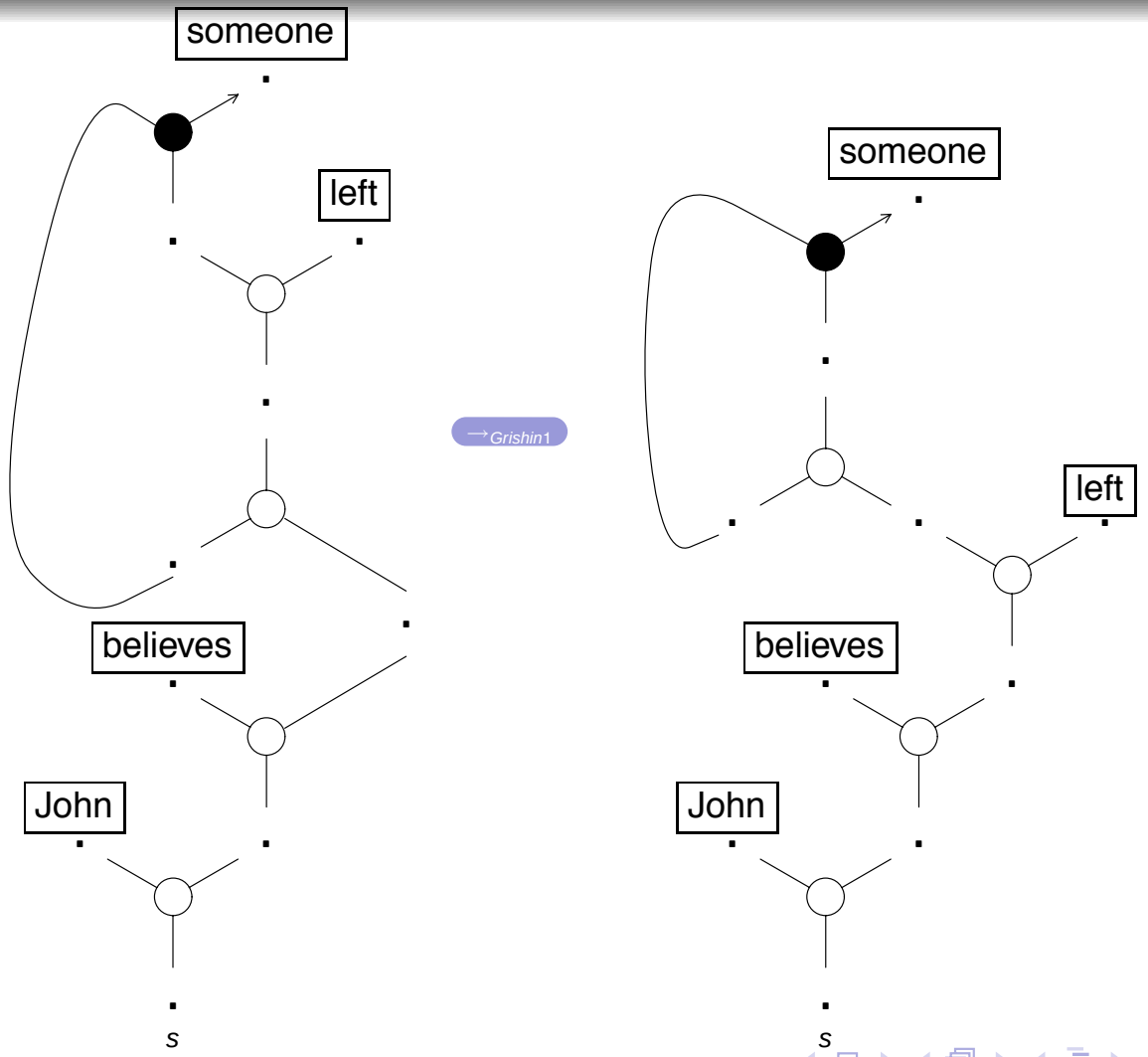
# Proof Nets

## Example: Contractions and Structural Rules



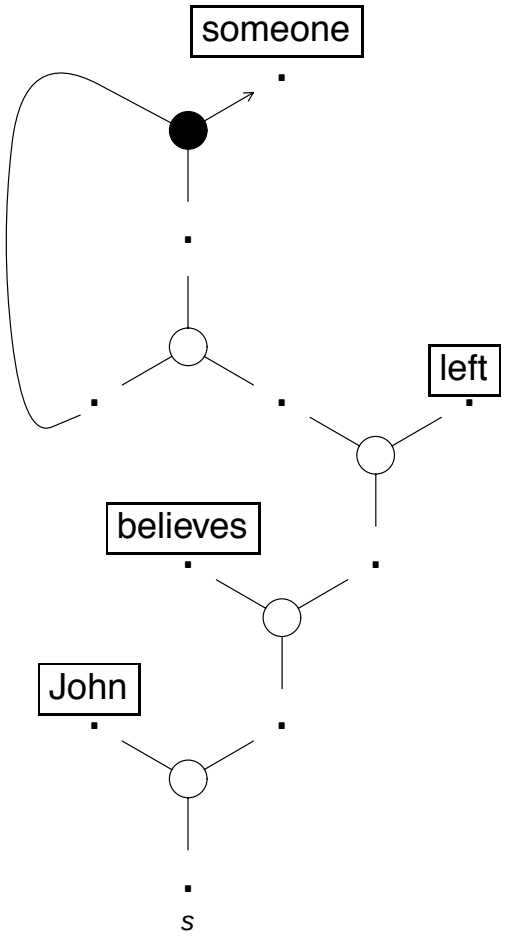
# Proof Nets

## Example: Contractions and Structural Rules



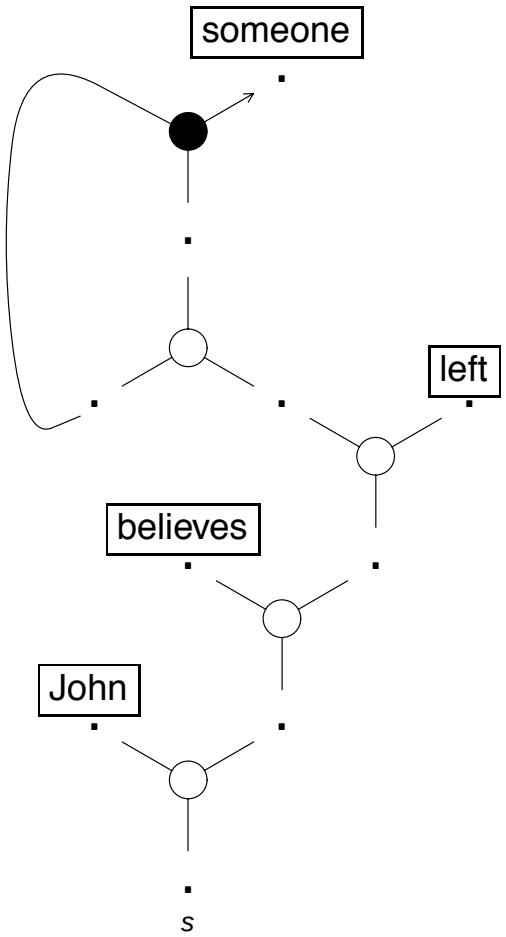
# Proof Nets

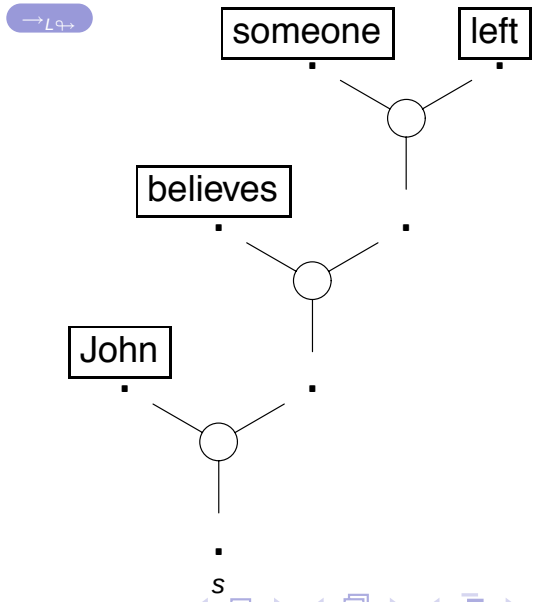
## Example: Contractions and Structural Rules



# Proof Nets

## Example: Contractions and Structural Rules



# Soundness and Completeness
## Theorem

**Theorem**

*A proof structure $\mathcal{S}$ is correct (ie. corresponds to a sequent proof of $\Gamma \vdash \Delta$) if and only if its abstract proof structure $\mathcal{A}$ converts to a tensor tree of $\Gamma \vdash \Delta$ (ie. $\mathcal{S}$ is a proof net).*

# Soundness and Completeness

## Soundness: Axiom

We proceed by induction on the depth of the sequent calculus proof. In case $d = 1$ we have an axiom. We verify it corresponds to a proof net.

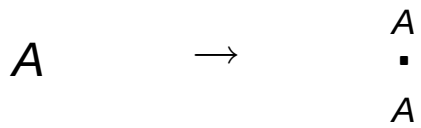
$$A \vdash A$$

# Soundness and Completeness

## Soundness: Axiom

We proceed by induction on the depth of the sequent calculus proof. In case $d = 1$ we have an axiom. We verify it corresponds to a proof net.

$$A \vdash A$$



# Soundness and Completeness

## Soundness: Inductive Case

Depth $d > 1$. We look at the last rule.

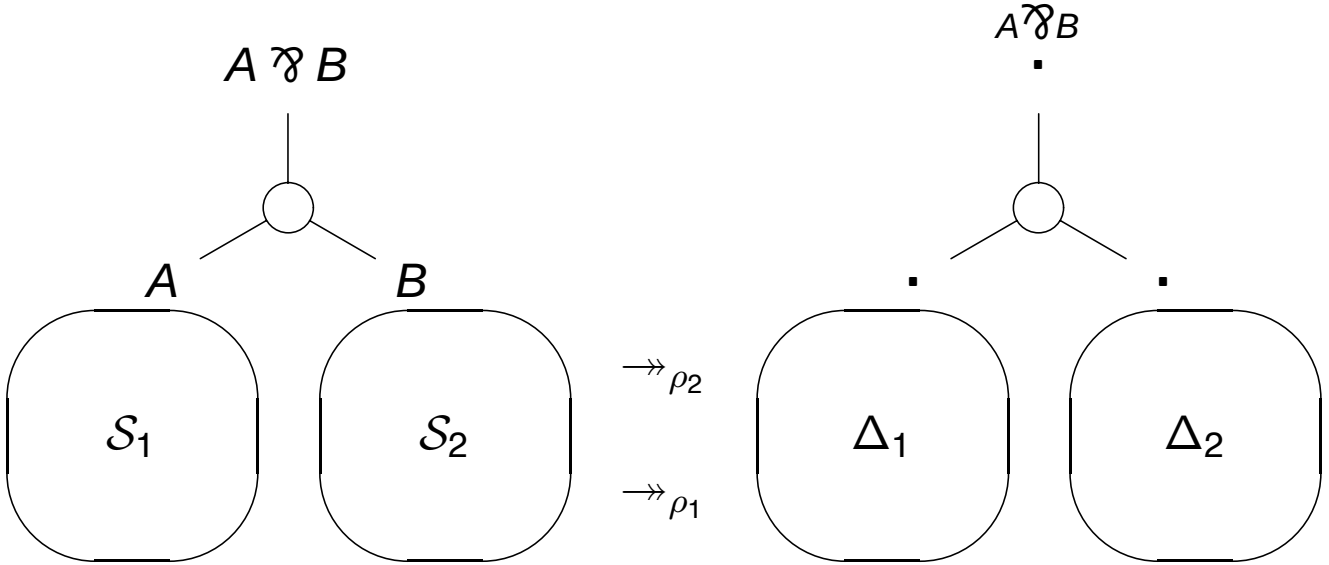
$$\frac{\begin{array}{cc}\vdots\ \pi_1 & \vdots\ \pi_2 \\ A \vdash \Delta_1 & B \vdash \Delta_2\end{array}}{A \parr B \vdash \Delta_1 \circ \Delta_2}\ [L\parr]$$

# Soundness and Completeness

## Soundness: Inductive Case

Depth $d > 1$. We look at the last rule.

$$\dfrac{\begin{array}{cc}\vdots\ \pi_1 & \vdots\ \pi_2 \\ A \vdash \Delta_1 & B \vdash \Delta_2\end{array}}{A \parr B \vdash \Delta_1 \circ \Delta_2}\ [L\parr]$$



