

# Types, Networks & Stuff

Kokos

LoLa 2019

7 January 2020

# Overview

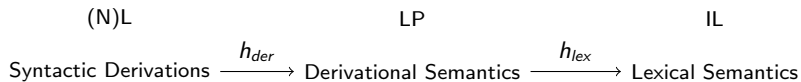
The paper starts with Lambek Calculus, some how uses dependancy labels in some of its semantic types, provides a parsing algorithm for it; there are neural networks and vectors used and some accuracy results provided, but I am still unsure about the contributions of the paper and their relevance

# Overview

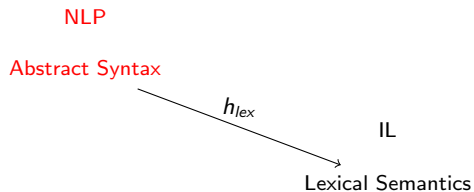
The paper starts with Lambek Calculus, some how uses dependancy labels in some of its semantic types, provides a parsing algorithm for it; there are neural networks and vectors used and some accuracy results provided, but I am still unsure about the contributions of the paper and their relevance

- λ Abstract Syntax with NLP
- λ Extracting & Learning Type Assignments
- λ Navigating proofs with neural nets
- λ Lexicalized syntax in language modeling

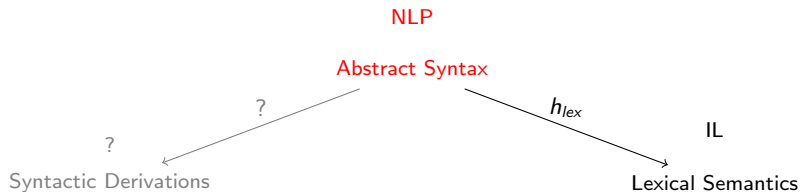
# This Timeline



# Alternative Timeline



# Alternative Timeline



## Grammar

### ILL + Structural Control Modalities

$$\mathcal{T} := A \mid \diamond^d T_1 \rightarrow T_2$$

$A \in \mathcal{A} \quad :: \quad$  Atoms denoting complete phrases (NP, S, ...)

$d \in \mathcal{D} \quad :: \quad$  Grammatical relations (subject, object, ...)

$\diamond^d T \quad :: \quad$  Type demarkated by *dependency domain*  $d$

$T_1 \rightarrow T_2 \quad :: \quad$  Linear functor from  $T_1$  to  $T_2$

-- I did not follow this part. First of all, why dependency labels are used? Secondly, how are they used?

## Why?

- ▶ Easier to extract from corpora
- ▶ Drastic reduction in lexical ambiguity
- ▶ More informative for semantics
- ▶ Built-in Interpretability
- ▶ Diamonds can regulate parsing (?)



-- I did not follow this part. First of all, why dependency labels are used? Secondly, how are they used?

## Why?

- ▶ Easier to extract from corpora
- ▶ Drastic reduction in lexical ambiguity
- ▶ More informative for semantics
- ▶ Built-in Interpretability
- ▶ Diamonds can regulate parsing (?)

## How?

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash (M N) : B} \rightarrow E$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow I$$

$$\frac{\Gamma \vdash A}{\langle \Gamma \rangle^d \vdash \diamond^d A} \diamond^d I$$

$$\frac{\Delta \vdash \diamond^d A \quad \Gamma, \langle A \rangle^d \vdash B}{\Gamma, \Delta \vdash B} \diamond^d E$$

A dataset of types & proofs

# Extracting (1/3)

## Algorithm

Graph flooding on syntactic parse graphs

Init with maps:

- from POS/Phrasal-tags to atoms
- dependency labels to diamond operators

Two subroutines; each *selects* untyped nodes and *types* them

# Extracting (1/3)

## Algorithm

Graph flooding on syntactic parse graphs

Init with maps:

