Data-driven Logic

or: type theory for the working engineer

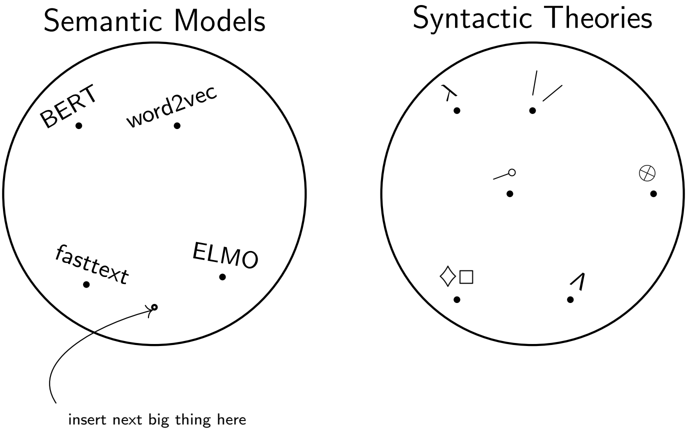
Kokos

Logic & Language ~~2020~~2021[1]

16/12/21

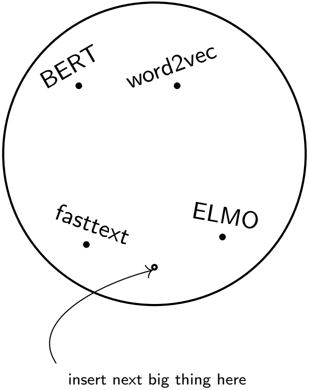
[1] totally not the same slides as last year