# Soundness and Completeness

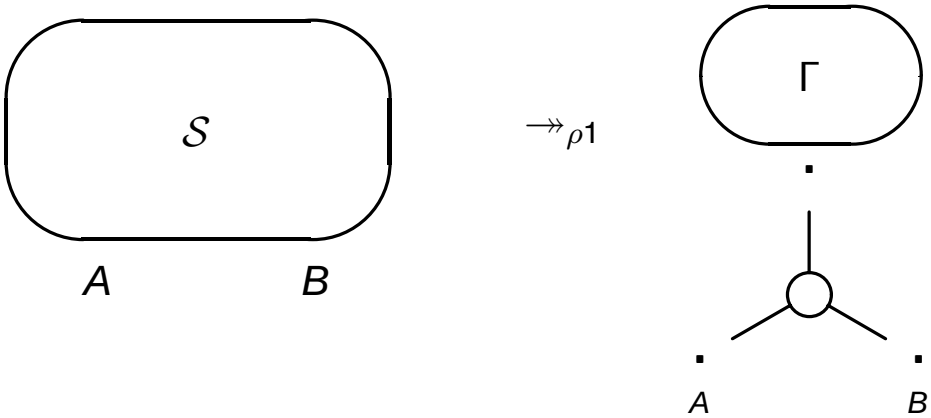
## Soundness: Inductive Case

$$\dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \Gamma \vdash A \circ B\end{array}}{\Gamma \vdash A \parr B}\ [R\parr]$$

# Soundness and Completeness

## Soundness: Inductive Case

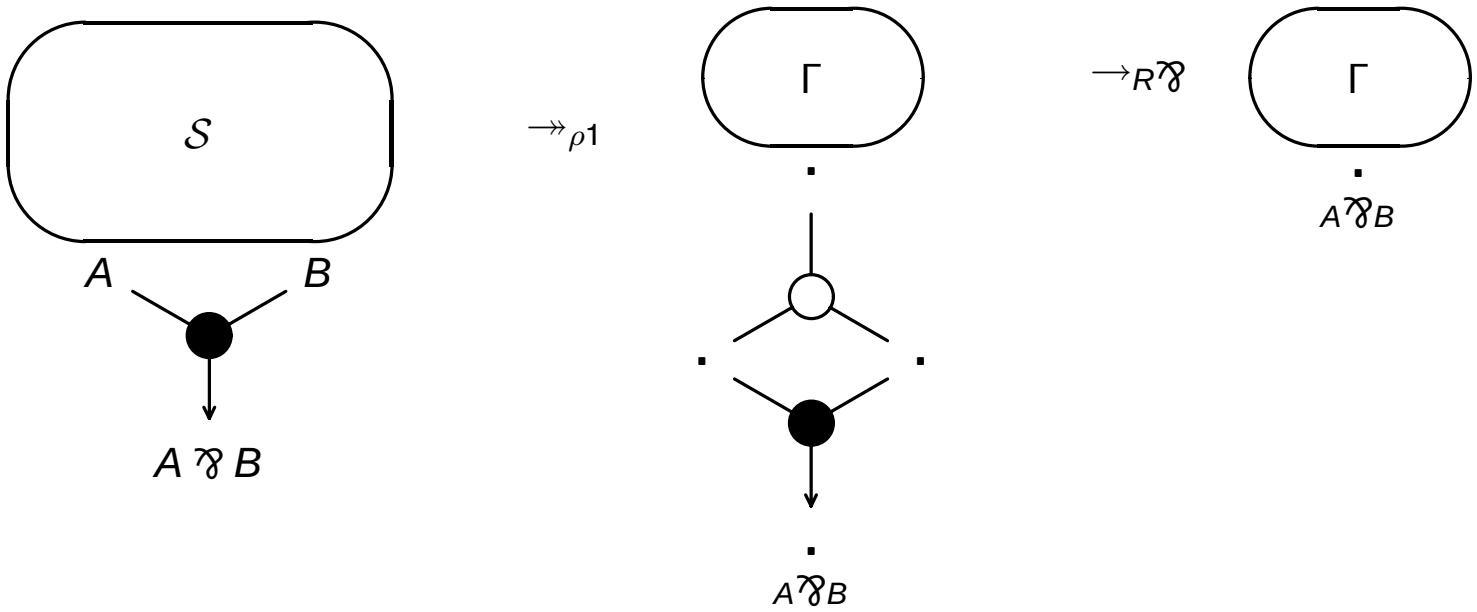
$$\frac{\begin{array}{c}\vdots\ \pi_1 \\ \Gamma \vdash A \circ B\end{array}}{\Gamma \vdash A \parr B}\ [R\parr]$$



# Soundness and Completeness

## Soundness: Inductive Case

$$\frac{\begin{array}{c}\vdots\ \pi_1 \\ \Gamma \vdash A \circ B\end{array}}{\Gamma \vdash A \parr B}\ [R\parr]$$



# Soundness and Completeness

## Completeness: Empty conversion sequence

We proceed by induction on the length $l$ of the conversion sequence. In case $l = 0$ then $\mathcal{S}$ corresponds to a tensor tree and we proceed by induction on the number of tensor links $t$.

$$t = 0$$

# Soundness and Completeness

## Completeness: Empty conversion sequence

We proceed by induction on the length $l$ of the conversion sequence. In case $l = 0$ then $\mathcal{S}$ corresponds to a tensor tree and we proceed by induction on the number of tensor links $t$.

$$t = 0$$

$$A \rightarrow \begin{matrix} A \\ \vdots \\ A \end{matrix}$$

# Soundness and Completeness

## Completeness: Empty conversion sequence

$$t > 0$$

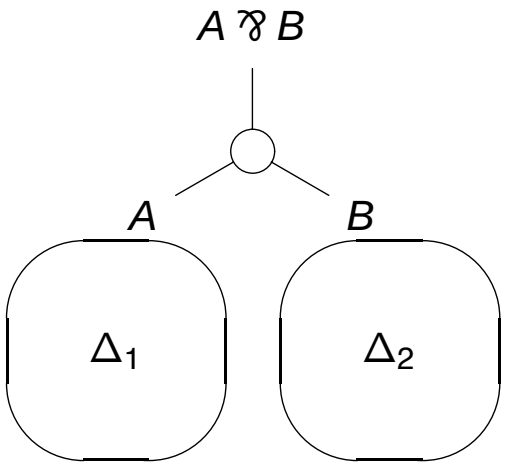
We find the principal leaf.

# Soundness and Completeness

## Completeness: Empty conversion sequence

$$t > 0$$

We find the principal leaf.

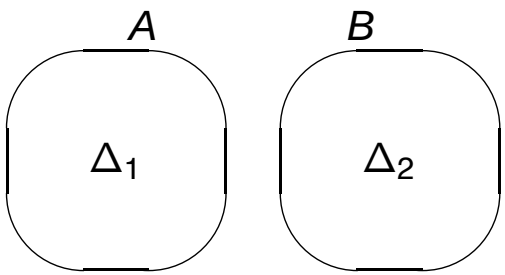


# Soundness and Completeness

## Completeness: Empty conversion sequence

$$t > 0$$

We find the principal leaf.



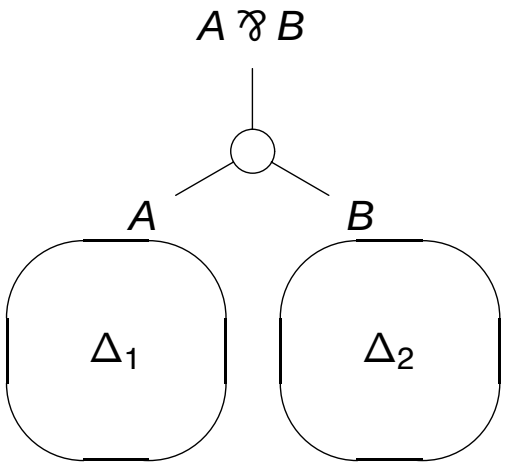
# Soundness and Completeness

## Completeness: Empty conversion sequence

$$t > 0$$

We find the principal leaf.

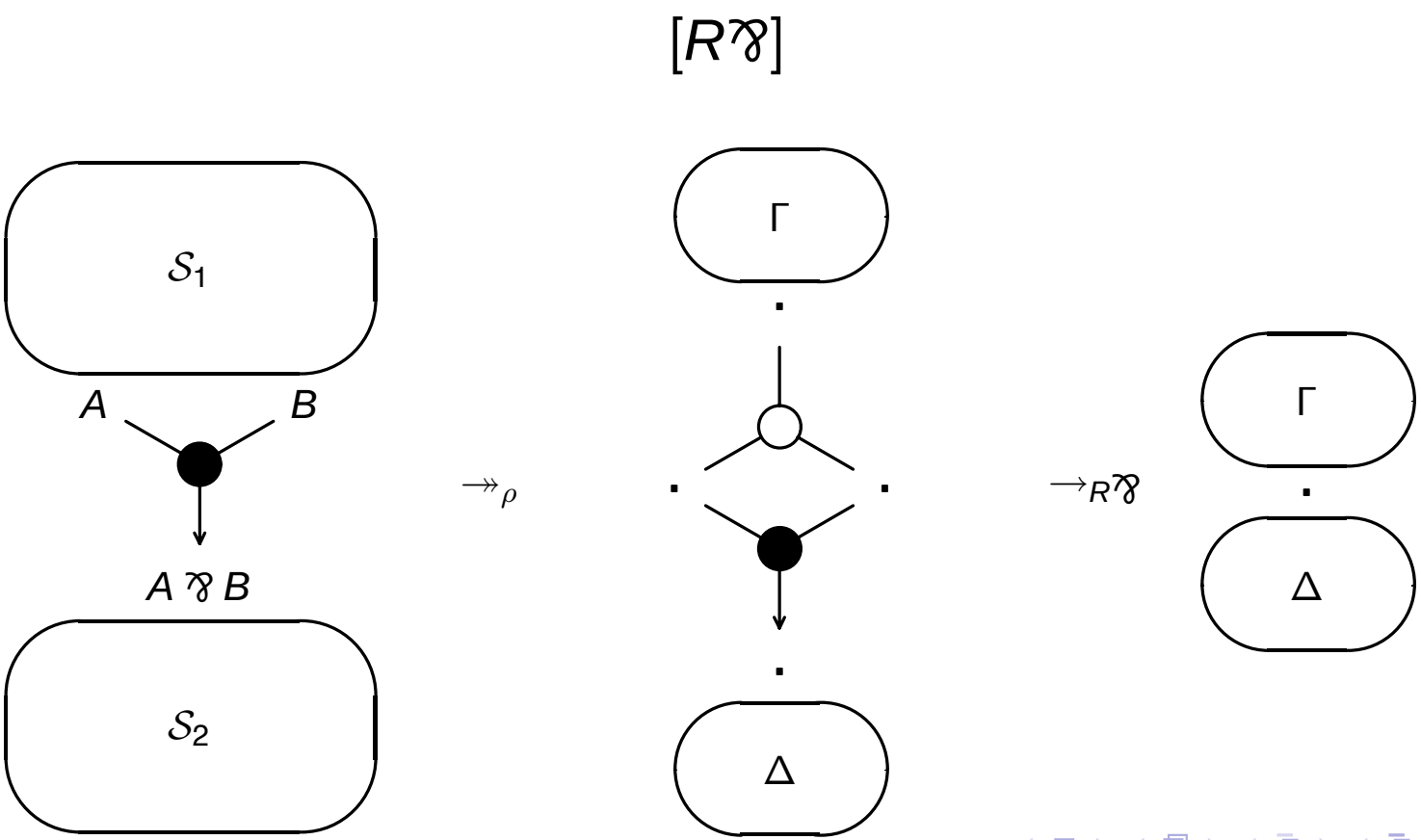
$$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{A \parr B \vdash \Delta_1 \circ \Delta_2}\ [L\parr]$$



# Soundness and Completeness

## Completeness: Inductive Case

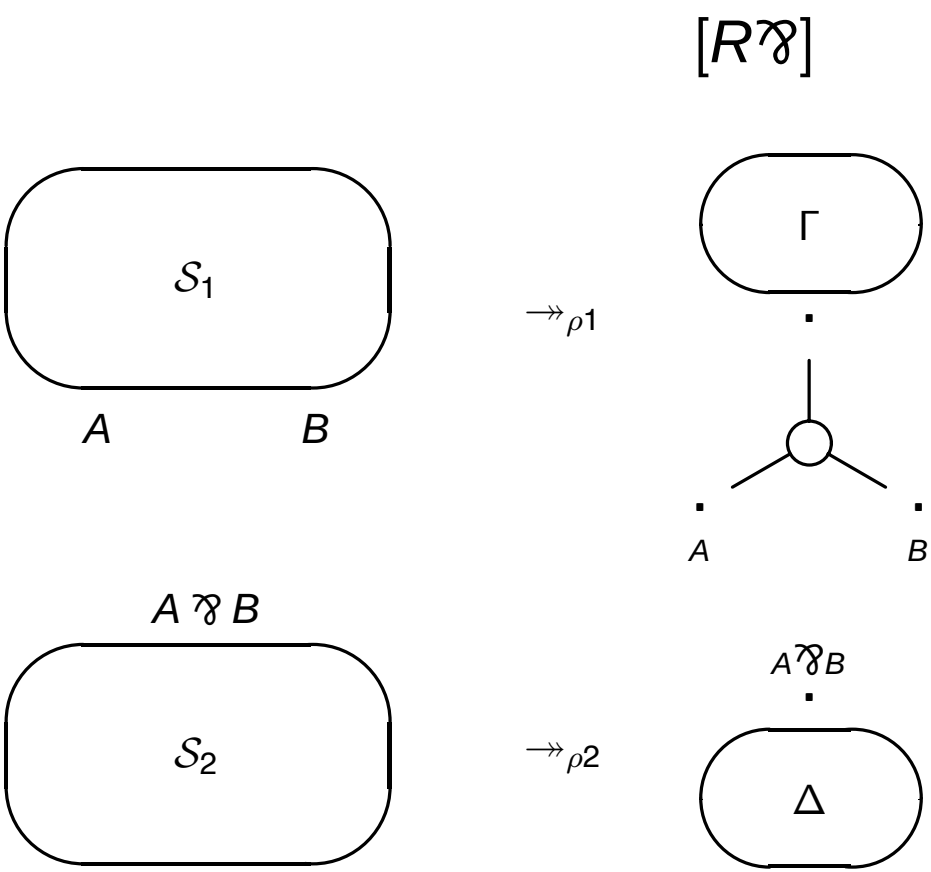
In case $l > 0$, we look at the last conversion.