- from POS/Phrasal-tags to atoms
- dependency labels to diamond operators

Two subroutines; each *selects* untyped nodes and *types* them

```
function TYPEDAG: DAG  $D \rightarrow$  DAG  
   $D \leftarrow$  DetachNonLocal( $D$ )  
  while  $D$  is not fully typed do  
     $D \leftarrow$  TypeStandaloneNodes( $D$ )  
     $D \leftarrow$  TypeHeadsAndMods( $D$ )  
  end while  
return AttachNonLocal( $D$ )  
end function
```

# Extracting (2/3)

## $\lambda$ 1 Stand-Alone Nodes

**select** untyped nodes with

- 1 no incoming head/mod edge
- 2 no untyped daughters (except heads/mods)

**type** translate pos/tag to atom

## $\lambda$ 2 Heads and Mods

**select** untyped nodes that

- 1 incoming head/mod edge
- 2 have a typed parent
- 3 have all sisters (except head/mods) typed

**type** endofunctor of parent if mod, else functor from vector of arguments<sup>1</sup> to parent type

---

<sup>1</sup>minus hypotheses

# Extracting (3/3)

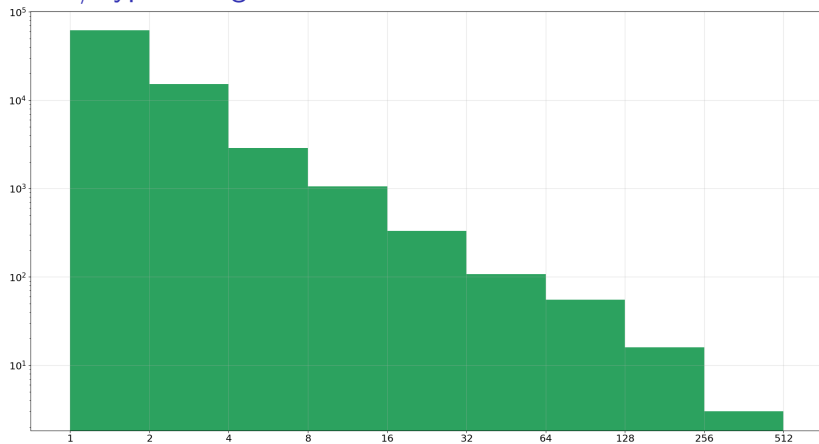
## Non-Local Dependencies

```
function DETACHNONLOCAL: DAG  $D \rightarrow$  TREE
  for each  $e \in \text{FILTER}(\text{REENTRANT}, D.\text{edges})$  do
     $c \leftarrow \text{copy}(e.\text{target})$ 
     $D.\text{nodes} \leftarrow D.\text{nodes} \cup \{c\}$ 
     $D.\text{edges} \leftarrow D.\text{edges} - \{e\}$ 
     $D.\text{edges} \leftarrow D.\text{edges} \cup \{(e.\text{source}, e.\text{label}, c)\}$ 
  end for
return  $D$ 
end function
```

Mommy, where do types come from?

# Lexical Type Ambiguity

Word/Type histogram





# Supertagging (1/3)

## General idea

$$p(t_1, t_2, \dots t_n | w_1, w_2, \dots w_n, \theta)$$

## Markov Assumption

$$\approx \prod_{i=1}^n p(t_i | w_1, w_2, \dots w_n, \theta)$$

- ☺ Seq2Seq Classification
- ☹ Single Answer
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

# Supertagging (1/3)

## General idea

$$p(t_1, t_2, \dots t_n | w_1, w_2, \dots w_n, \theta)$$

## Markov Assumption

$$\approx \prod_{i=1}^n p(t_i | w_1, w_2, \dots w_n, \theta)$$

- ☺ Seq2Seq Classification
- ☹ Single Answer
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

# Supertagging (2/3)

## General idea

$$p(t_1, t_2, \dots t_n | w_1, w_2, \dots w_n, \theta)$$

## ~~Markov Assumption~~

$$\approx \prod_{i=1}^n p(t_i | t_1, \dots t_{i-1}, w_1, \dots, w_n, \theta)$$

- ☺ Seq2Seq **Transduction**
- ☺ **Many Answers**
- ☹ Sample Sparsity
- ☹ Closed Domain Assumption

