Cyclic Multiplicative and Additive Proof Nets with an application to Language Parsing

Roberto Maieli
joint work with V. Michele Abrusci

Dipartimento di Matematica e Fisica
Università degli Studi "Roma Tre"
{abrusci,maieli}@mat.uniroma3.it

The 20th Conference on Formal Grammar
August 8-9, 2015 - UPF Barcelona, Spain

outline

1. motivations
2. proof nets for the CyMALL fragment of LL (updated version!)
3. embedding (extended) Lambek Calculus in to PNs
4. parsing examples of lexical ambiguity via Lambek PNs
5. conclusions and further works

motivations: Parsing with Lambek Calculus

- LC represents the first attempt of **parsing as deduction**, i.e., parsing of natural language by means of a logical system.
- In LC parsing is interpreted as type checking in the form of theorem proving of Gentzen sequents.
- LC parsing presents some syntactical **ambiguity problems**:

  (non canonical proofs) more than one (cut-free) proof for the same sequent conclusion;

  (lexical polymorphism) more than one type associated with a single word;

- we propose a syntax for CyMALL Lambek PNs as a solution for both these problems.

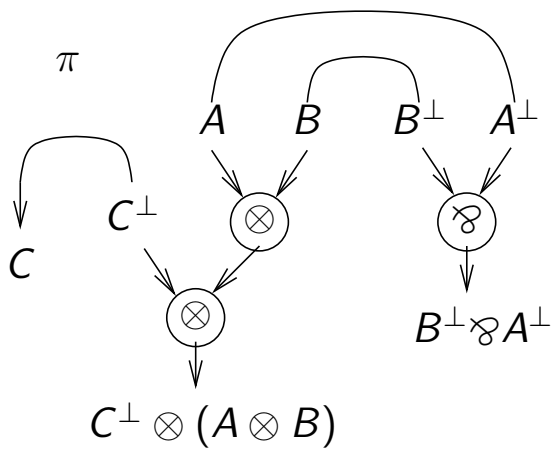
motivations: "why Proof Nets"?

In his seminal article on *linear logic* (LL, 1987), Jean-Yves Girard develops two alternative notations for proofs:

- a **sequential syntax** where proofs are expressed as **derivation trees** in a sequent calculus

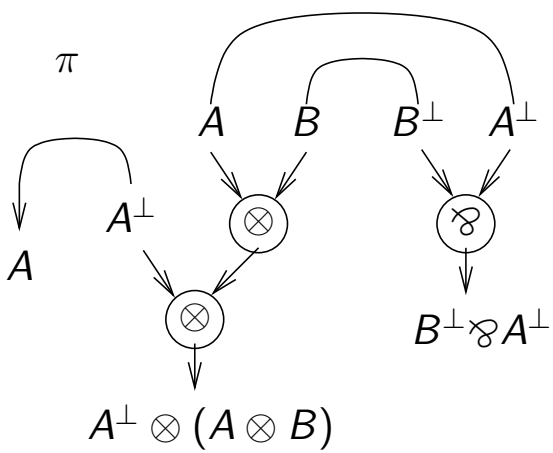
$$\Pi' : \dfrac{\vdash C, C^\perp \qquad \dfrac{\dfrac{\vdash A, A^\perp \qquad \vdash B, B^\perp}{\vdash A \otimes B, B^\perp, A^\perp}\otimes}{\vdash A \otimes B, B^\perp \parr A^\perp}\parr}{\vdash C, C^\perp \otimes (A \otimes B), B^\perp \parr A^\perp}\otimes \qquad \Pi'' : \dfrac{\dfrac{\vdash C, C^\perp \qquad \dfrac{\vdash A, A^\perp \qquad \vdash B, B^\perp}{\vdash A \otimes B, B^\perp, A^\perp}\otimes}{\vdash C, C^\perp \otimes (A \otimes B), B^\perp, A^\perp}\otimes}{\vdash C, C^\perp \otimes (A \otimes B), B^\perp \parr A^\perp}\parr$$

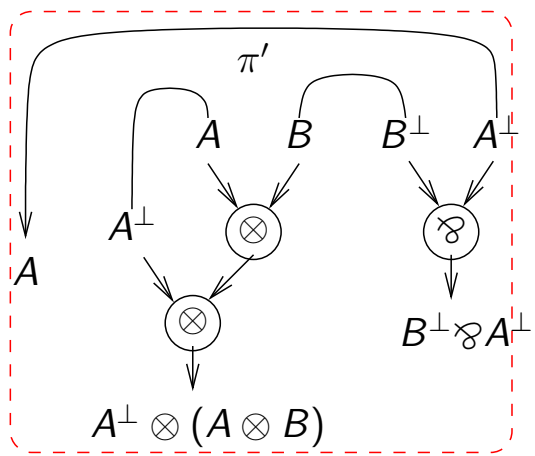
- a **parallel syntax** where proofs are expressed as **bipartite graphs** called **proof-nets**



motivations: "why Proof Nets?" (continues)

- PNs represent demonstrations in a "geometric" (="non inductive") way, abstracting away from the bureaucracy of sequent proofs.
- PNs quotient classes of derivations that are equivalent up to some irrelevant permutations of inference rules instances.
  - while a derivation tree defines a unique proof-net, a PN represents several derivation trees, each derivation tree witnessing a particular order of the PN sequentialization;
  - a PN requires to separate "real proofs" (*proof-nets*) from "proof alike" (*proof-structures*) via **correctness criteria**;
  - correctness criteria reveal the "geometric" essence of the logic, beyond its "grammatical" presentation as a sequent calculus.





motivations: "why CyMALL?"

- **natural languages:** lexical ambiguity (type polymorphism)

  Sollozzo believes Vito. (1)

  Sollozzo believes Vito trusts him. (2)

- **formal languages:** what's the recognizing power of CyMALL?
  - **CyMLL:** Mati Pentus proved (LICS, 1993) the *Chomsky conjecture*, i.e., the languages recognized by basic Lambek Categorial Grammars are precisely the Context-free ones.
  - **CyMALL:** open question.

  EXAMPLE: assume two CFLs as below

  $$(a^* b^n c^n) \quad \text{and} \quad (a^n b^n c^*)$$

  then, their *intersection* is not a CFL

  $$(a^* b^n c^n) \cap (a^n b^n c^*) = (a^n b^n c^n)$$

The CyMALL fragment: sequent calculus

- Assume **literals** $a, a^\perp, b, b^\perp, \ldots$ with a polarity: *positive* for atoms, $a, b, \ldots$ and *negative* $a^\perp, b^\perp \ldots$ for their duals.
- A **formula** is built from literals by means of:
  - *negative connectives* : ∇ ("par") and $\&$ ("with")
  - *positive connectives* : $\otimes$ ("tensor") and $\oplus$ ("plus").
- **De Morgan laws**: $(A \otimes B)^\perp = B^\perp \nabla A^\perp$ and $(A \nabla B)^\perp = B^\perp \otimes A^\perp$
  $(A \& B)^\perp = B^\perp \oplus A^\perp$ and $(A \oplus B)^\perp = B^\perp \& A^\perp$
- A **CyMLL proof** is any derivation tree built by the following inference rules where sequents Γ, Δ are lists of formulas occurrences endowed with a **total cyclic order** (or **cyclic permutation**):

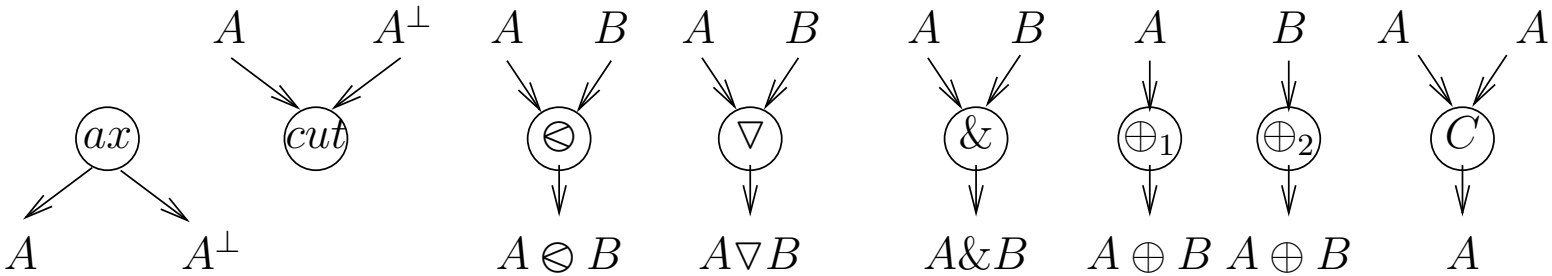
identity:
$$\frac{}{\vdash A, A^\perp}\, AX \qquad \frac{\vdash \Gamma, A \qquad A^\perp \Delta}{\vdash \Gamma, \Delta}\, cut$$

multiplicatives:
$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta} \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \nabla B} \nabla$$

additives:
$$\frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \& \qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus_i A_2} \oplus_{i=1,2}$$

concrete proof structure (CPS)

A *CyMALL proof-structure (PS)* is an oriented graph π, in which edges (resp., nodes) are labeled by formulas (resp., by connectives or contraction C) and built by juxtaposing the below (bipartite) graphs, called **links**, in which incoming edges are called *premises* while outgoing edges are called *conclusions* of the link:



In a PS π:

- each premise of a link must be conclusion of exactly one link of π;
- each conclusion of a link must be premise of at most one link of π.