The state of NLP affairs



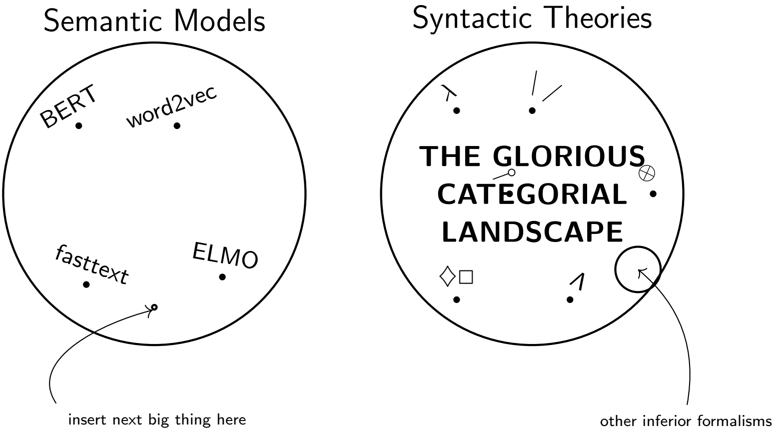
The state of NLP affairs

Semantic Models

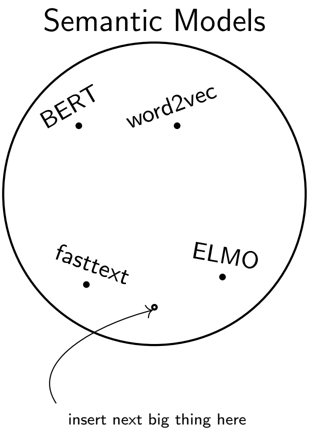


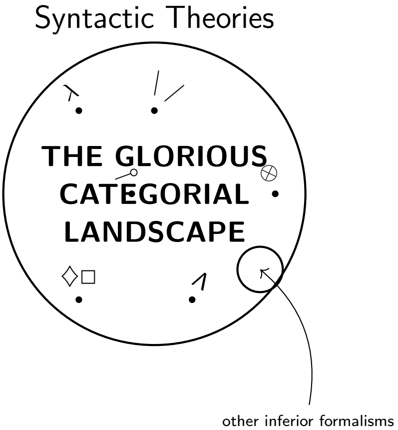
Syntactic Theories

The state of NLP affairs



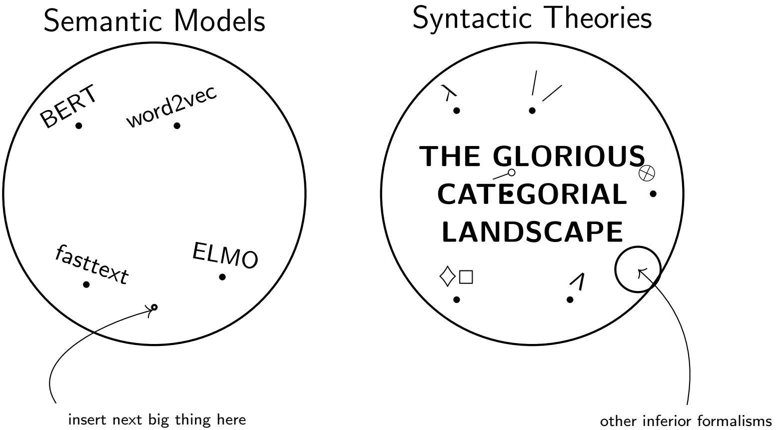
The state of NLP affairs





What's missing?

The state of NLP affairs



What's missing?

No (working) unifying theory or application.

Neural Type-Driven Representations

The agenda:

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations

Neural Type-Driven Representations

The agenda:

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations

Syntax-Semantics Interface

Type-logical perspective

$$\begin{tikzcd}
\begin{array}{c}\text{Syntax}\\ \mathrm{(N)L}_{(\diamond,\Box)}\end{array} \arrow[d, "h_{der}"] \\
\begin{array}{c}\text{Derivational Semantics}\\ \mathrm{ILL}_{\multimap}\end{array} \arrow[d, "h_{lex}"] \\
\begin{array}{c}\text{Lexical Semantics}\\ \mathrm{IL}_{\rightarrow}\end{array}
\end{tikzcd}$$

Syntax-Semantics Interface

Type-logical perspective

$$
\begin{tikzcd}
\begin{array}{c}\text{Syntax}\\ \text{(N)L}_{(\diamond,\Box)}\end{array} \arrow[d, "h_{der}"] \\
\begin{array}{c}\text{Derivational Semantics}\\ \text{ILL}_{\multimap}\end{array} \arrow[d, "h_{lex}"] \\
\begin{array}{c}\text{Lexical Semantics}\\ \text{IL}_{\rightarrow}\end{array}
\end{tikzcd}
$$

ACG perspective

$$
\begin{tikzcd}
& \begin{array}{c}\text{Tectogrammar}\\ \text{ILL}_{\multimap}\end{array} \arrow[dl, leftrightarrow, "h_{syn}"'] \arrow[dr, leftrightarrow, "h_{sem}"] & \\
\begin{array}{c}\text{Syntax}\\ \text{ILL}_{\multimap}\end{array} & & \begin{array}{c}\text{Semantics}\\ \text{IL}_{\rightarrow}\end{array}
\end{tikzcd}
$$

A twist

Now

Tectogrammar
$\mathrm{ILL}_{\multimap,\Diamond,\Box}$

Syntax
???

CHC

Derivational Semantics
$\mathrm{ILL}_{\multimap,\Diamond,\Box}$

Lexical Semantics
???

Abstract syntax with NLP

Grammar

ILL$_{\multimap}$ plus $\Diamond, \Box$ modalities for *dependency domain demarkation*.

Types inductively defined by:

$$\mathcal{T} := A \mid T \multimap T' \mid \Diamond^d T \mid \Box^d T \qquad A \in \mathcal{A}, T \in \mathcal{T}$$