# Soundness and Completeness

## Completeness: Inductive Case

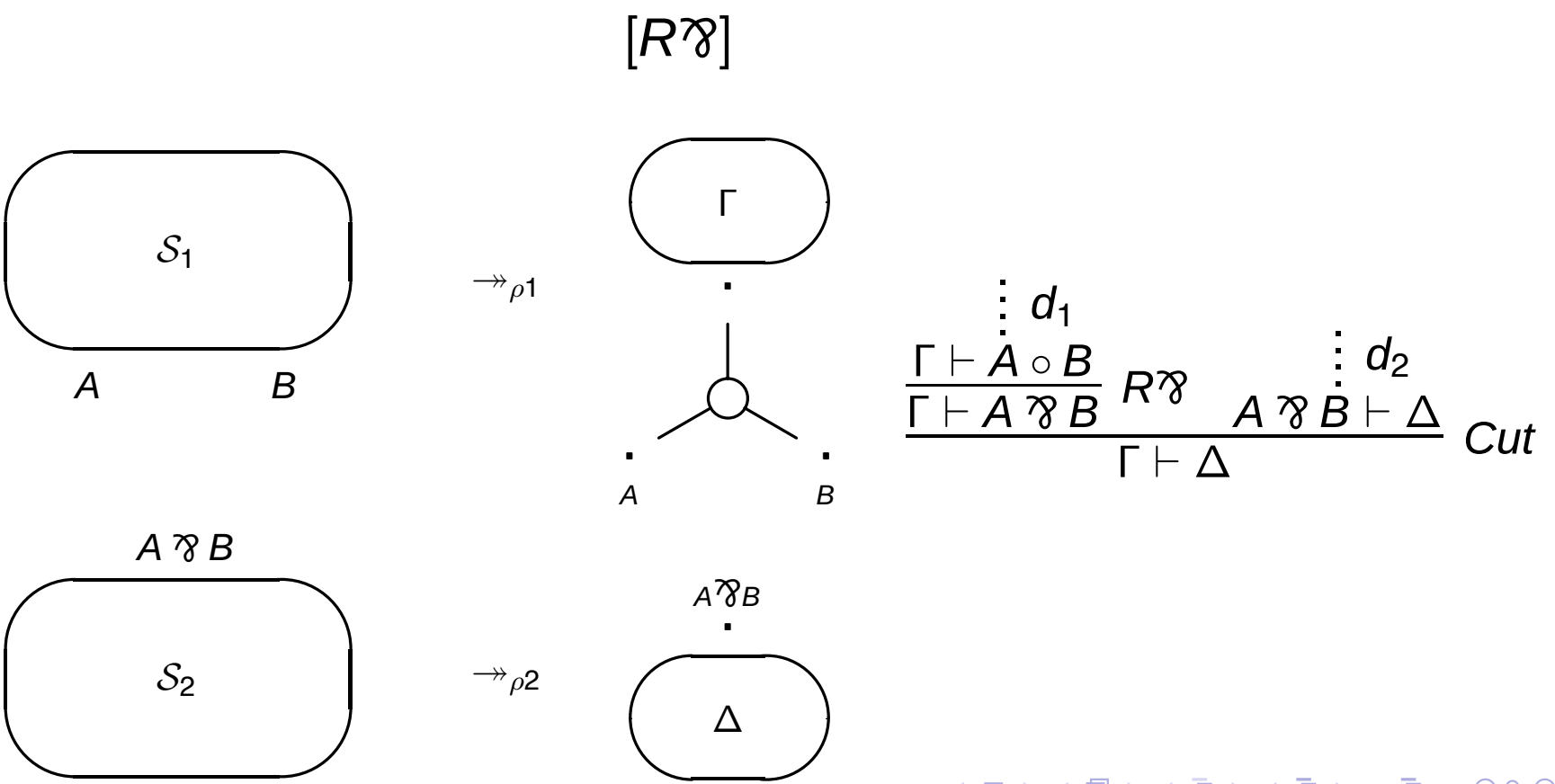
In case $I > 0$, we look at the last conversion.



# Soundness and Completeness

## Completeness: Inductive Case

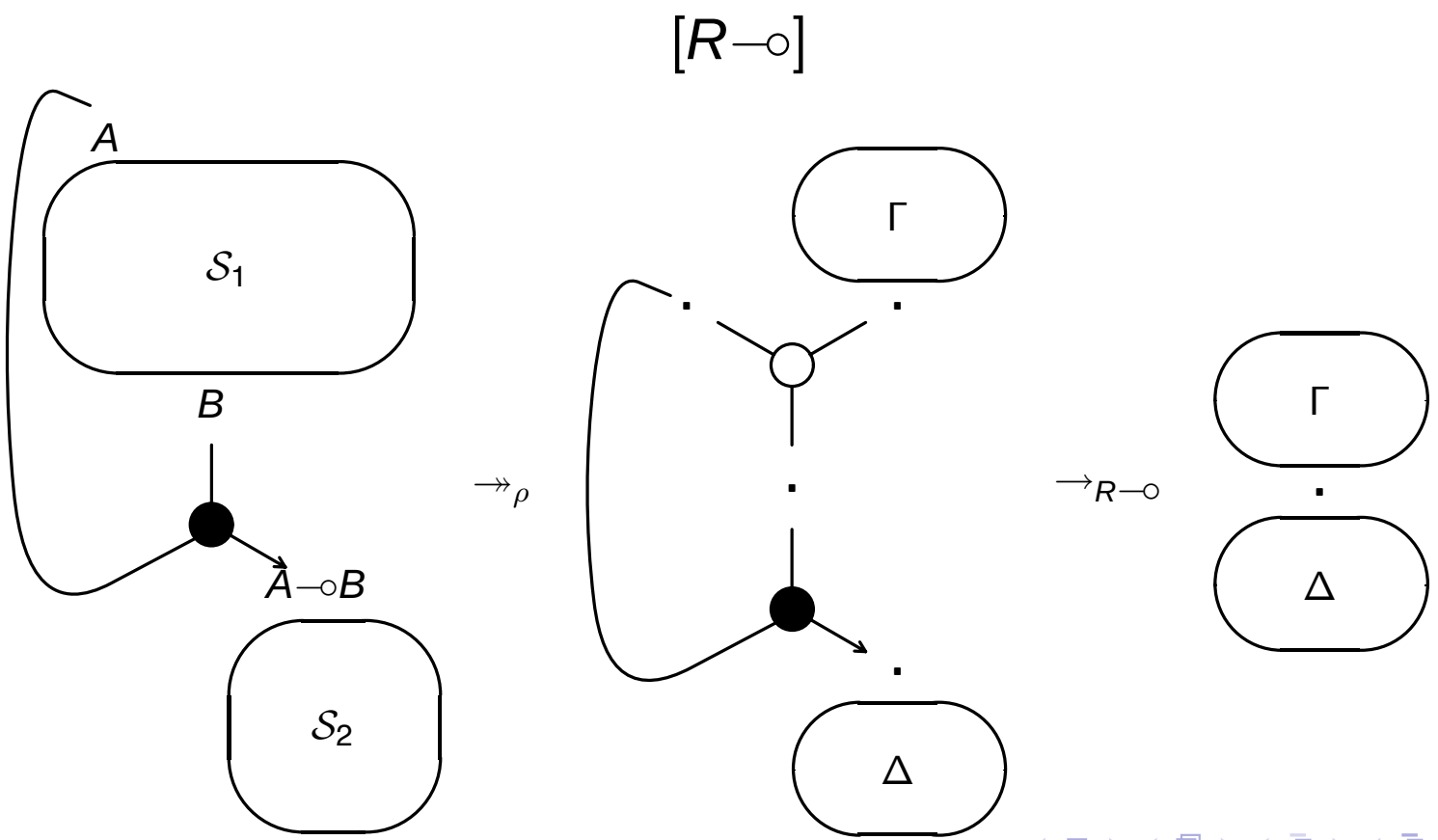
In case $l > 0$, we look at the last conversion.



# Soundness and Completeness

## Completeness: Inductive Case

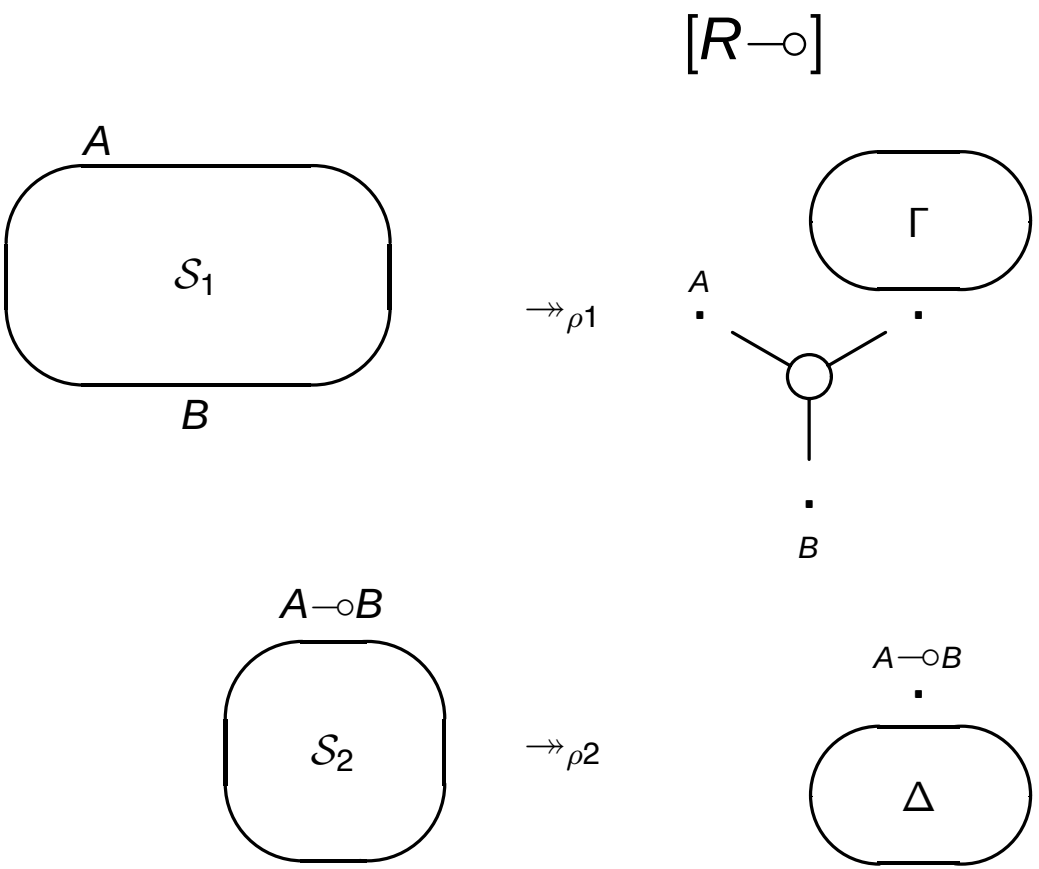
We look at the last conversion.