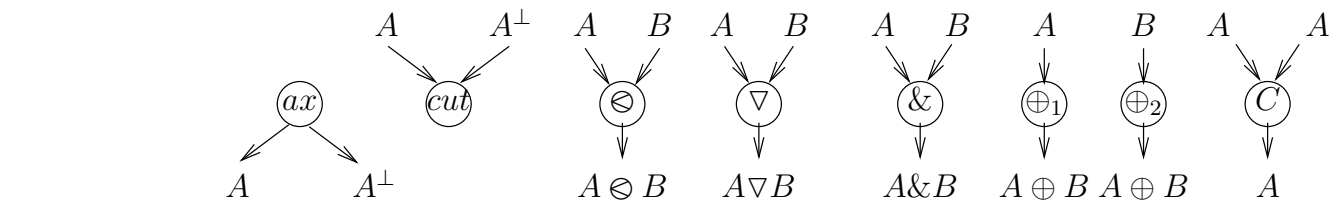
A *conclusion of* π is any outgoing edge that is not premises of any link.

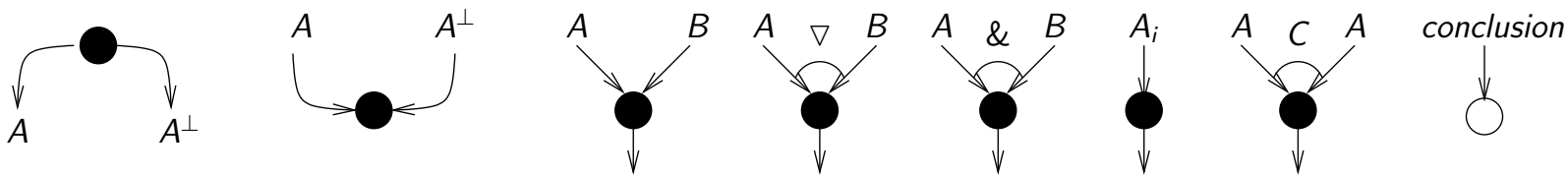
In the following we characterize those CyMLL PSs that are images of CyMLL proofs: these are called **correct proof structures** or **proof nets**

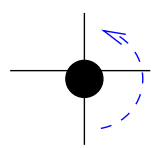
abstract proof structure (APS)

An *APS* is a(n oriented) graph π equipped with a set $\mathcal{C}(\pi)$ of **pairs** of coincident edges graphically denoted by a crossing *arc* close to the base and labeled by a *type*: $\parr$, & or C the *additive contraction*.
Edges are labeled by CyMALL formulas. **Nodes** are displayed as bullets ($\bullet$) except the handling ones (i.e., conclusions) displayed as circles ($\circ$); all edges incident to a node have an anti-clockwise order.

ABSTRACTING FROM CPS TO APS : $\pi \mapsto \pi^{ab}$



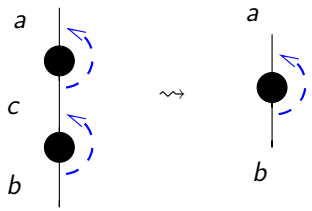




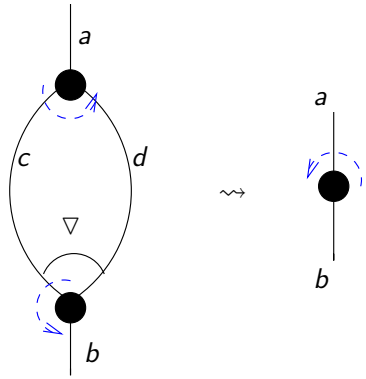
ANTI-CLOCKWISE ORIENTATION

Retraction of APS

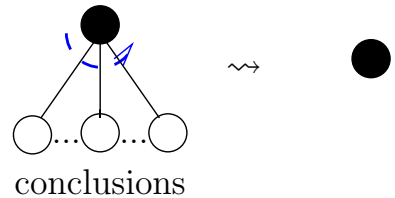
Given an APS π, a **retraction step** is a replacement (also, *deformation* or *rewriting*) of a subgraph S (called, *redex graph*) of π with a new graph S' (called, *reductum graph*), leading to an APS π' according to one of the following rules (preserving the anti-clockwise orientation):



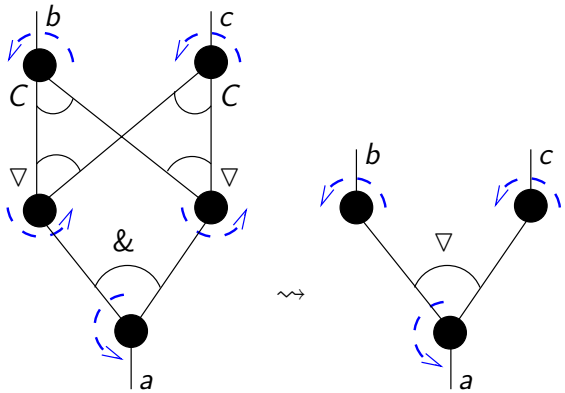
structural



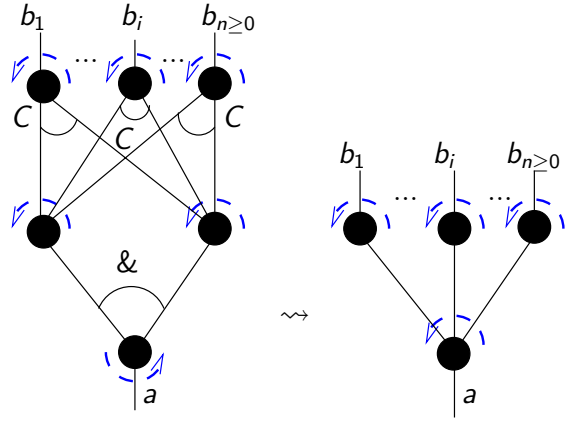
multiplicative



conclusions



distributivity



semi-distributivity

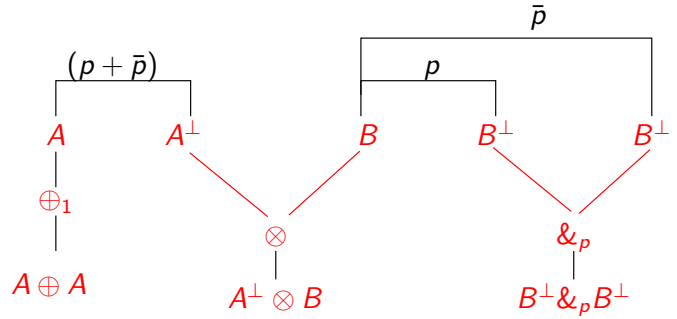
Correctness of PS by retraction

- an APS π is **retractile** if there exists an APS $\pi' \neq \pi$ s.t. $\pi \rightsquigarrow^* \pi'$ after a non empty sequence of retraction steps;
- a non retractile APS is called **terminal**; a terminal APS consisting of a single node is called **collapsed** (or *elementary*);
- a PS π is **correct**, or it is a **proof net** (PN), when the corresponding APS π^{ab} collapses (i.e., it retracts in to a collapsed node);
- the retraction system is convergent (terminating and confluent);
- (Retraction of PNs is preserved by cut-reduction (omitted))

Comparing retractile PNs wr.t. other syntaxes

Each retractile PN is a Girard PN (1996) by adding monomial weights

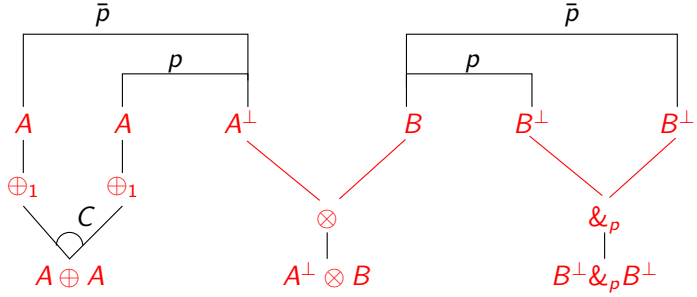
this is also a PN for
Hughes-van Glabbeek (2003)



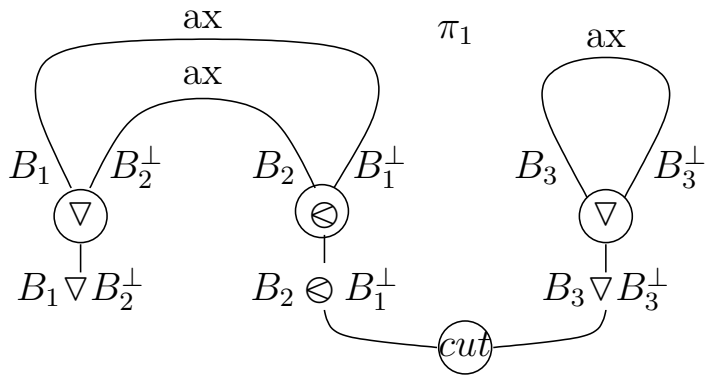
1. Fact: if π is a Girard PN, then each boolean evaluation induces a *slice* that is a multiplicative PN; moreover, fixed a slice, each DR-switching is a **seaweed** (a cyclic order) on the conclusion of π.
2. Lemma: all seaweeds induce the *same order* on the conclusions.
3. Corollary: retractile PNs are sequentializable.