# Supertagging (3/3)

## A step further

The syntax of the type system forms a simple CFG

$$S \Longrightarrow A \qquad \forall A \in \mathcal{A}$$

$$S \Longrightarrow d S S \qquad \forall d \in \mathcal{D}$$

# Supertagging (3/3)

## A step further

The syntax of the type system forms a simple CFG

$$S \Longrightarrow A \qquad \forall A \in \mathcal{A}$$

$$S \Longrightarrow d S S \qquad \forall d \in \mathcal{D}$$

- ▶ Supertagging as conditional CFG generation
- ▶ CFG terminals as decoding targets

$$\approx \prod_{i=1}^m p(\sigma_i | \sigma_1, \dots, \sigma_{i-1}, w_1, \dots, w_n, \theta)$$

$$\sigma \in \mathcal{A} \cup \mathcal{D} \cup \{<\text{SEP}>\}$$

# Supertagging (3/3)

## A step further

The syntax of the type system forms a simple CFG

$$\begin{aligned} S &\Longrightarrow A && \forall A \in \mathcal{A} \\ S &\Longrightarrow d S S && \forall d \in \mathcal{D} \end{aligned}$$

- ▶ Supertagging as conditional CFG generation
- ▶ CFG terminals as decoding targets

$$\approx \prod_{i=1}^m p(\sigma_i | \sigma_1, \dots, \sigma_{i-1}, w_1, \dots, w_n, \theta)$$

$$\sigma \in \mathcal{A} \cup \mathcal{D} \cup \{<\text{SEP}>\}$$

- ☺ ~~Sample Sparsity~~ Many (sub)type examples
- ☺ ~~Closed Domain Assumption~~ Inductive construction of any type

## Structural Ambiguity

# Navigating Proofs

## Parse State

- ▶ A logical judgement (premises & conclusion)
- ▶ Word associations for (some) premise formulas
- ▶ A single element stack



# Navigating Proofs

## Parse State

- ▶ A logical judgement (premises & conclusion)
- ▶ Word associations for (some) premise formulas
- ▶ A single element stack

## Framework

Given a parse state:

- 1 Decide between introduction  $\oplus$  elimination
- 2 Perform either
- 3 Update state(s)
- 4 Repeat

# Proof Ambiguity

## The Problem

- ▶ Introduction steps are deterministic
- ▶ Eliminations are not

# Proof Ambiguity

## The Problem

- ▶ Introduction steps are deterministic
- ▶ Eliminations are not

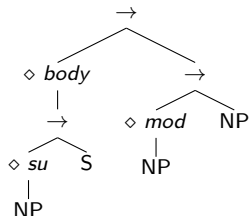
## Key Insight

Elimination branching  $\sim$  binary sequence chunking  
Semantic content can help disambiguate structure

# Vectorizing Types

Types are trees

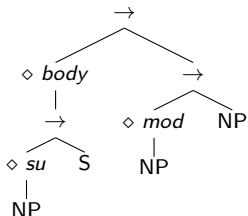
$$\diamond^{body}(\diamond^{su}NP \rightarrow S) \rightarrow \diamond^{mod}NP \rightarrow NP$$



# Vectorizing Types

Types are trees

$$\diamond^{body}(\diamond^{su} NP \rightarrow S) \rightarrow \diamond^{mod} NP \rightarrow NP$$



- ▶ Atoms as vectors in  $\mathbb{R}^n$
- ▶ Dependencies as functions  $\mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n$