# Soundness and Completeness

## Completeness: Inductive Case

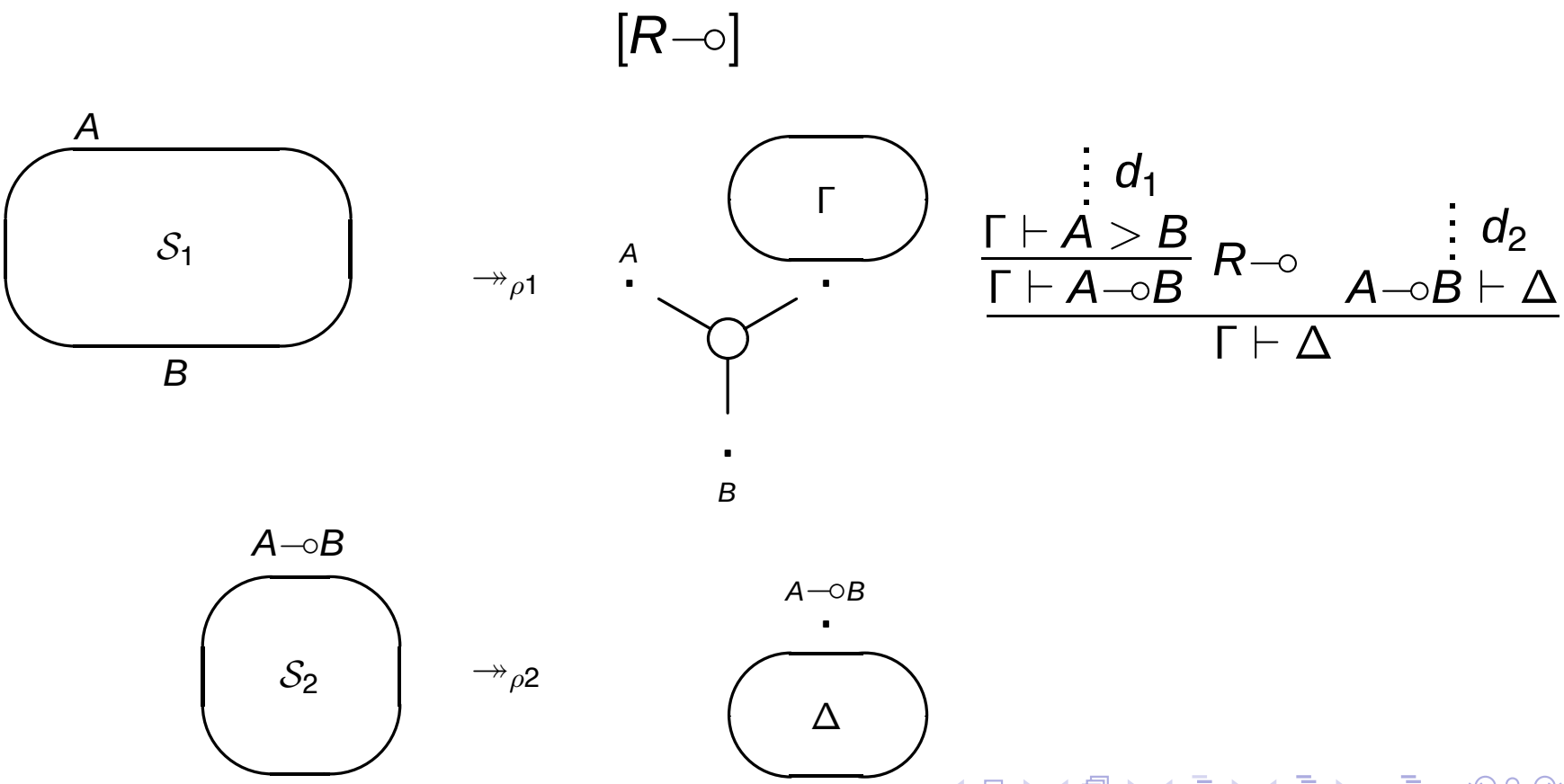
We look at the last conversion.



# Soundness and Completeness

## Completeness: Inductive Case

We look at the last conversion.

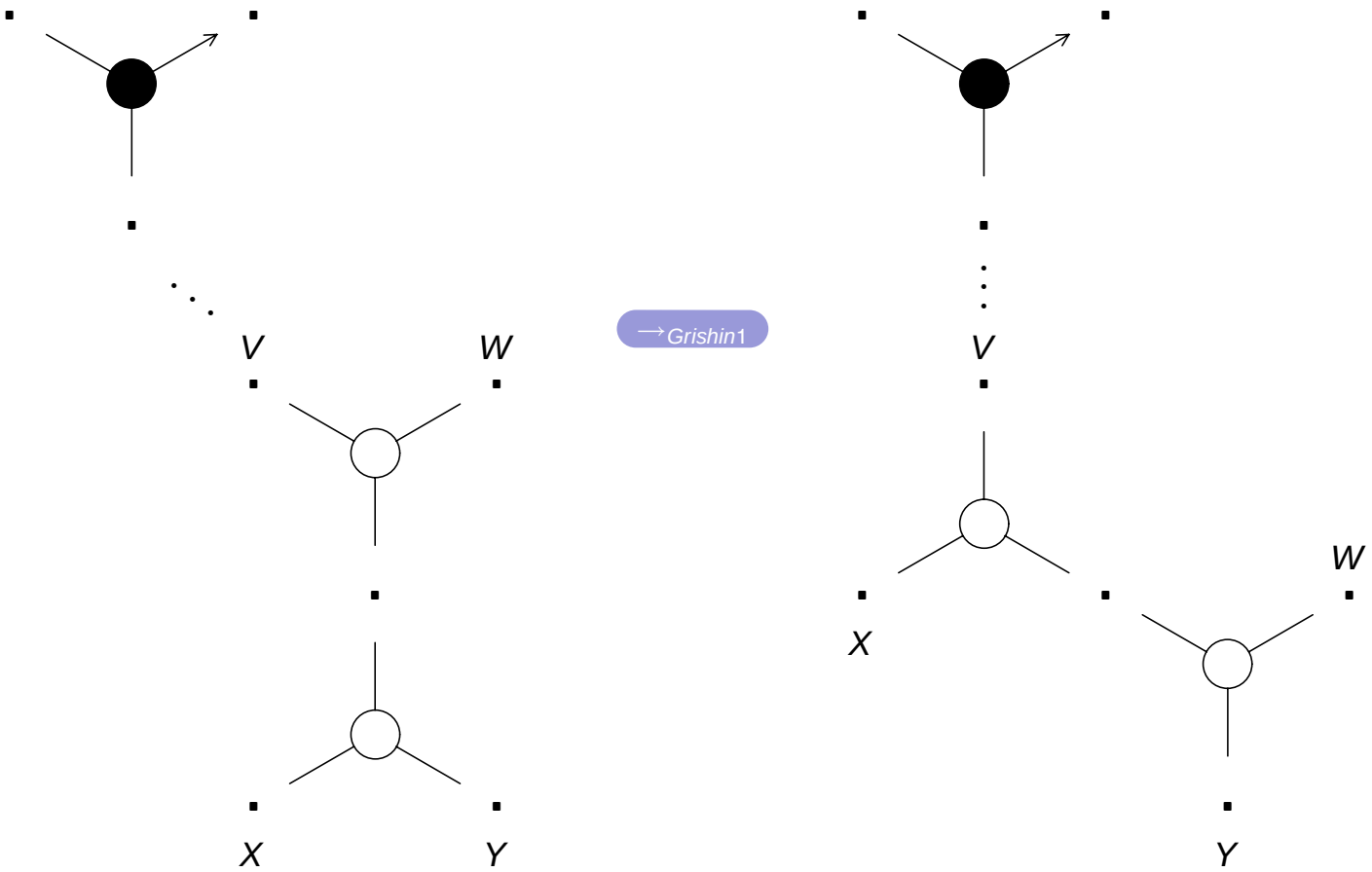


# Applications

- Now that we have the basic results in place, we can reflect on some of the possible applications of the Lambek-Grishin calculus **LG**.
- We have already seen Bernardi & Moortgat's (2007) type assignments for quantifier scope.
- I will give new applications by giving proof nets which can embed *tree adjoining grammars*.
- Be first we will look at the interaction between the contractions and the structural rules.