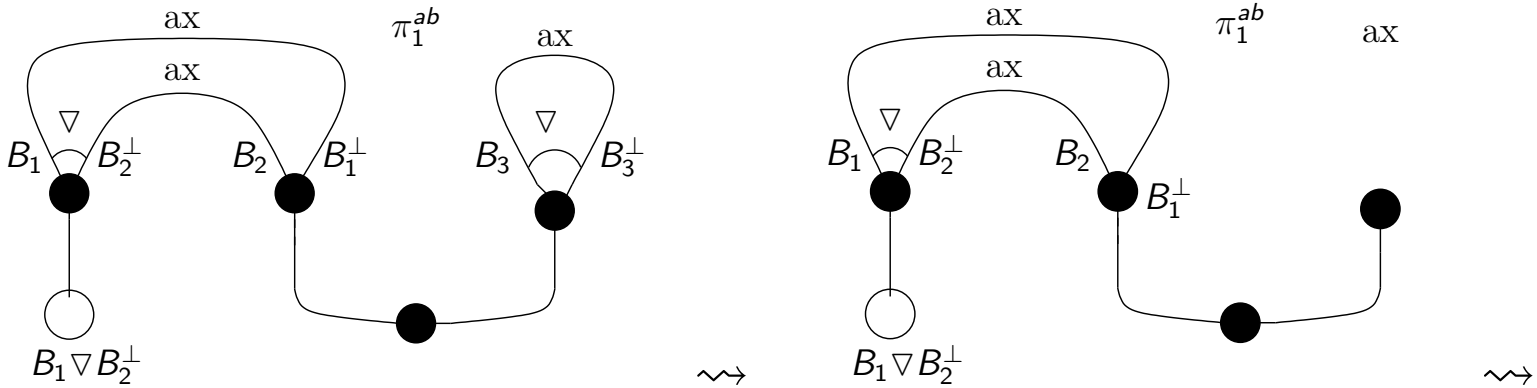
Vice-versa, there exist some Girard PNs that are not retractile:

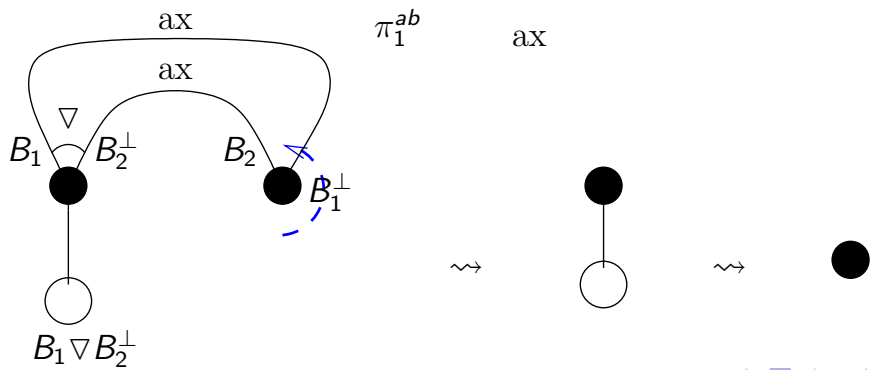
this is neither a PN for
Hughes-van Glabbeek (2003)