# Proof Traversal

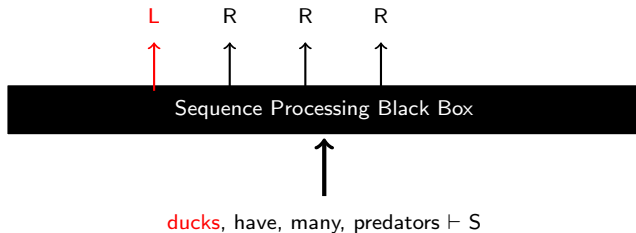
$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{many} \vdash \text{NP} \rightarrow \text{NP}} \quad \rightarrow E \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \frac{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S} \quad \text{Ax.}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \rightarrow E}{\text{ducks, have, many, predators} \vdash \text{S}}
 \end{array}$$

Sequence Processing Black Box

ducks, have, many, predators  $\vdash$  S

# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{many} \vdash \text{NP} \rightarrow \text{NP}} \quad \rightarrow E \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP}}{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}} \quad \text{Ax.}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \rightarrow E \\
 \hline
 \text{ducks, have, many, predators} \vdash \text{S}
 \end{array}$$



# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{many} \vdash \text{NP} \rightarrow \text{NP}} \quad \rightarrow E \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP}}{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}} \quad \text{Ax.}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \rightarrow E \\
 \hline
 \text{ducks, have, many, predators} \vdash \text{S}
 \end{array}$$

Sequence Processing Black Box

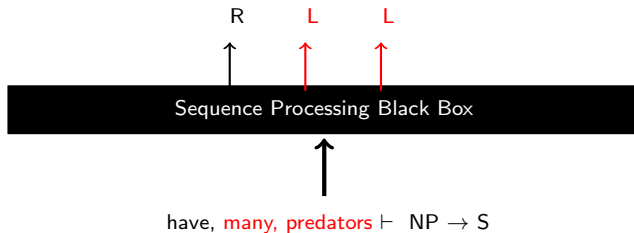


have, many, predators  $\vdash$  NP  $\rightarrow$  S



# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{many} \vdash \text{NP} \rightarrow \text{NP}} \quad \rightarrow E \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \text{Ax.}}{\text{ducks, have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \rightarrow E \\
 \hline
 \text{ducks, have, many, predators} \vdash \text{S}
 \end{array}$$



# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{many} \vdash \text{NP} \rightarrow \text{NP}} \quad \rightarrow E \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S} \quad \text{Ax.}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \rightarrow E \\
 \hline
 \text{ducks, have, many, predators} \vdash \text{S}
 \end{array}$$

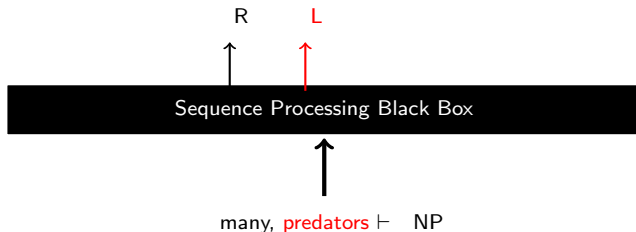
Sequence Processing Black Box



many, predators  $\vdash$  NP

# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{many, predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}} \quad \text{Ax.} \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \text{many} \vdash \text{NP} \rightarrow \text{NP}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \quad \rightarrow E}{\text{ducks, have, many, predators} \vdash \text{S}} \quad \rightarrow E
 \end{array}$$



# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{many, predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}} \quad \text{Ax.} \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \rightarrow E}{\text{ducks, have, many, predators} \vdash \text{S}} \rightarrow E
 \end{array}$$

Sequence Processing Black Box

# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{many, predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\rightarrow E} \quad \frac{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S} \quad \text{Ax.}}{\rightarrow E} \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}}}{\text{ducks, have, many, predators} \vdash \text{S}}
 \end{array}$$

Sequence Processing Black Box

|                 |   |                              |                        |
|-----------------|---|------------------------------|------------------------|
| Training sample | : | Elimination branching        | } Training Parallelism |
| Sentence        | : | N <i>independent</i> samples |                        |