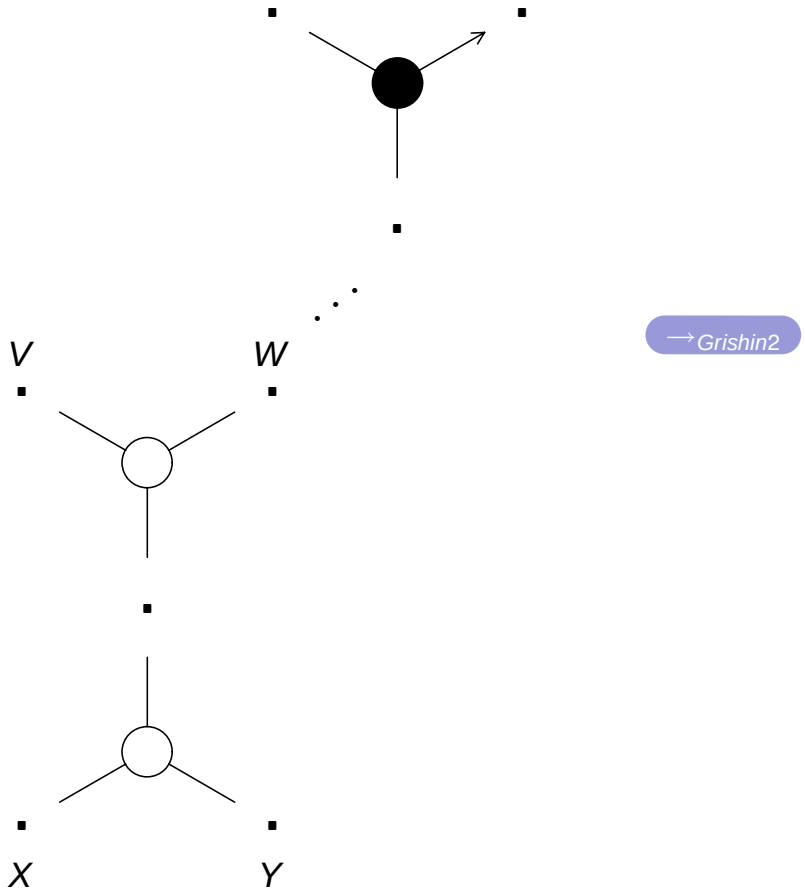
# Grishin Contractions

## Left: Grishin 1

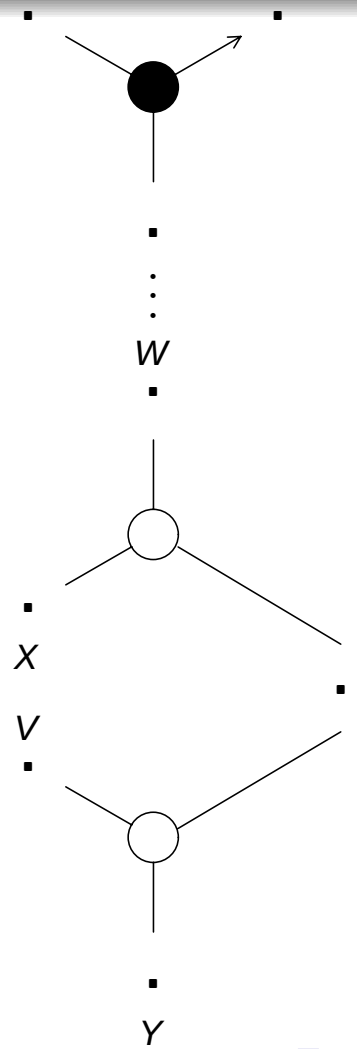


# Grishin Contractions

## Right: Grishin 2

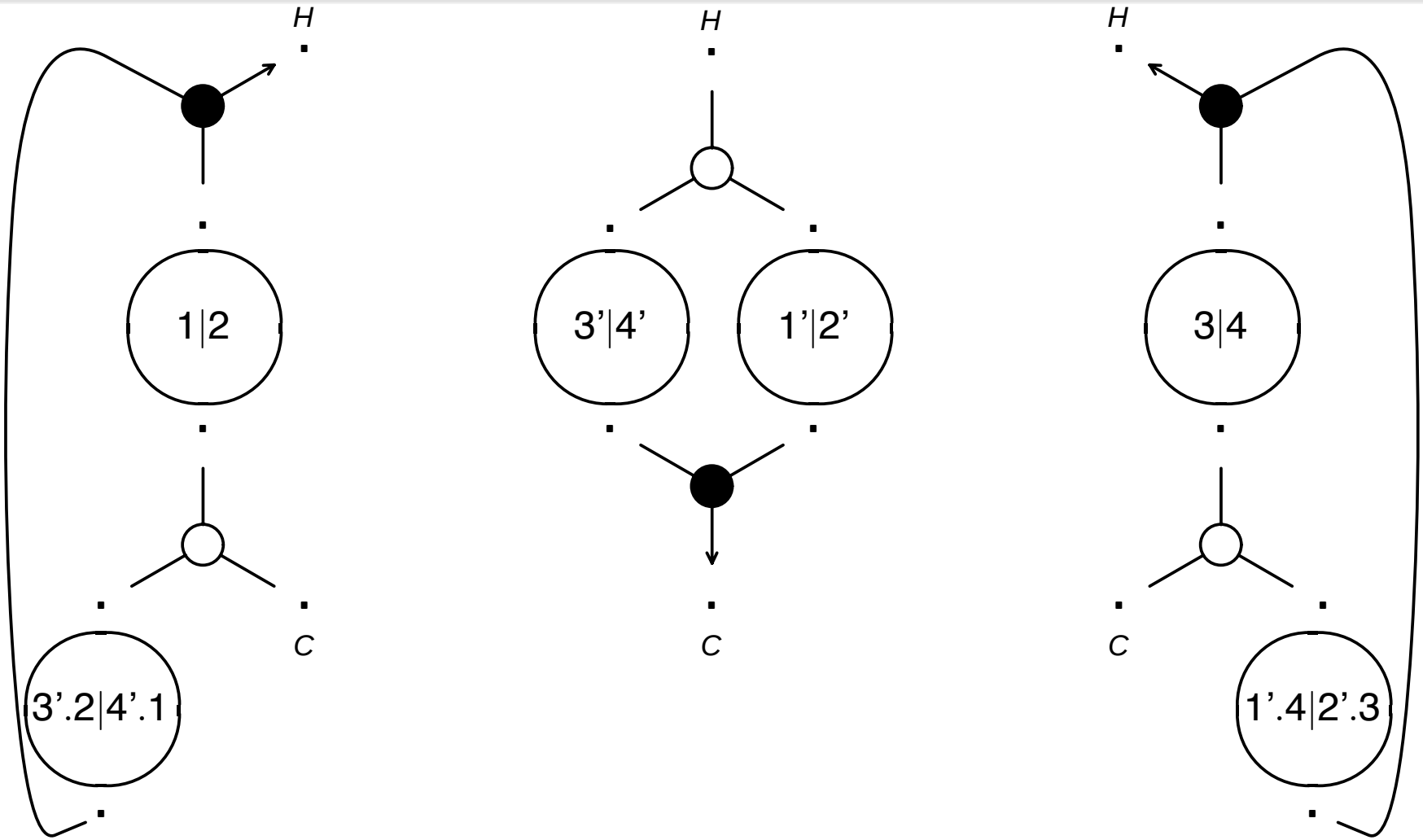



→Grishin2



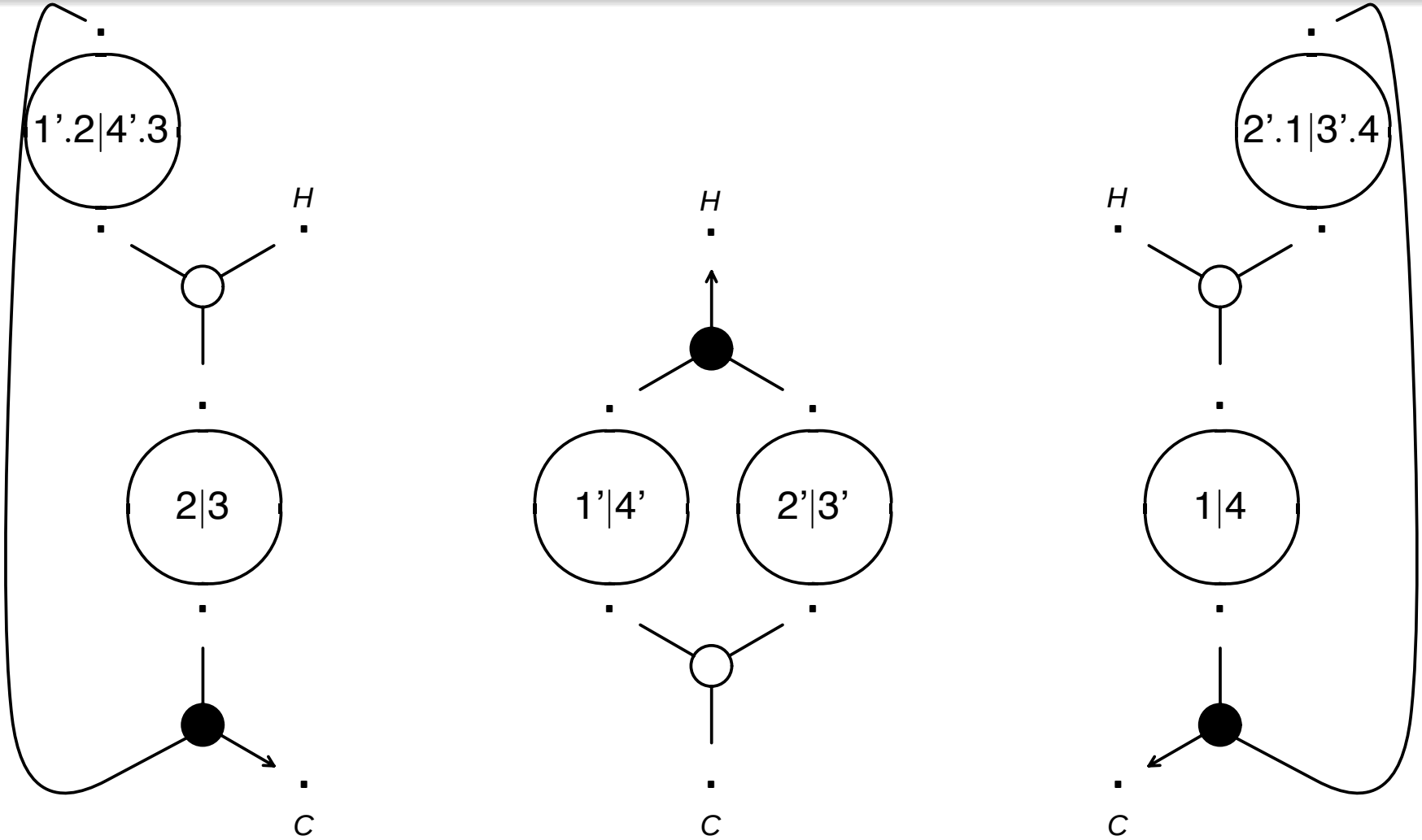
# Contractions for the Lambek-Grishin Calculus

## Dual Residuated Contractions With Grishin Rules



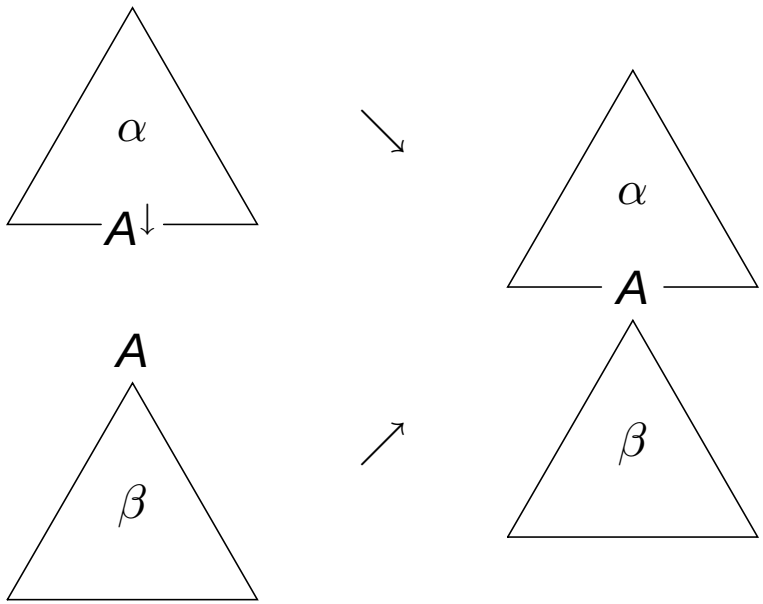
# Contractions for the Lambek-Grishin Calculus