CyMLL examples: retractile proof structure

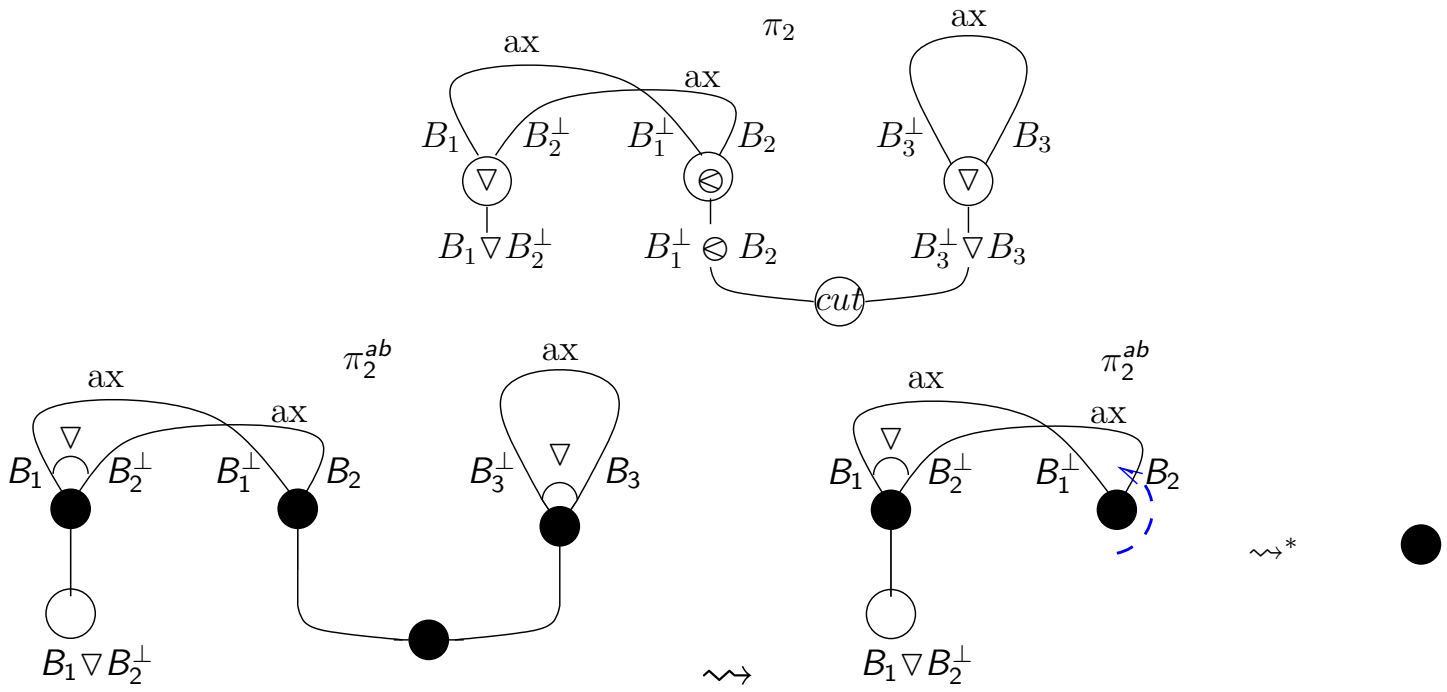




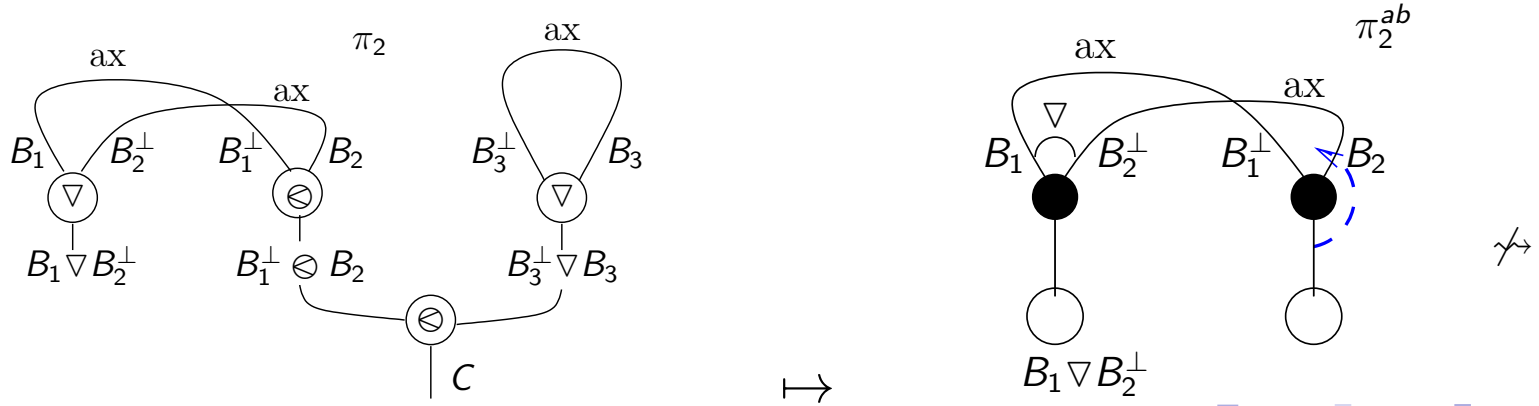


CyMLL examples: retractile proof structure

observe that the following (non-planar) proof structure π_2 is correct



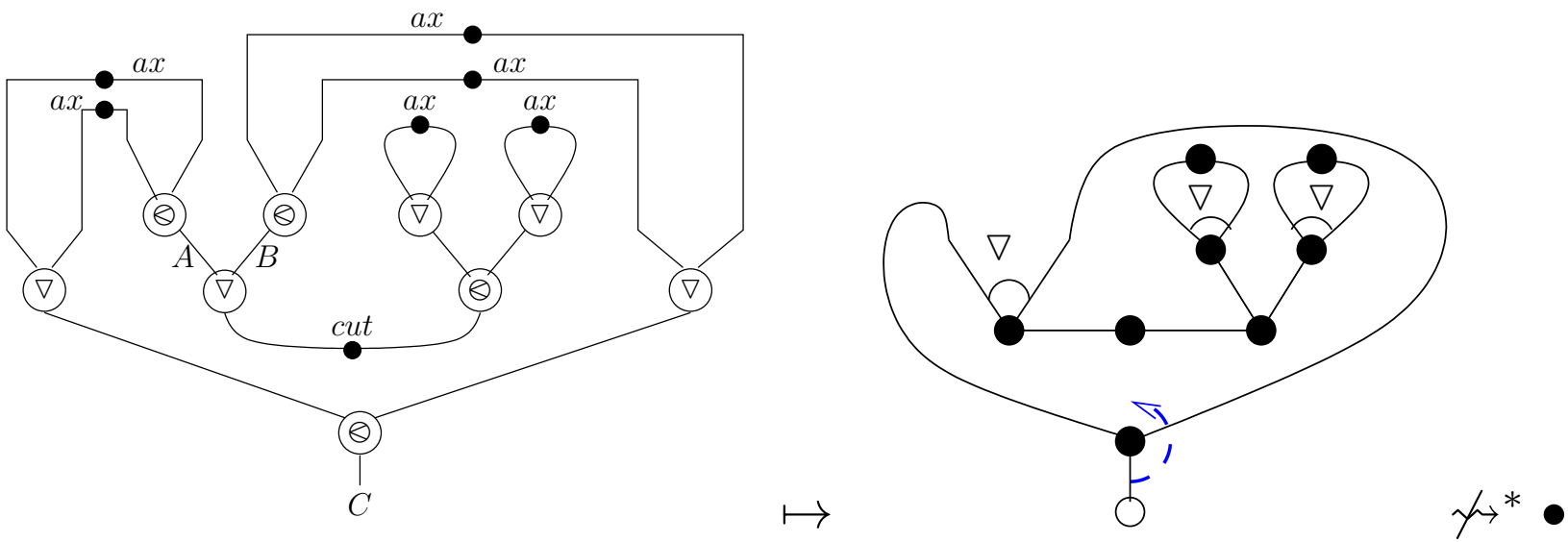
Replacing the cut-link with a tensor $\otimes$-link, leads to a non correct PS!



CyMLL examples: Melliès proof structure

in spite of what happens in the commutative MLL case, the presence of cut links is "quite tricky" in the non-commutative case, since cut links are not equivalent, from a topological point of view, to tensor links: these latter make appear new conclusions that may disrupt the original order.

According to P.-A. Mellies (2003) the below **PS is not correct** by our **retraction correctness criterion**! that is not the case with other criteria (eg Abrusci-Ruet 2000).

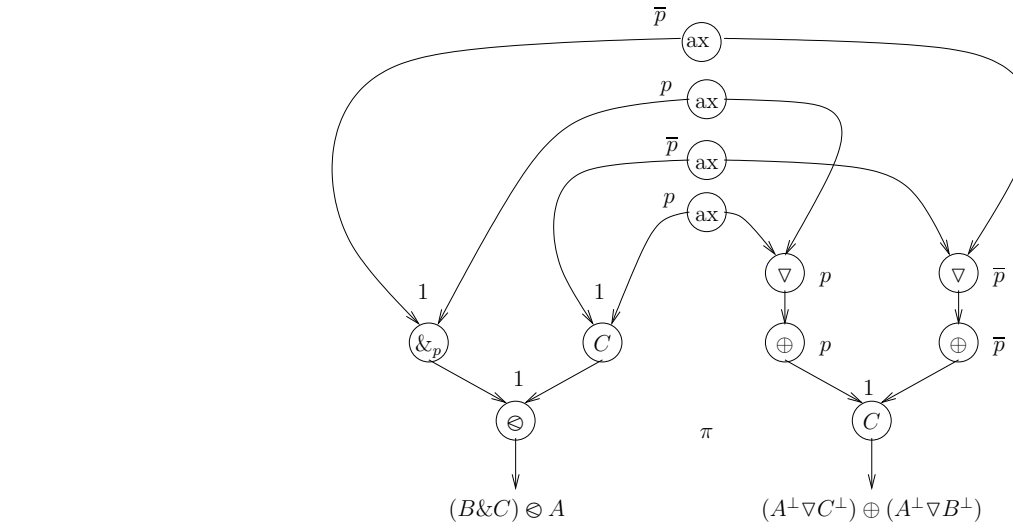


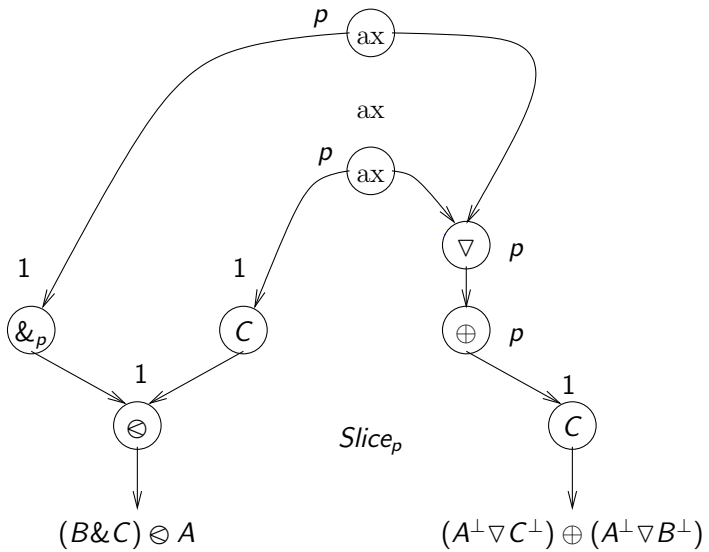
(Anyway, Melliès's proof structure becomes correct after **cut reduction**).

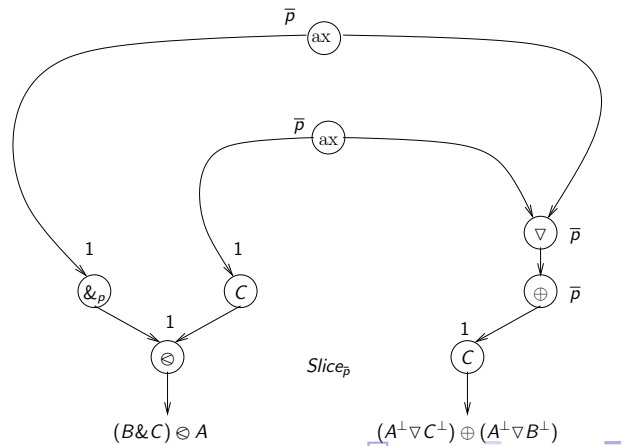
CyMALL examples: Girard's proof structure

The following proof structure π is not correct (not retractile);

- nevertheless, every multiplicative slice, $Slice_p$ and $Slice_{\overline{p}}$, is retractile!
- a criterion only based on "correctness by mult. slices" is not enough!







embedding Lambek Calculus into CyMALL

1. A is a **pure Lambek formula (pLF)** if it is a CyMLL formula recursively built according to this grammar:

$$A \ := \ \text{positive atoms} \mid A \oslash A \mid A^{\perp} \triangledown A \mid A \triangledown A^{\perp}.$$

2. A is an **additive Lambek formula (aLF or simply LF)** if it is a CyMALL formula recursively built according this grammar:

$$A \ := \ \text{pLF} \mid A \& A \mid A \oplus A.$$

3. S is a **Lambek sequent of CyMALL** iff $S = (\Gamma^{\perp}, A)$, where A is a non-void LF and $\Gamma^{\perp}$ is a possibly empty finite sequence of negations of LFs (i.e., Γ is a possibly empty sequence of LFs and $\Gamma^{\perp}$ is obtained by taking the negation of each formula in Γ).
4. a **Lambek proof** is any derivation built by the CyMALL inference rules whose premise(s) and conclusions are Lambek sequents.
5. a **Lambek CyMALL proof net** is any CyMALL PN whose edges are labeled by LF or negation of LF and whose conclusions build a Lambek sequent.

Parsing with Lambek Calculus

- LC represents the first attempt of **parsing as deduction**, i.e., parsing of natural language by means of a logical system.
- In LC parsing is interpreted as type checking in the form of theorem proving of Gentzen sequents.
- Types (i.e. propositional formulas) are associated to words in the lexicon; when a string $w_1 ... w_n$ is tested for grammaticality, types $t_1, ..., t_n$ are associated with these words, then parsing reduces to proving the derivability of a two-sided sequent $t_1, ..., t_n \vdash s$.
- Remind that proving a two sided Lambek derivation $t_1, ..., t_n \vdash s$ is equivalent to prove the one-sided sequent $\vdash t_n^{\perp}, ... t_1^{\perp}, s$ where $t_i^{\perp}$ is the dual (i.e., linear negation) of type t_i.

  In one-sided sequent calculus, phrases or sentences should be read "like in a mirror" (following opposite direction to the natural one).
- Forcing some constraints on the Exchange rule (e.g., by allowing only *cyclic permutations* over sequents of formulas) gives the required computational control needed to view theorem proving (or PN construction) as parsing in Lambek Categorial Grammar style.

main syntactical ambiguity problems with LC parsing

LC parsing presents some syntactical ambiguity problems; there may be:

(non canonical proofs) more than one (cut-free) proof for the same sequent conclusion;

(lexical polymorphism) more than one type associated with a single word.