# Proof Traversal

$$\begin{array}{c}
 \frac{\text{predators} \vdash \text{NP} \quad \text{Ax.}}{\text{many, predators} \vdash \text{NP}} \quad \frac{\text{many} \vdash \text{NP} \rightarrow \text{NP} \quad \text{Ax.}}{\text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}} \quad \text{Ax.} \\
 \frac{\text{ducks} \vdash \text{NP} \quad \text{Ax.} \quad \frac{\text{many, predators} \vdash \text{NP} \quad \text{have} \vdash \text{NP} \rightarrow \text{NP} \rightarrow \text{S}}{\text{have, many, predators} \vdash \text{NP} \rightarrow \text{S}} \rightarrow E}{\text{ducks, have, many, predators} \vdash \text{S}} \rightarrow E
 \end{array}$$

Sequence Processing Black Box

Parsing in  $\mathcal{O}(1)^2$

# Parsing as Latent Permutation

$$np, np \rightarrow np \rightarrow s, np \rightarrow np, np \vdash s$$



# Parsing as Latent Permutation

$$np_1, np_2 \rightarrow np_3 \rightarrow s_1, np_4 \rightarrow np_5, np_6 \vdash s_2$$

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

|        | $np_2$ | $np_3$ | $np_4$ | $s_2$ |
|--------|--------|--------|--------|-------|
| $np_1$ |        | X      |        |       |
| $s_1$  |        |        |        | X     |
| $np_5$ | X      |        |        |       |
| $np_6$ |        |        | X      |       |

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4 \times 4} =$$

|        | $np_2$ | $np_3$ | $np_4$ | $s_2$ |
|--------|--------|--------|--------|-------|
| $np_1$ | 0      | 1      | 0      | 0     |
| $s_1$  | 0      | 0      | 0      | 1     |
| $np_5$ | 1      | 0      | 0      | 0     |
| $np_6$ | 0      | 0      | 1      | 0     |

$$\text{match}(N) = P\mathcal{R}^T$$

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4 \times 4} =$$

|        | $np_2$ | $np_3$ | $np_4$ | $s_2$ |
|--------|--------|--------|--------|-------|
| $np_1$ | 0      | 1      | 0      | 0     |
| $s_1$  | 0      | 0      | 0      | 1     |
| $np_5$ | 1      | 0      | 0      | 0     |
| $np_6$ | 0      | 0      | 1      | 0     |

$$\text{match}(N) = P\mathcal{R}^T$$

►  $\mathcal{R}$  is *discrete*

# Parsing as Latent Permutation

$$np_1^+, np_2^- \rightarrow np_3^- \rightarrow s_1^+, np_4^- \rightarrow np_5^+, np_6^+ \vdash s_2^-$$

$$P = [np_1, s_1, np_5, np_6] \quad N = [np_2, np_3, np_4, s_2]$$

$$\mathcal{R} \in \mathbb{B}^{4 \times 4} =$$

|        | $np_2$ | $np_3$ | $np_4$ | $s_2$ |
|--------|--------|--------|--------|-------|
| $np_1$ | 0      | 1      | 0      | 0     |
| $s_1$  | 0      | 0      | 0      | 1     |
| $np_5$ | 1      | 0      | 0      | 0     |
| $np_6$ | 0      | 0      | 1      | 0     |

$$\text{match}(N) = P\mathcal{R}^T$$

- ▶  $\mathcal{R}$  is *discrete*
- ▶ .. but its continuous relaxations can be approximated (Sinkhorn-Knopp)

# References

## Grammar & Extraction

- ▶ **ÆTHEL: Automatically Extracted Type-Logical Derivations for Dutch** [link](#)

## Supertagging & Parsing

- ▶ **Constructive Type-Logical Supertagging with Self-Attention Networks** [link](#)
- ▶ **Deductive Parsing with an Unbounded Type Lexicon** [link](#)

## Permutations

- ▶ **Sinkhorn-Knopp Algorithm** [link](#)
- ▶ **Learning Latent Permutations with Gumbel-Sinkhorn Networks** [link](#)