## Residuated Contractions With Grishin Rules



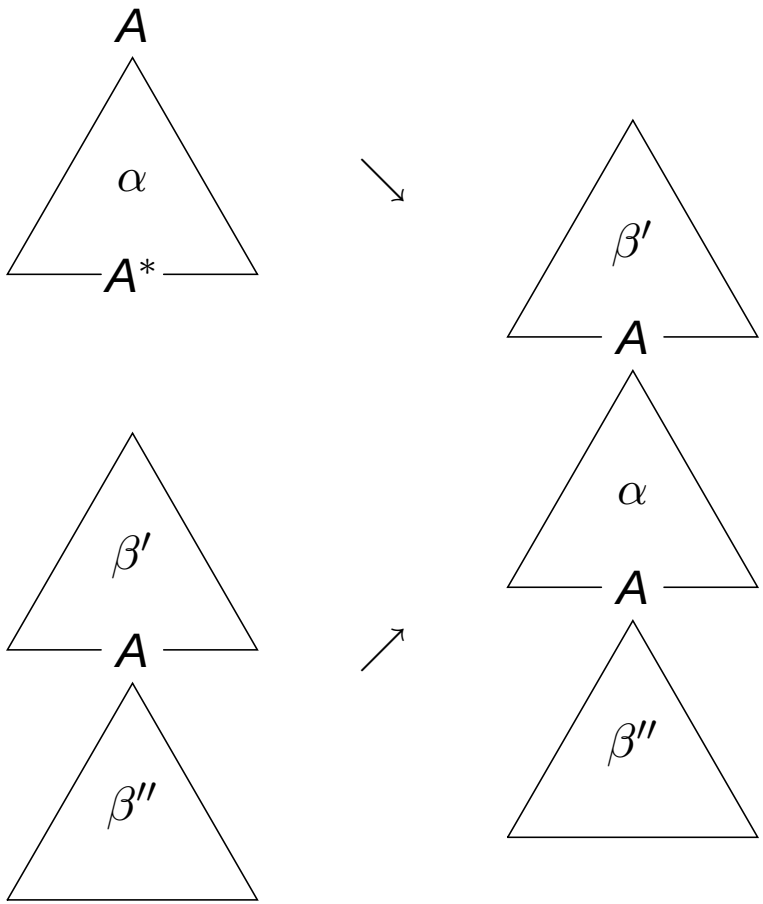
# Tree Adjoining Grammars

## Definition: Substitution



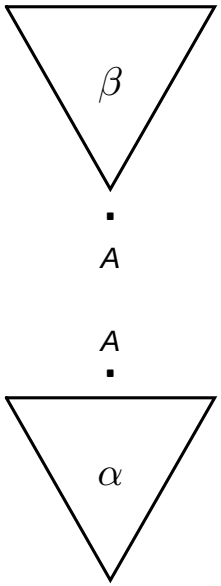
# Tree Adjoining Grammars

## Definition: Adjunction



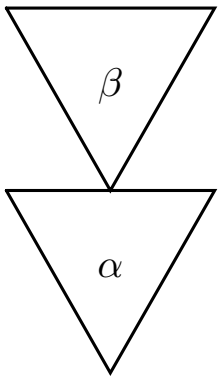
# Tree Adjoining Grammars

## Substitution



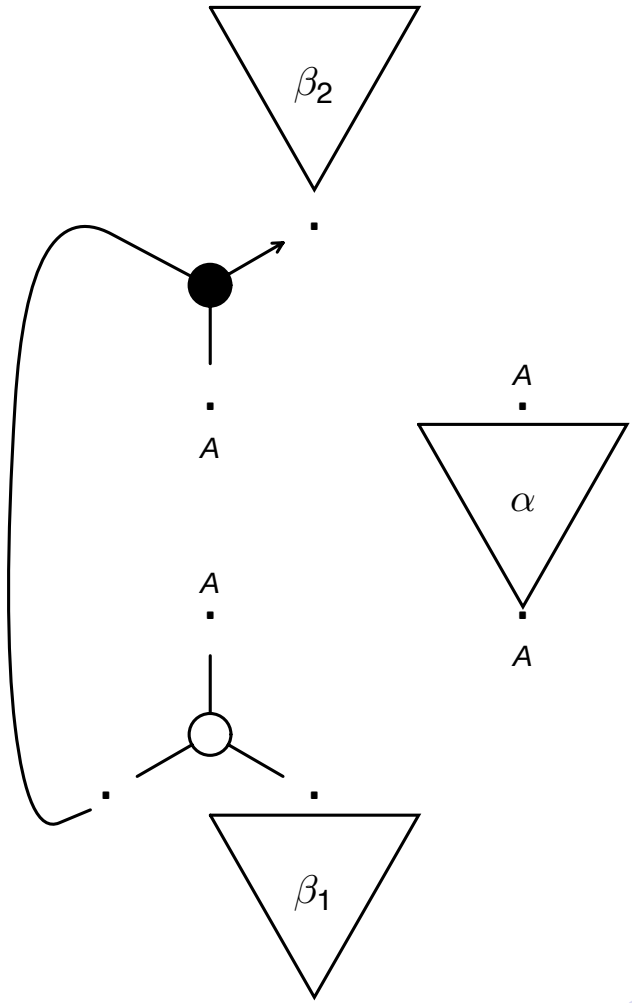
# Tree Adjoining Grammars

## Substitution: Axiom



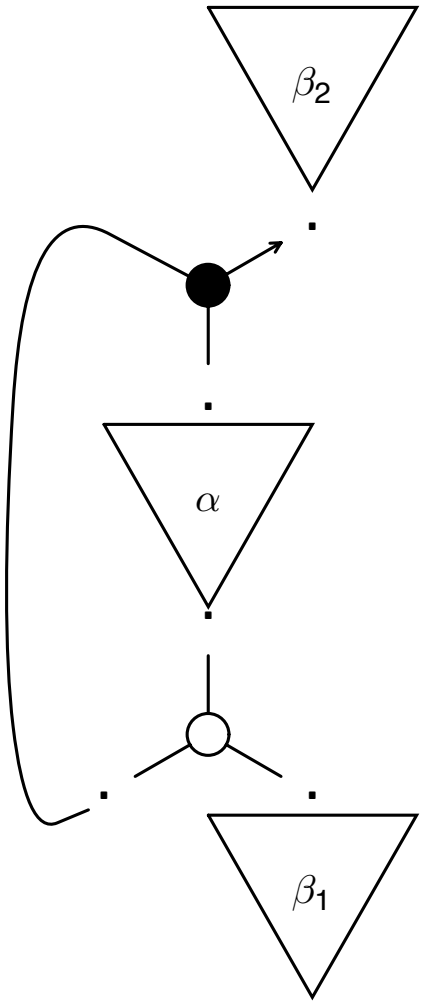
# Tree Adjoining Grammars

## Adjunction



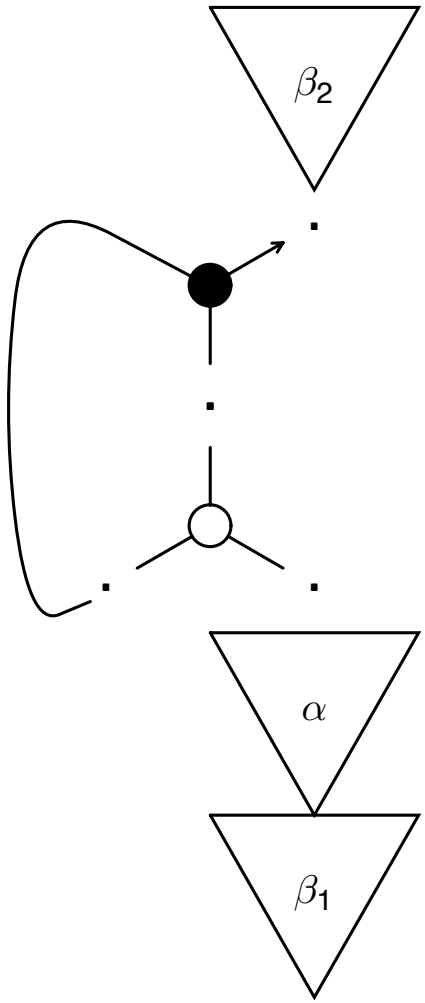
# Tree Adjoining Grammars

## Adjunction: Axiom



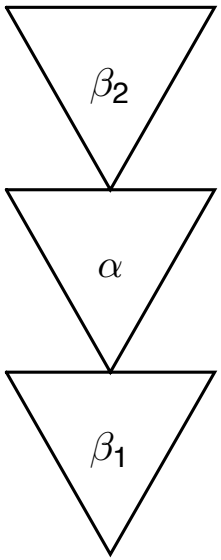
# Tree Adjoining Grammars

## Adjunction: Grishin 1,2



# Tree Adjoining Grammars

## Adjunction: Contraction



# Illustration: Dutch verb clusters

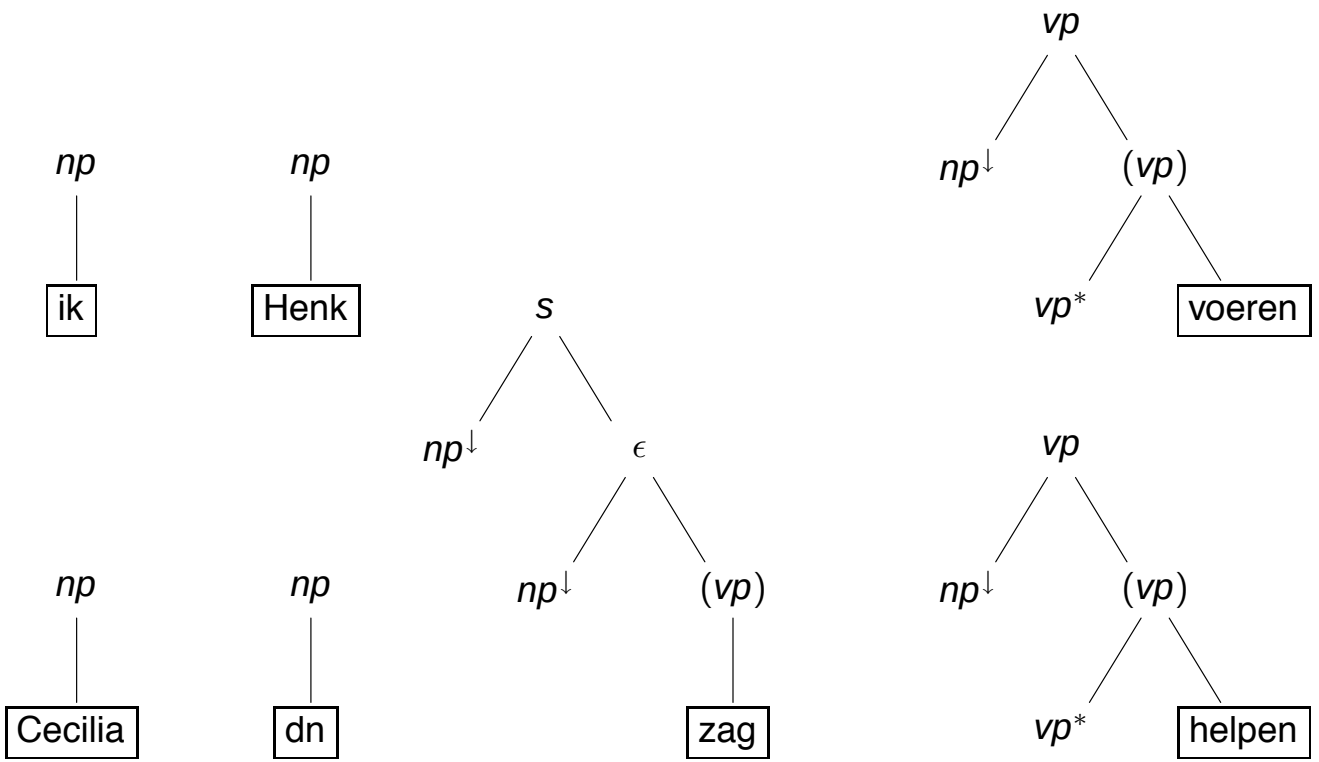
## Data

These are the famous ‘Hippo sentences’. Remark that ‘nijlpaarden’ (hippos) is the object of ‘voeren’ (feed) and that ‘Cecilia’ is the object of ‘zag’ (saw).
We can nest these construction abritrarily deeply, though I always have trouble figuring out which object goes with which verb for longer sentences!

1. (dat) ik Cecilia de nijlpaarden zag voeren.
2. (dat) ik Cecilia Henk de nijlpaarden zag helpen voeren.

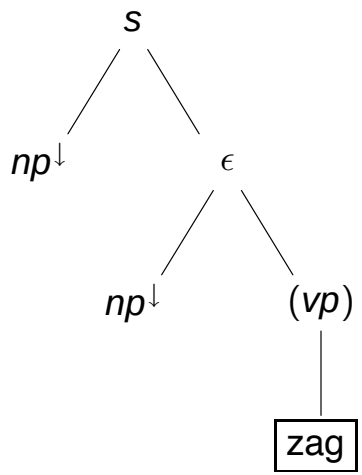
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



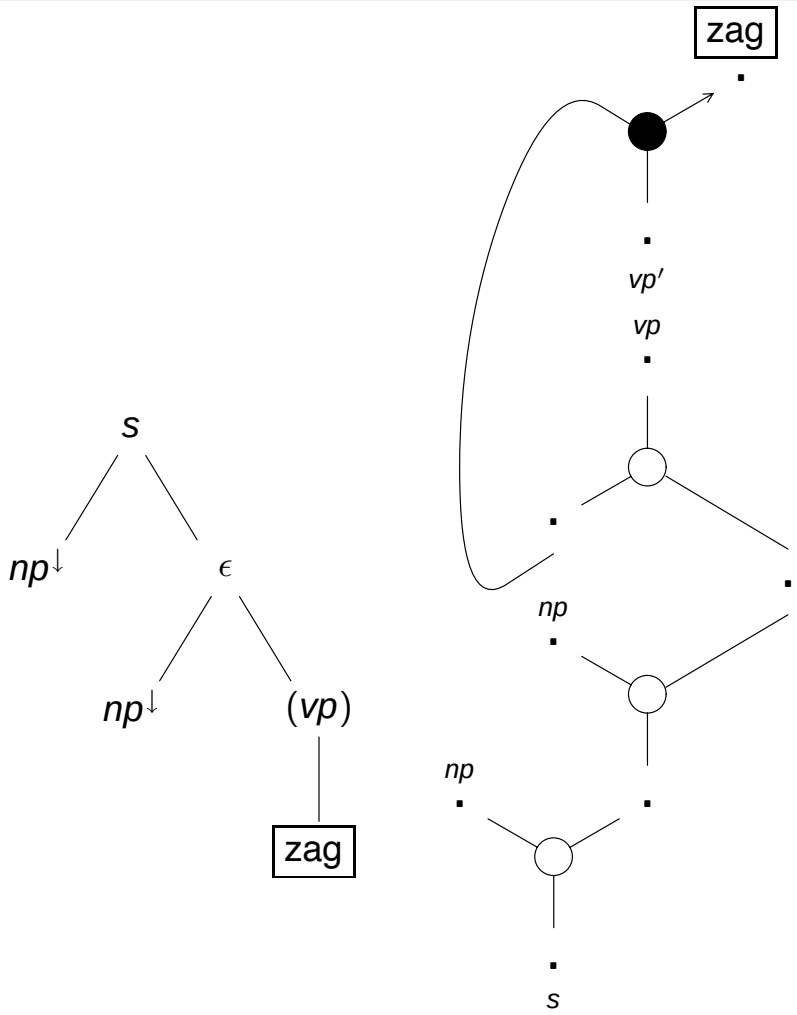
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



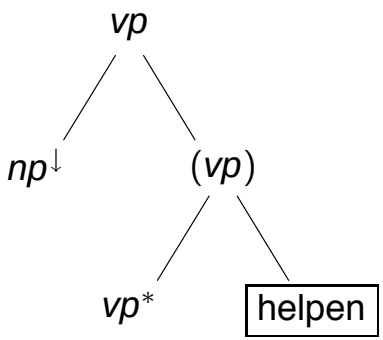
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