- CyMALL Lambek PNs may be considered a(n elegant) solution to both of these problems.

linguistic parsing via PNs

Assume the following lexicon, where *linear implication* $\multimap$ (resp., $\circ\!\!-$) is traditionally used for expressing types in two-sided sequent parsing:

1. *Sollozzo, Sam, Vito* $=$ np;
2. $\begin{array}{lllll} \textit{trusts} & = & np \multimap (s \circ\!\!- np) & = & np^{\perp} \nabla (s \nabla np^{\perp}) \\ & \equiv & (np \multimap s) \circ\!\!- np & = & (np^{\perp} \nabla s) \nabla np^{\perp}; \end{array}$
3. $\textit{him} = (s \circ\!\!- np) \multimap s = (s \nabla np^{\perp})^{\perp} \nabla s = (np \oslash s^{\perp}) \nabla s;$

Cases of lexical ambiguity follow to words with several possible formulas A and B assigned it. For example, a verb like "*to believe*" can express a relation between two persons (interpreted as np) like in S1, or between a person and a statement (interpreted as s) like in S2:

Sollozzo believes Vito	(S1)
Sollozzo believes Vito trusts him	(S2)

We can express this verb ambiguity by two lexical assignments as follows:

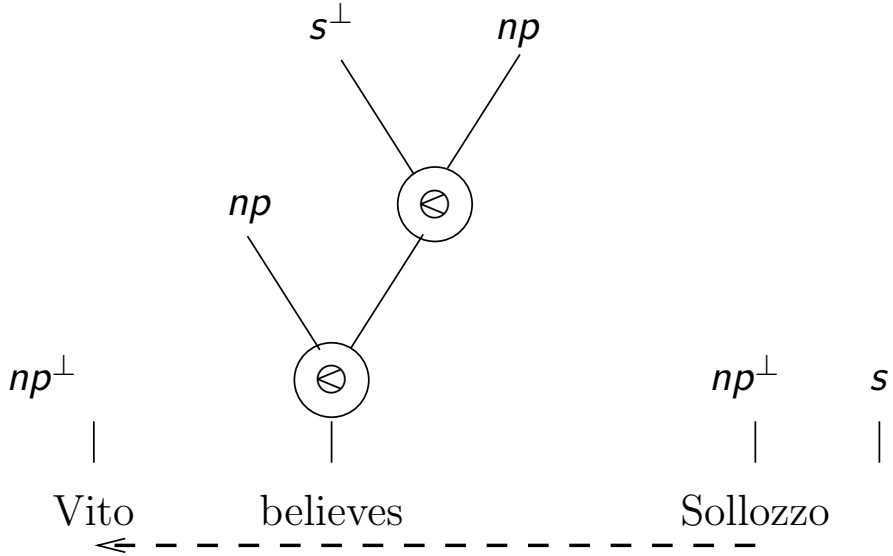
3. $\textit{believes} = (np \multimap s) \circ\!\!- np = (np^{\perp} \nabla s) \nabla np^{\perp};$
4. $\textit{believes} = (np \multimap s) \circ\!\!- s = (np^{\perp} \nabla s) \nabla s^{\perp}.$

parsing of S1: "Sollozzo believes Vito"

- **via derivation in the sequent calculus**:

$$\dfrac{\dfrac{}{np^{\perp}, np}\, id_1 \qquad \dfrac{\dfrac{}{s^{\perp}, s}\, id_2 \qquad \dfrac{}{np, np^{\perp}}\, id_3}{s^{\perp} \otimes np, np^{\perp}, s}\, \otimes}{np^{\perp}, np \otimes (s^{\perp} \otimes np), np^{\perp}, s}\, \otimes$$

- **via proof net construction**: we start with the formula tree of each conclusion (no matter the order!) including s (type for sentence)

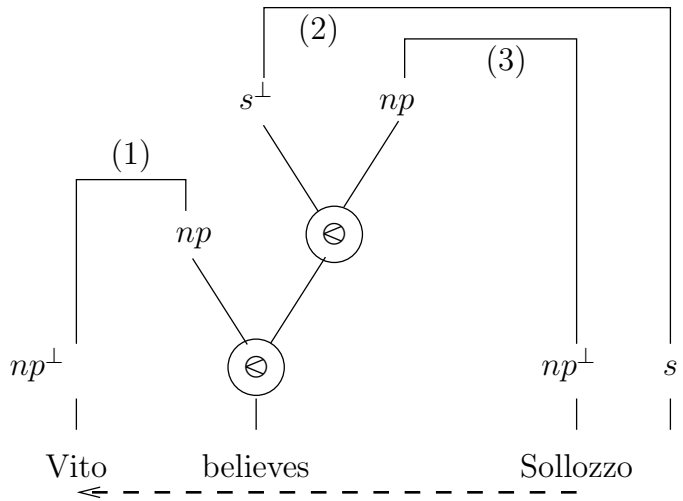


parsing of S1: "Sollozzo believes Vito" (continues)

- via derivation in the sequent calculus:

$$\dfrac{\dfrac{}{np^\perp, np}\, id_1 \qquad \dfrac{\dfrac{}{s^\perp, s}\, id_2 \qquad \dfrac{}{np, np^\perp}\, id_3}{s^\perp \otimes np, np^\perp, s}\, \otimes}{np^\perp, np \otimes (s^\perp \otimes np), np^\perp, s}\, \otimes$$

- via proof net construction:
  – we start with the formula tree of each conclusions
  – then we "incrementally put" the axiom links on the top.

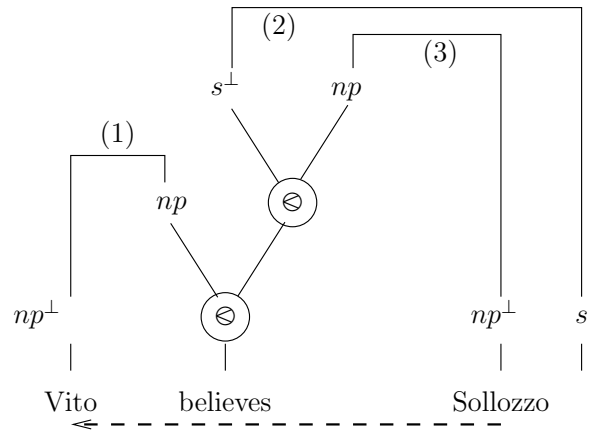


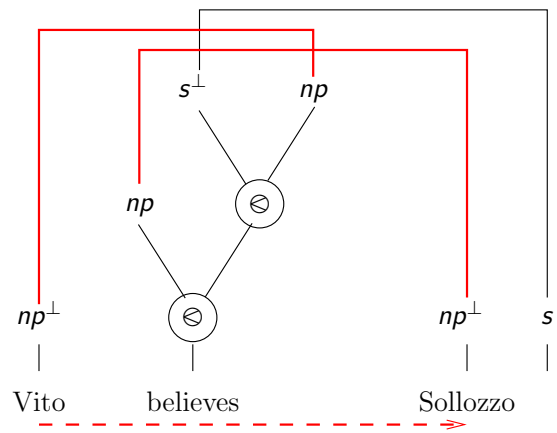
parsing of S1: "Sollozzo believes Vito" (continues)

- via derivation in the sequent calculus:

$$\dfrac{\dfrac{}{np^{\perp}, np}\, id_1 \qquad \dfrac{\dfrac{}{s^{\perp}, s}\, id_2 \qquad \dfrac{}{np, np^{\perp}}\, id_3}{s^{\perp} \parr np, np^{\perp}, s} \parr}{np^{\perp}, np \parr (s^{\perp} \parr np), np^{\perp}, s} \parr$$

- via proof net construction:
  - we start with the formula tree of each conclusions
  - then we "incrementally put" the axiom links on the top.





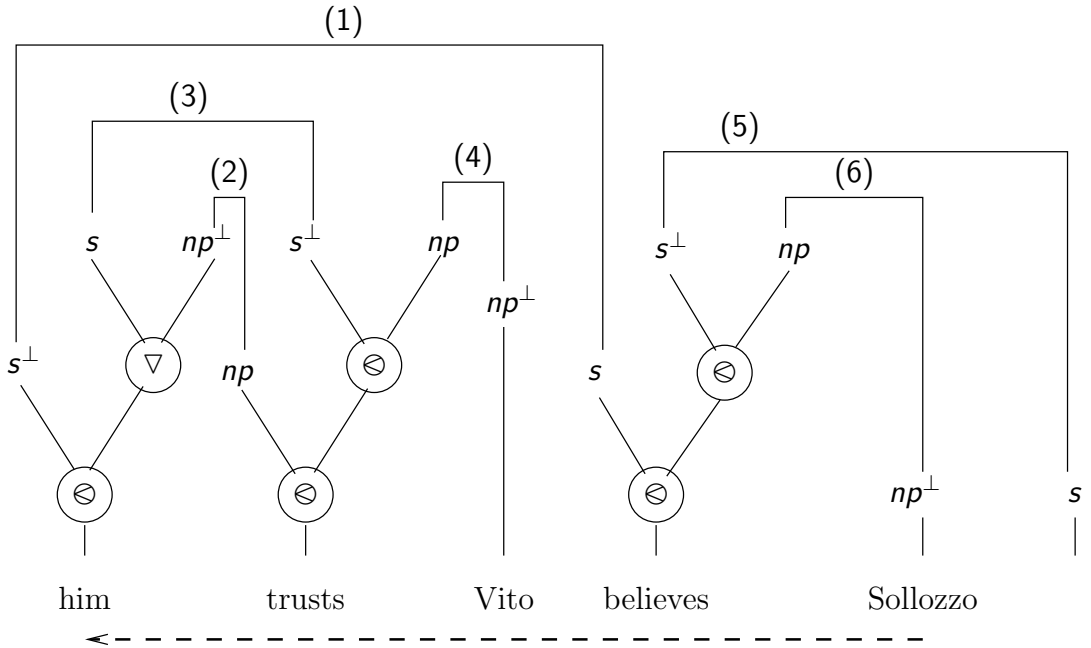
Remarks

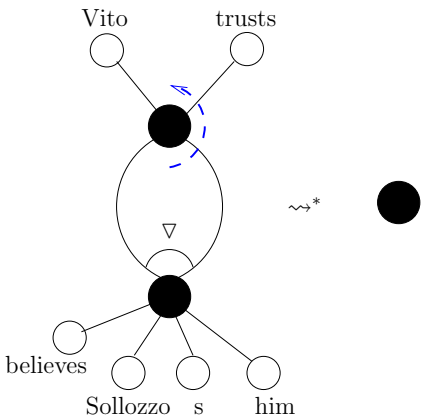
there are two ways of linking dual pairs of literals $(np, np^{\perp})$ both of them leading to correct PNs; but only one of them corresponds to S1.

parsing of S2: "Sollozzo believes Vito trusts him"

$$\dfrac{\dfrac{}{s^\perp, s}\,id_1 \qquad \dfrac{\dfrac{\dfrac{}{np^\perp, np}\,id_2 \qquad \dfrac{\dfrac{}{s^\perp, s}\,id_3 \qquad \dfrac{}{np, np^\perp}\,id_4}{s^\perp \otimes np, np^\perp, s}\,\otimes}{s, np^\perp, np \otimes (s^\perp \otimes np), np^\perp}\,\otimes}{s \nabla np^\perp, np \otimes (s^\perp \otimes np), np^\perp}\,\nabla}{s^\perp \otimes (s \nabla np^\perp), np \otimes (s^\perp \otimes np), np^\perp, s}\,\otimes \qquad \dfrac{\dfrac{}{s^\perp, s}\,id_5 \qquad \dfrac{}{np, np^\perp}\,id_6}{s^\perp \otimes np, np^\perp, s}\,\otimes$$

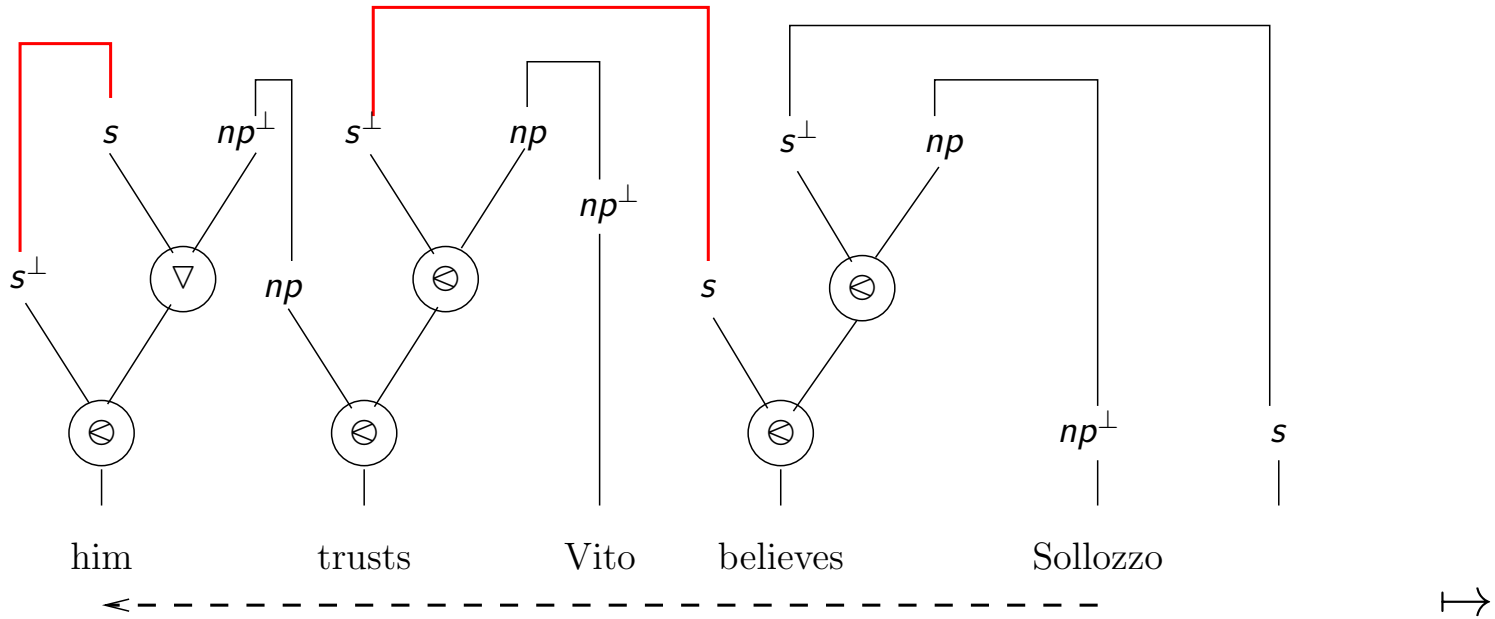
$$\dfrac{s^\perp \otimes (s \nabla np^\perp), np \otimes (s^\perp \otimes np), np^\perp, s \otimes (s^\perp \otimes np), np^\perp, s}{}\,\otimes$$

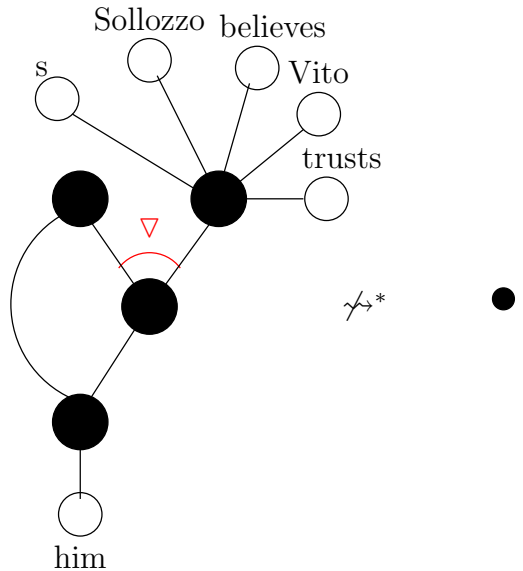




a wrong solution for "Sollozzo believes Vito trusts him"

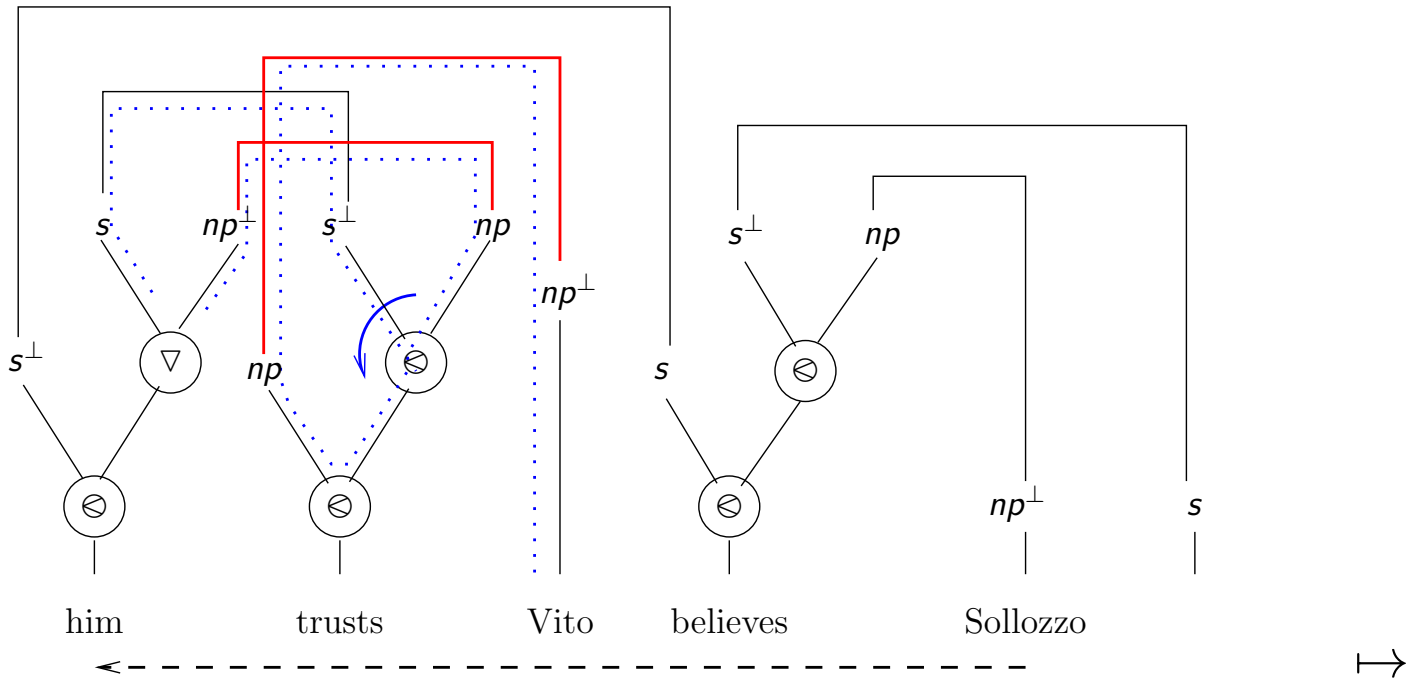
a wrong solution with an un-correct axiom linkings for literal pairs $s, s^{\perp}$