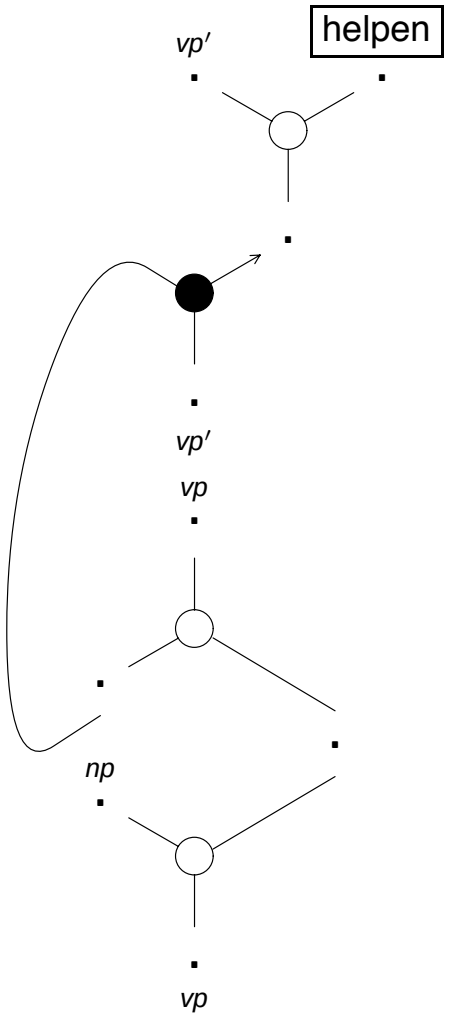
# Illustration: Dutch verb clusters

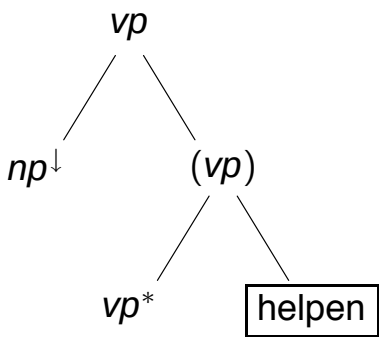
## From Tree Adjoining Grammar to LG



# Illustration: Dutch verb clusters

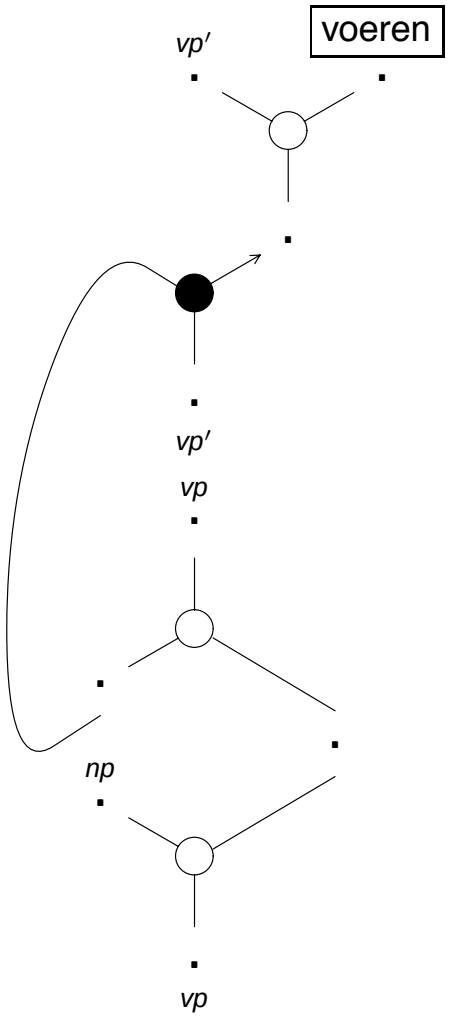
## From Tree Adjoining Grammar to LG

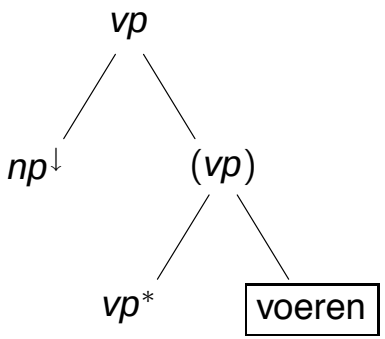


# Illustration: Dutch verb clusters

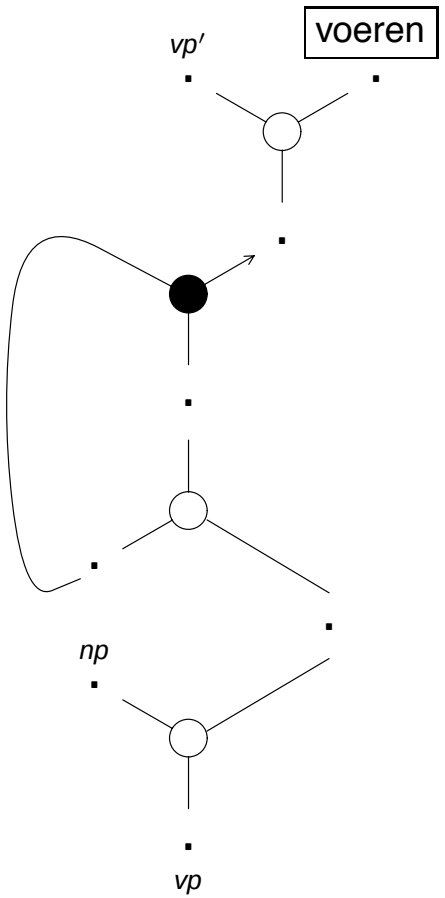
## From Tree Adjoining Grammar to LG

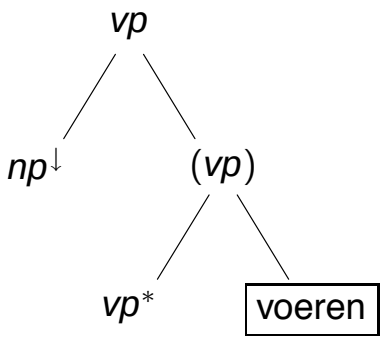


# Illustration: Dutch verb clusters

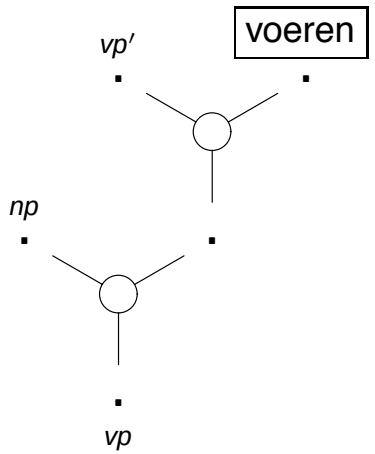
## From Tree Adjoining Grammar to LG

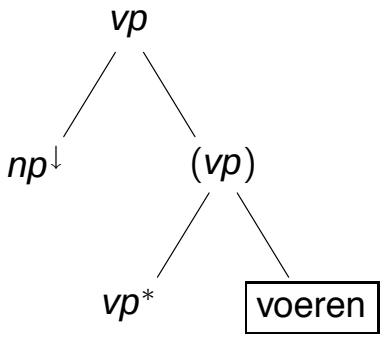


# Illustration: Dutch verb clusters

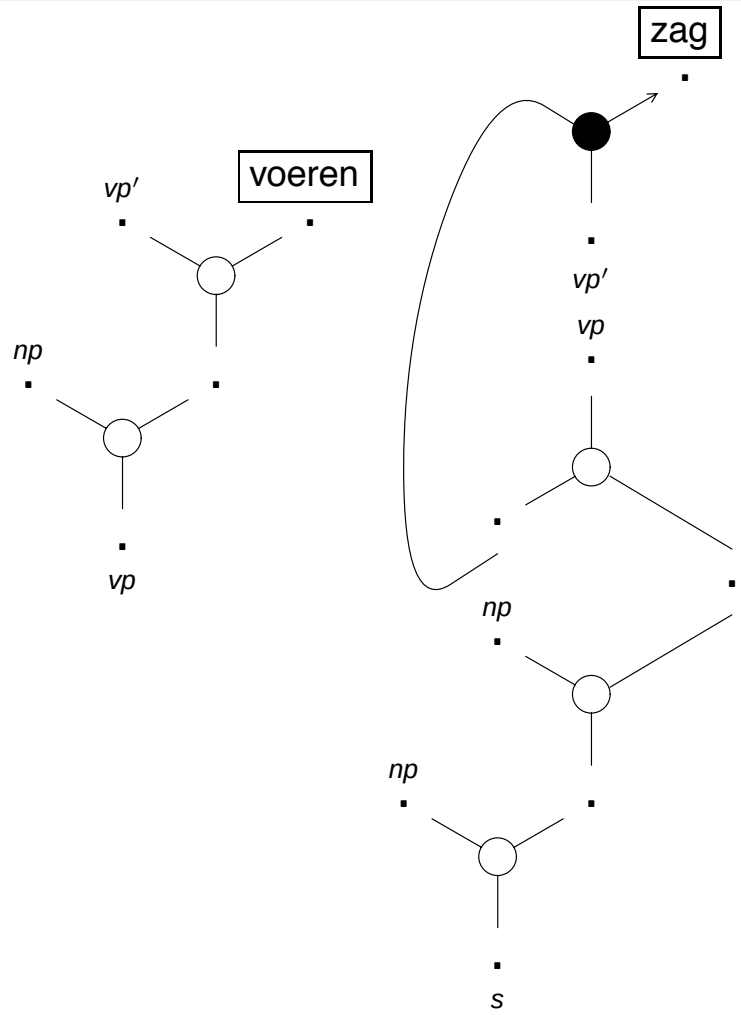
## From Tree Adjoining Grammar to LG



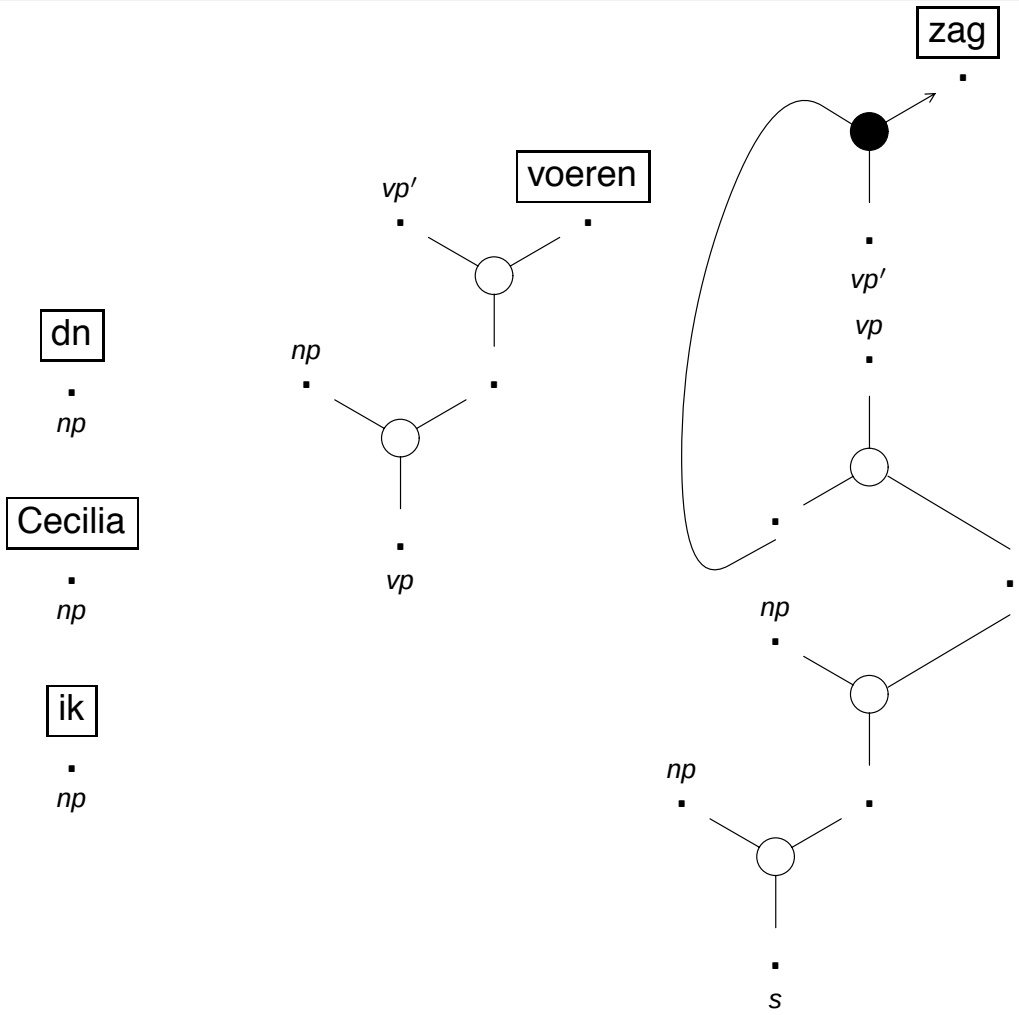
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