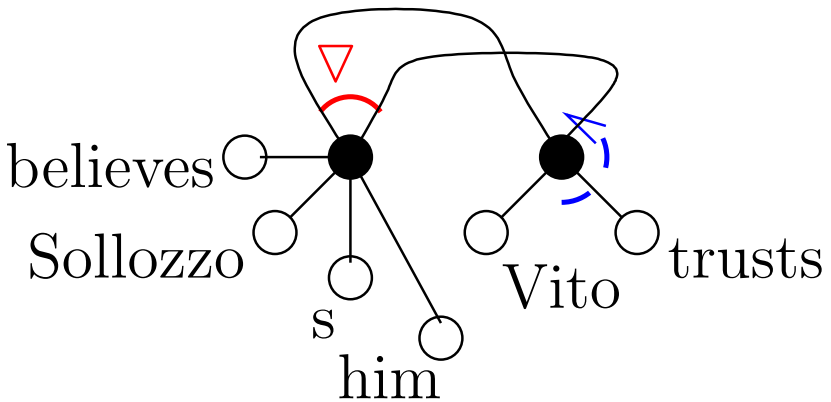




an other wrong solution for "Soll. believes Vito trusts him"

a wrong solution with an un-correct axiom linkings for pairs $np, np^{\perp}$

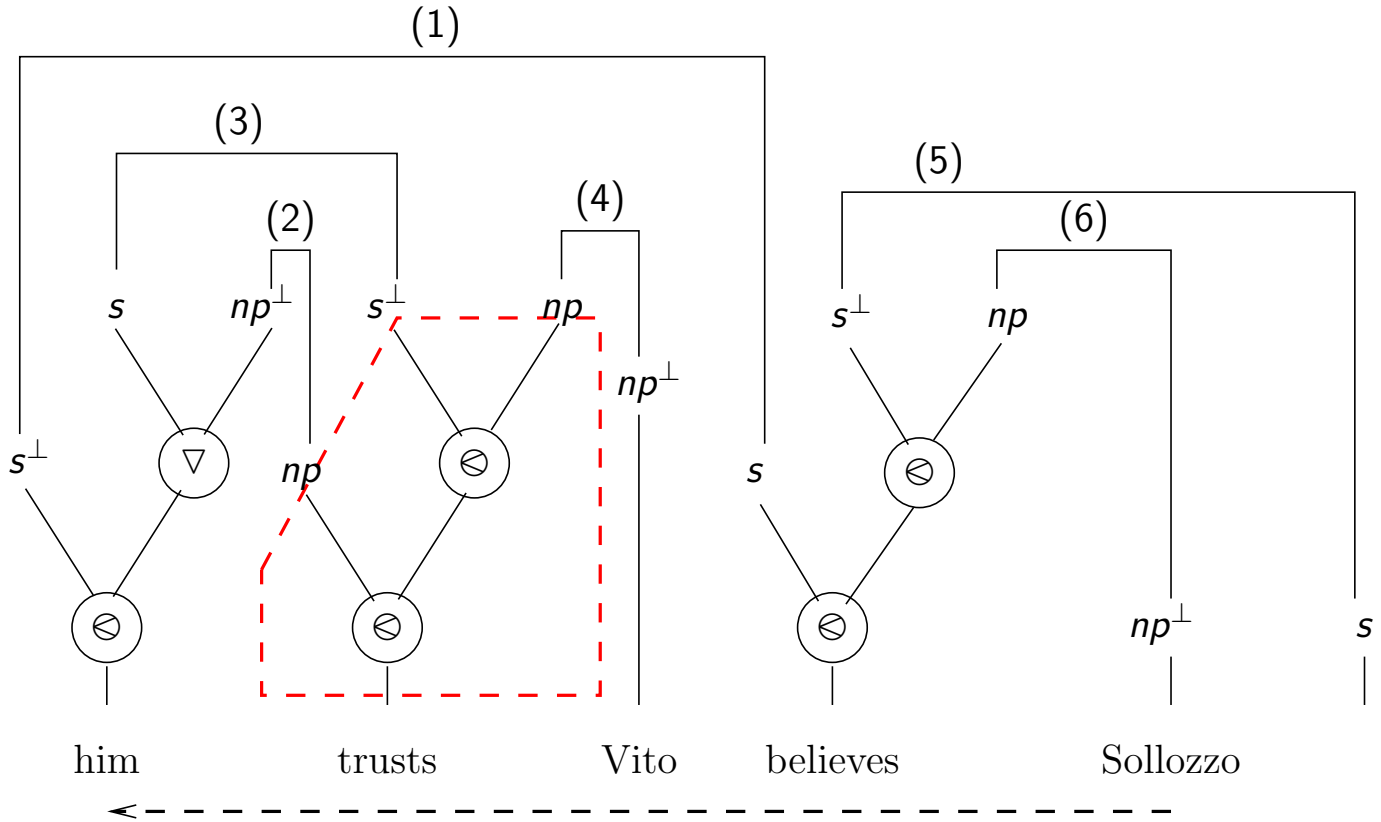




$\leadsto^*$

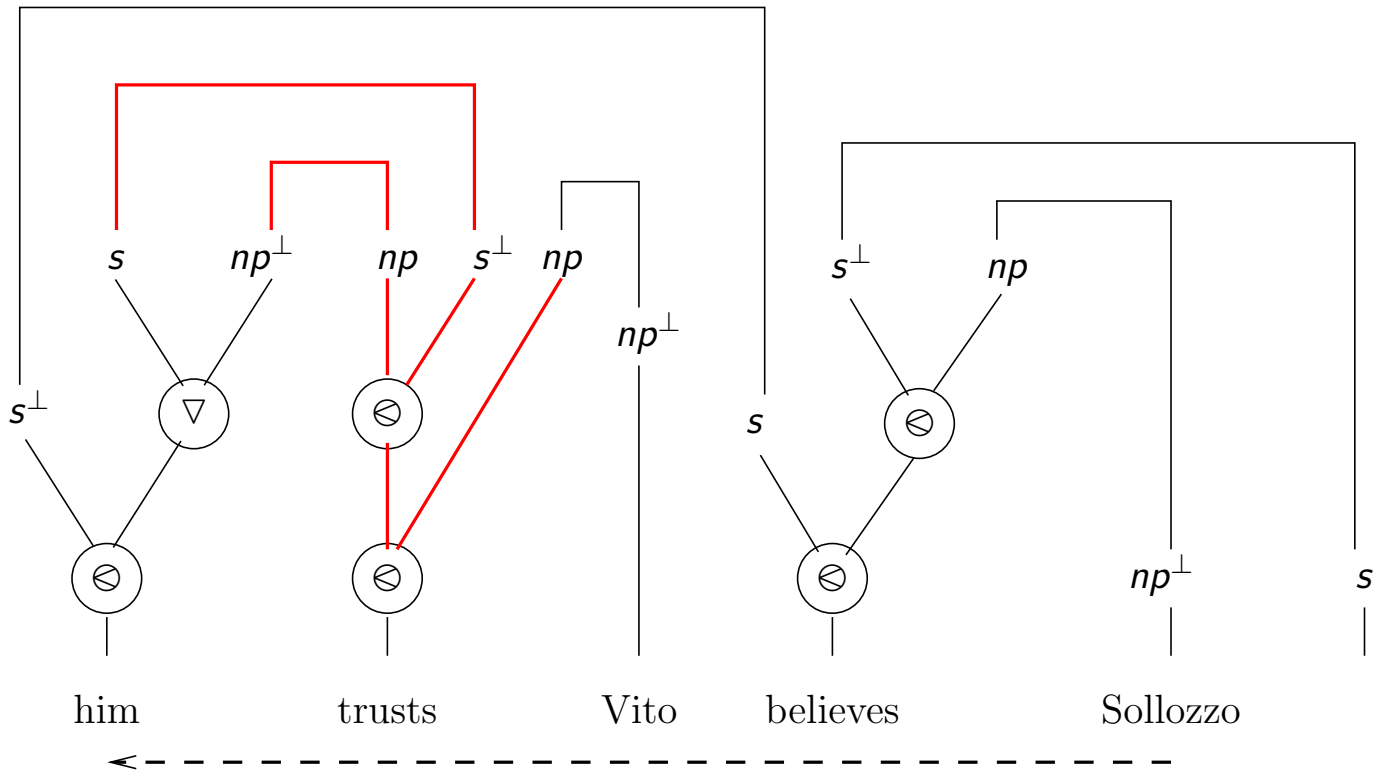
alternative parsing for "Sollozzo believes Vito trusts him"

PNs are modular: in a correct PN we can replace/interchange "modules" with same "behavior" and get still a correct PN.



alternative parsing for "Sollozzo believes Vito trusts him"

PNs are modular: in a correct PN we can replace/interchange "modules" with same "behavior" and get still a correct PN.