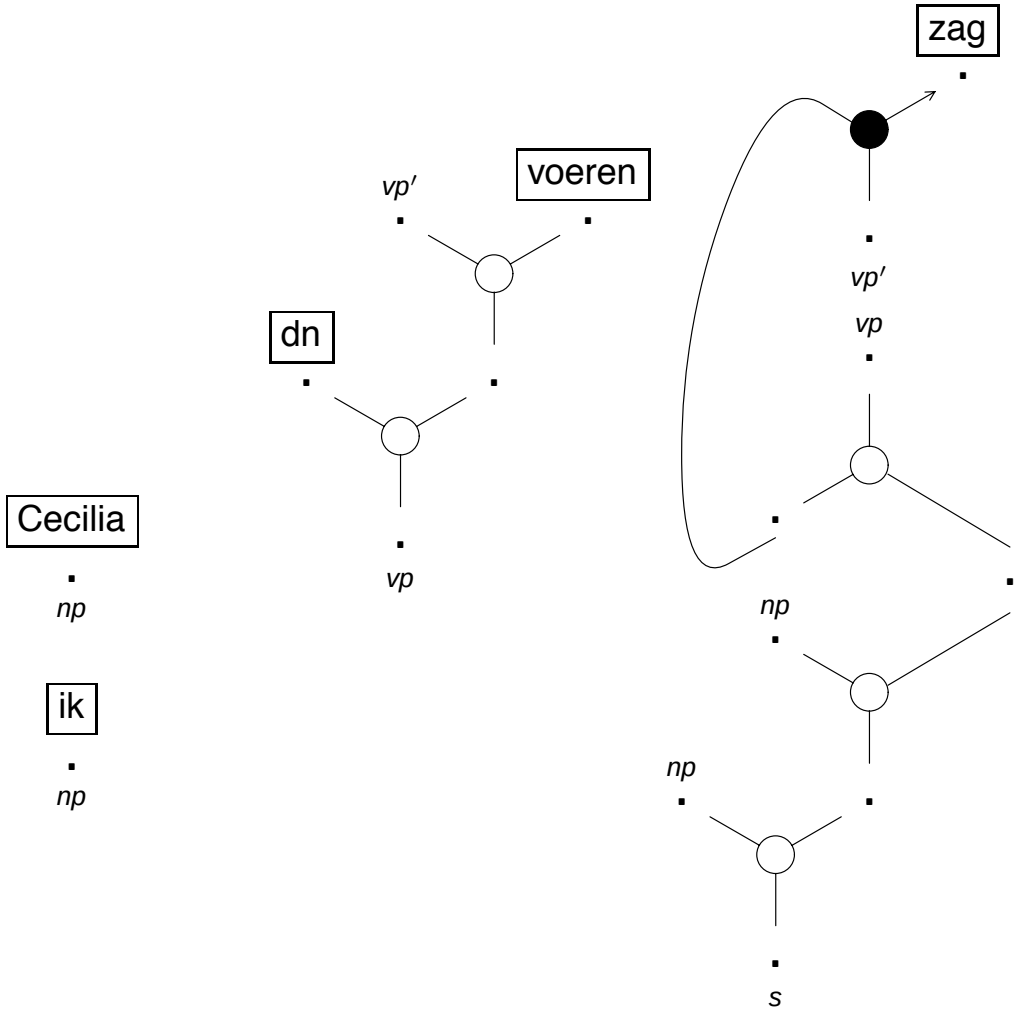
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



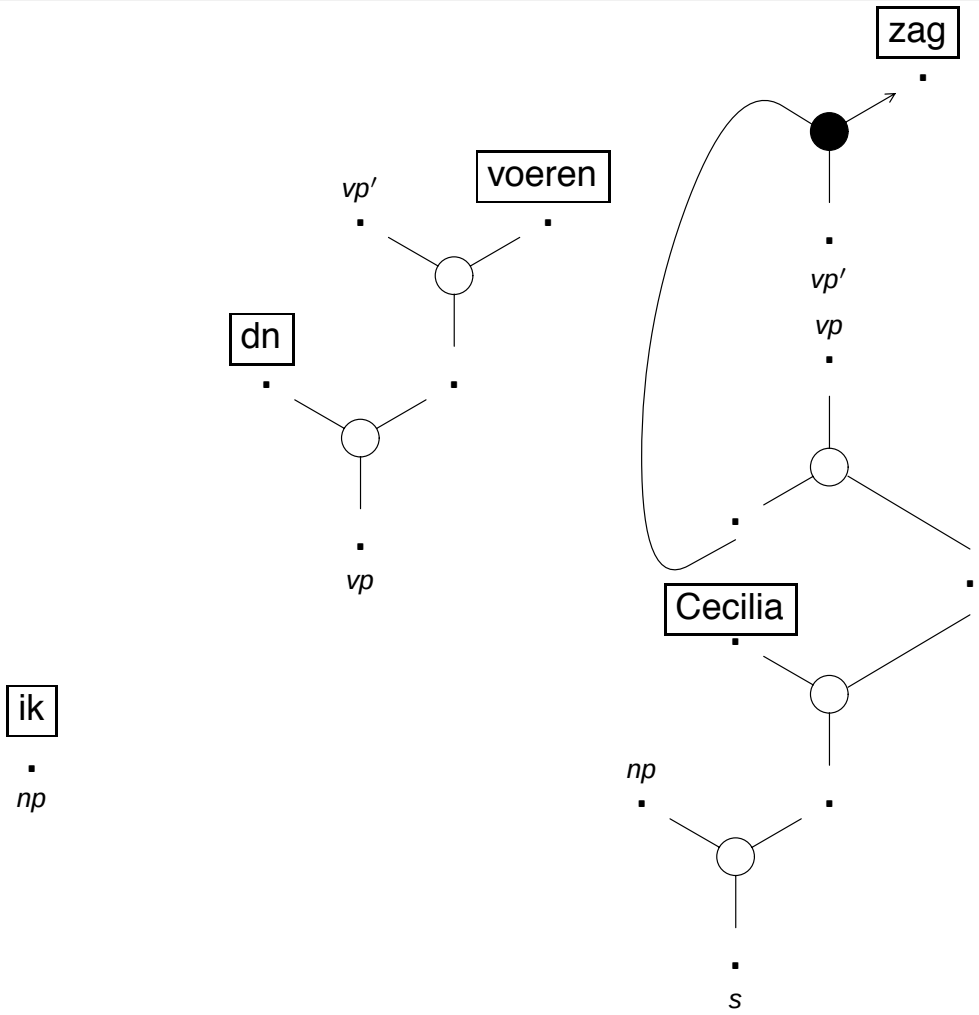
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



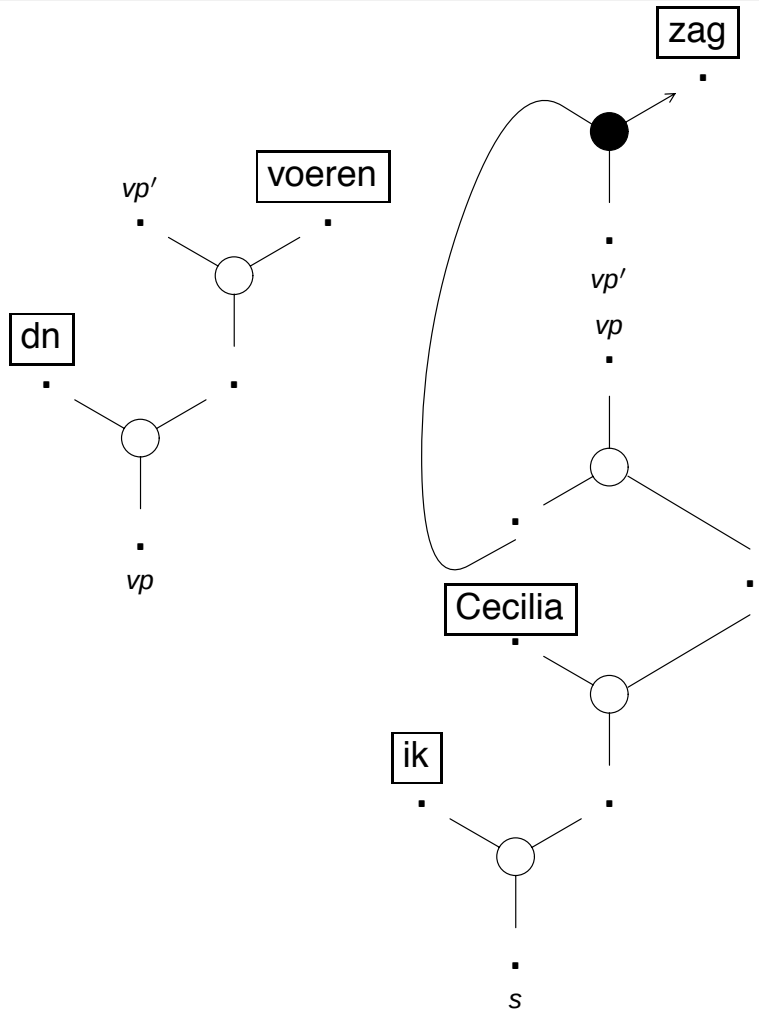
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



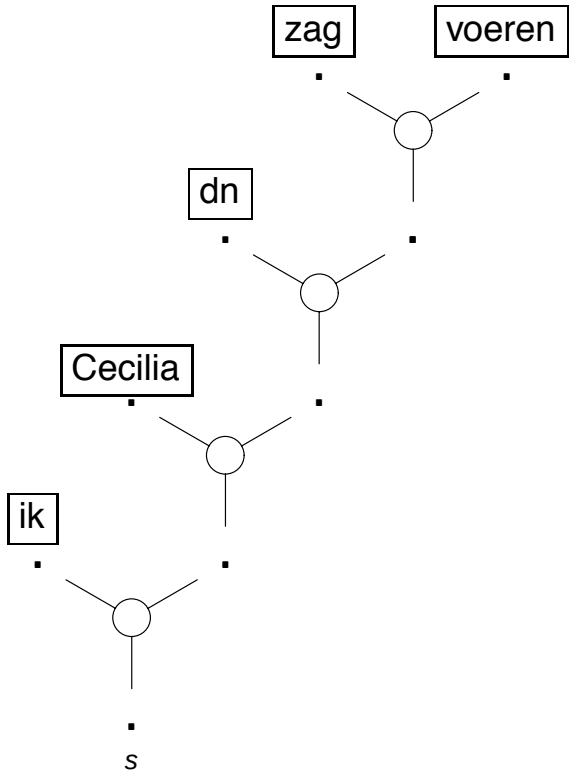
# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



# Illustration: Dutch verb clusters

## From Tree Adjoining Grammar to LG



# Conclusions and Open Questions

## Conclusions

- By exploring the symmetries between hypotheses and conclusions in proof nets, we obtaing proof nets for the Lambek-Grishin calculus LG.
- Soundness and completeness results incorporate these changes without problems.
- An embedding of tree adjoining grammars shows that LG can generate more than just context-free languages.

# Conclusions and Open Questions

## Open Questions

- What is the computational complexity of **LG**?
- What class of languages does **LG** generate?
- What is the influence of the 8 different Grishin structural rules on the complexity/class of languages?

# Appendix
## Sequent Rules

# Sequent Rules

# Sequent Rules
## Identity and Display Rules

$$\frac{}{A \vdash A}\,[Ax] \qquad \frac{\Gamma \vdash A \quad A \vdash \Delta}{\Gamma \vdash \Delta}\,[Cut]$$

$$\frac{\dfrac{\Gamma_1 \vdash \Delta < \Gamma_2}{\Gamma_1 \circ \Gamma_2 \vdash \Delta}\,[rc]}{\Gamma_2 \vdash \Gamma_1 > \Delta}\,[rc] \qquad \frac{\dfrac{\Gamma < \Delta_2 \vdash \Delta_1}{\Gamma \vdash \Delta_1 \circ \Delta_2}\,[drc]}{\Delta_1 > \Gamma \vdash \Delta_2}\,[drc]$$

# Sequent Rules
## Binary Connectives

$$\frac{A \circ B \vdash \Delta}{A \otimes B \vdash \Delta}\,[L\otimes] \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1 \circ \Gamma_2 \vdash A \otimes B}\,[R\otimes]$$

$$\frac{\Gamma \vdash A \quad B \vdash \Delta}{A \multimap B \vdash \Gamma > \Delta}\,[L\multimap] \qquad \frac{\Gamma \vdash A > B}{\Gamma \vdash A \multimap B}\,[R\multimap]$$

$$\frac{A \vdash \Delta \quad \Gamma \vdash B}{A \circ\!\!- B \vdash \Delta < \Gamma}\,[L\circ\!\!-] \qquad \frac{\Gamma \vdash A < B}{\Gamma \vdash A \circ\!\!- B}\,[R\circ\!\!-]$$

$$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{A \parr B \vdash \Delta_1 \circ \Delta_2}\,[L\parr] \qquad \frac{\Gamma \vdash A \circ B}{\Gamma \vdash A \parr B}\,[R\parr]$$

$$\frac{A > B \vdash \Delta}{A \mathbin{\rotatebox{180}{$\multimap$}} B \vdash \Delta}\,[L\mathbin{\rotatebox{180}{$\multimap$}}] \qquad \frac{A \vdash \Delta \quad \Gamma \vdash B}{\Delta > \Gamma \vdash A \mathbin{\rotatebox{180}{$\multimap$}} B}\,[R\mathbin{\rotatebox{180}{$\multimap$}}]$$

$$\frac{A < B \vdash \Delta}{A \mathbin{\rotatebox{180}{$\circ\!\!-$}} B \vdash \Delta}\,[L\mathbin{\rotatebox{180}{$\circ\!\!-$}}] \qquad \frac{\Gamma \vdash A \quad B \vdash \Delta}{\Gamma < \Delta \vdash A \mathbin{\rotatebox{180}{$\circ\!\!-$}} B}\,[R\mathbin{\rotatebox{180}{$\circ\!\!-$}}]$$