Here is an alternative parsing solution for sentence 2 with same matching for the axiom links but different (even though equivalent) type for the lexical item "trusts" $= np \multimap (s \circ\!\!- np) \;=\; np^{\perp} \nabla (s \nabla np^{\perp})$ (*take the dual!*)

lexical ambiguity

Additive connectives, & **and** $\oplus$, allow superpositions of types;
in particular we can join the previous two assignments 3 and 4

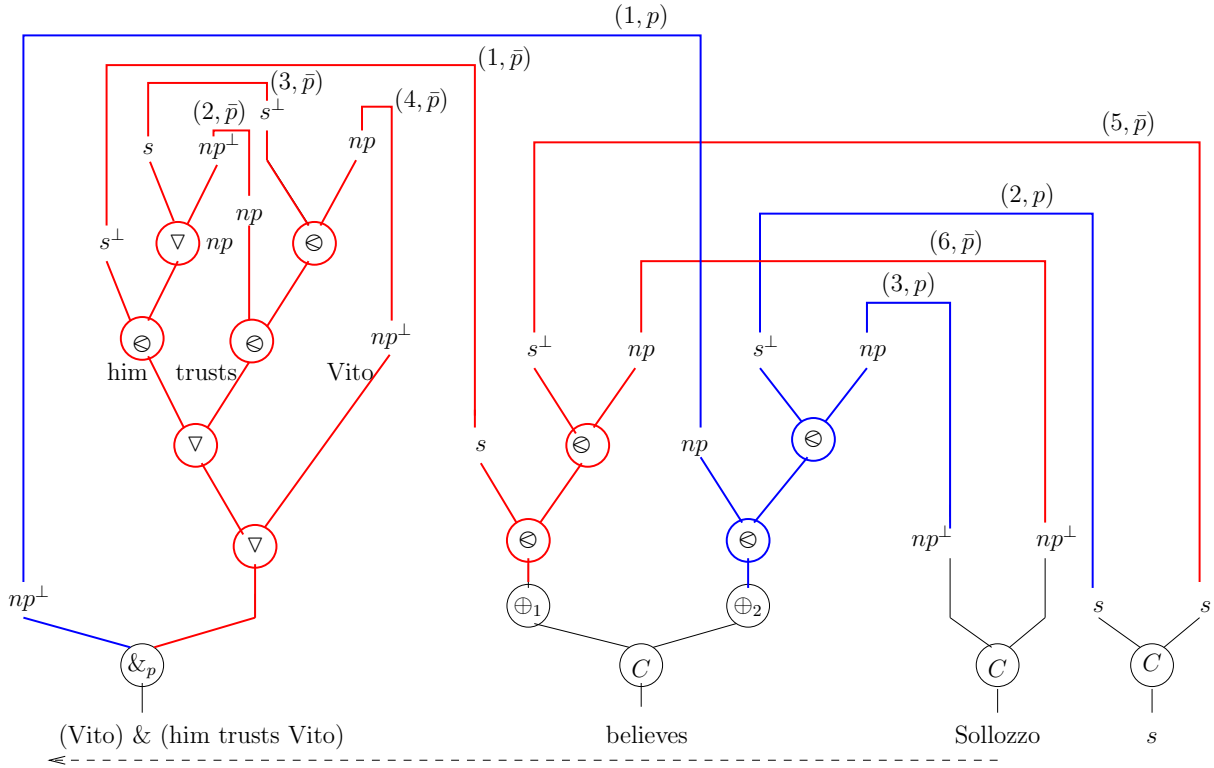
3. $\mathit{believes} = (np \multimap s) \circ\!\!- np \;=\; (np^{\perp} \triangledown s) \triangledown np^{\perp}$;
4. $\mathit{believes} = (np \multimap s) \circ\!\!- s \;=\; (np^{\perp} \triangledown s) \triangledown s^{\perp}$.

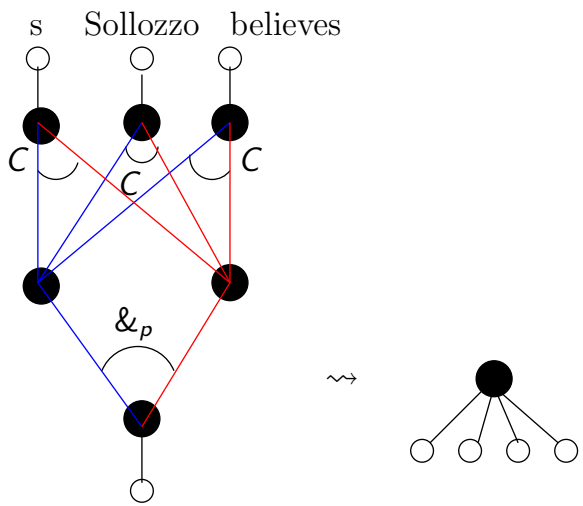
into a single additive assignment 5

5. $\text{believes} = ((np \multimap s) \circ\!\!- np) \& ((np \multimap s) \circ\!\!- s) = ((np^{\perp} \triangledown s) \triangledown np^{\perp}) \& ((np^{\perp} \triangledown s) \triangledown s^{\perp})$.

and get an unique PN parsing the superposition of both sentences:
(S1) *Sollozzo believes Vito* & *Sollozzo believes Vito trusts him* (S2)

a “sequent” or “box-like” solution (minimal superposition)



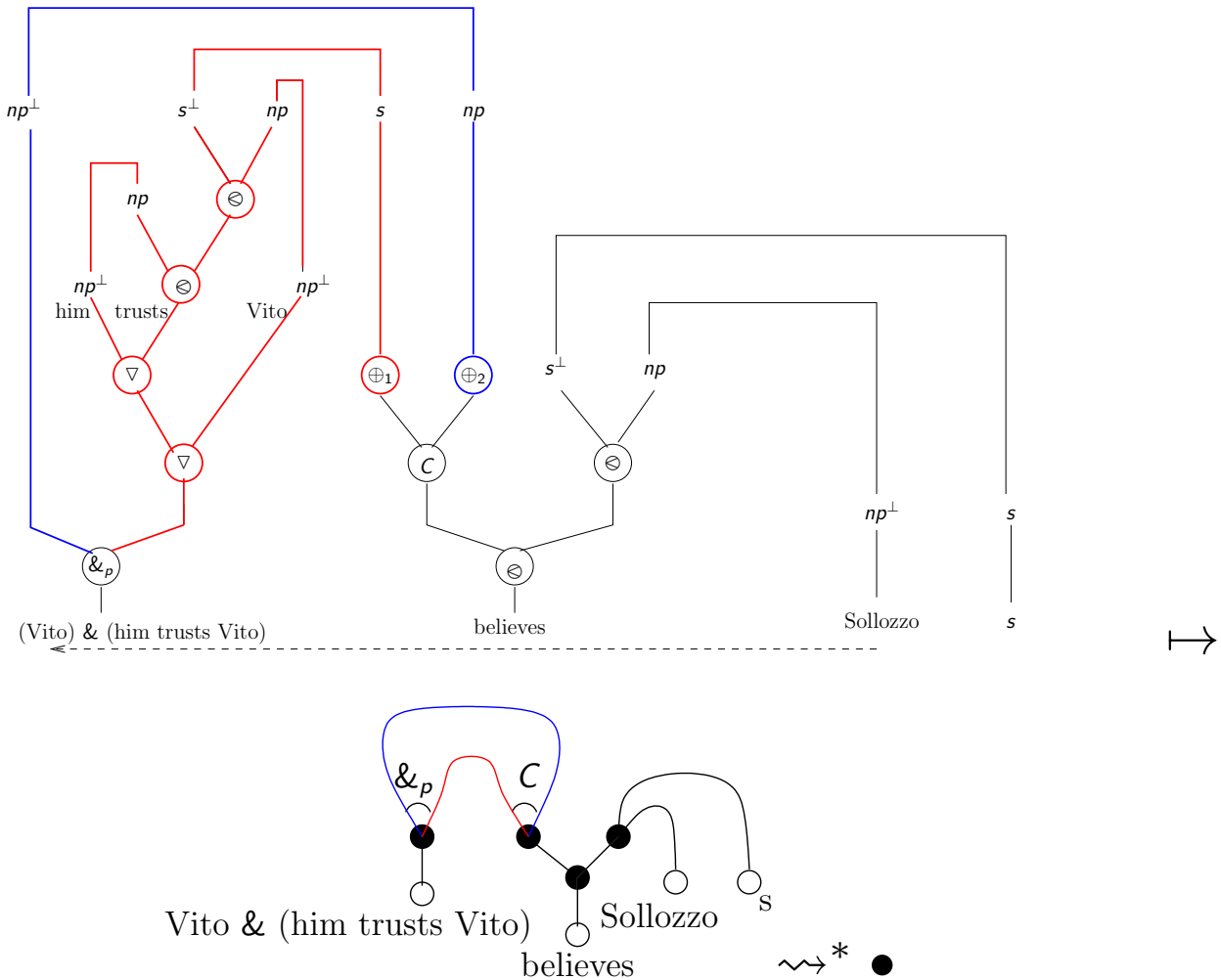


a "more compact" solution (maximal superposition)

by distributivity of negative connectives, a more compact lexical entry 6

5. believes $= ((np \multimap s) \circ\!\!- np) \& ((np \multimap s) \circ\!\!- s) = ((np^{\perp} \triangledown s) \triangledown np^{\perp}) \& ((np^{\perp} \triangledown s) \triangledown s^{\perp})$.

6. believes $= ((np \multimap s) \circ\!\!- (s \oplus nps)) = ((np^{\perp} \triangledown s) \triangledown (np^{\perp} \& s^{\perp}))$.



conclusions and further works

Retraction Systems represent an useful computational tool for:

- proof search, since by confluence, we can chose some "optimal retraction strategies" w.r.t. searching space, backtracking, ecc., (Maieli, RTA-2014)
- classifying the complexity class (in time) of correctness criteria:
  - **MLL PS correctness**:
    – **Linear in time** w.r.t. the PSs size (Guerrini, LICS-1999);
    – **NL-complete** (de Naurois-Mogbil, CSL-2007);
  - **CyMLL PS correctness**:
    – **Quadratic in time** w.r.t. the PSs size (Mogbil, CSL-2001);
  - **CyMALL PS correctness**:
    – **NL-complete** (de Naurois-Mogbil, LICS-2008);
    – **complexity in time is unknown**.

thank you for your attention!