Rules

$$\frac{\Gamma \vdash \mathtt{s} : A \multimap B \quad \Delta \vdash \mathtt{t} : A}{\Gamma, \Delta \vdash \mathtt{s}\ \mathtt{t} : B} \multimap E \qquad \frac{\Gamma, \mathtt{x} : A \vdash \mathtt{s} : B}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{s} : A \multimap B} \multimap I$$

$$\frac{\Gamma \vdash \mathtt{t} : A}{\langle \Gamma \rangle^d \vdash \triangle^d\ \mathtt{t} : \Diamond^d A} \Diamond^d I \qquad \frac{Y \vdash \mathtt{s} : \Diamond^d A \quad X[\langle \mathtt{x} : A \rangle^d] \vdash \mathtt{t} : B}{X[Y] \vdash \mathtt{t}[\triangledown^d \mathtt{s}/\mathtt{x}] : B} \Diamond^d E$$

$$\frac{\langle X \rangle^d \vdash \mathtt{s} : A}{X \vdash \blacktriangle^d \mathtt{s} : \Box^d A} \Box^d I \qquad \frac{X \vdash \mathtt{s} : \Box^d A}{\langle X \rangle^d \vdash \blacktriangledown^d \mathtt{s} : A} \Box^d E$$

Abstract syntax with NLP

Lexicon $\mathcal{L}$ assigning words types from:

Abstract syntax with NLP

Lexicon $\mathcal{L}$ assigning words types from: A

animals, ducks, I : np

$$\frac{\text{ducks}}{\texttt{ducks} : np}\ \mathcal{L}$$

Abstract syntax with NLP

Lexicon $\mathcal{L}$ assigning words types from: $A \mid \Diamond^{d} T \multimap T'$

$$\begin{array}{rcl} \text{animals, ducks, I} & : & np \\ \text{like} & : & \Diamond^{obj} np \multimap \Diamond^{obj} np \multimap s \end{array} \qquad \begin{array}{rcl} \text{fly, swim} & : & \Diamond^{su} np \multimap s \end{array}$$

$$\dfrac{\dfrac{\text{fly}}{\Diamond^{su} np \multimap s}\ \mathcal{L} \qquad \dfrac{\dfrac{\text{ducks}}{np}\ \mathcal{L}}{\langle\text{ducks}\rangle^{su} \vdash \Diamond^{su} np}\ \Diamond^{su} I}{\langle\text{ducks}\rangle^{su}\ \text{fly} \vdash s}\ \multimap E$$

$\texttt{fly}\ \triangle^{su}\ \texttt{ducks}$

Abstract syntax with NLP

Lexicon $\mathcal{L}$ assigning words types from: $A \mid \Diamond^d T \multimap T' \mid \Box^d (T \multimap T')$

$$\begin{array}{rcl} \text{animals, ducks, I} & : & np \\ \text{like} & : & \Diamond^{obj} np \multimap \Diamond^{obj} np \multimap s \end{array} \qquad \begin{array}{rcl} \text{fly, swim} & : & \Diamond^{su} np \multimap s \\ \text{gracefully} & : & \Box^{mod} (s \multimap s) \end{array}$$

$$\dfrac{\dfrac{\dfrac{\text{gracefully}}{\Box^{mod}(s \multimap s)}\ \mathcal{L}}{\langle \text{gracefully} \rangle^{mod} \vdash s \multimap s}\ \Box^{mod} E \qquad \dfrac{\vdots}{\langle \text{ducks} \rangle^{su}\ \text{fly} \vdash s}}{\langle \text{ducks} \rangle^{su}\ \text{fly}\ \langle \text{gracefully} \rangle^{mod} \vdash s}\ \multimap E$$

$$\blacktriangledown^{mod}\texttt{gracefully}\ (\texttt{fly}\ \ \triangle^{su}\ \texttt{ducks})$$

Abstract syntax with NLP

Lexicon $\mathcal{L}$ assigning words types from: $A \mid \Diamond^{d}\, T \multimap T' \mid \Box^{d}\,(T \multimap T')$

$$
\begin{array}{rcl}
\text{animals, ducks, I} & : & np \\
\text{like} & : & \Diamond^{obj}\, np \multimap \Diamond^{obj}\, np \multimap s \\
\text{that} & : & \Diamond^{body}\,(\Diamond^{su}\, np \multimap s) \multimap \Box^{mod}\,(np \multimap np)
\end{array}
\qquad
\begin{array}{rcl}
\text{fly, swim} & : & \Diamond^{su}\, np \multimap s \\
\text{gracefully} & : & \Box^{mod}\,(s \multimap s)
\end{array}
$$

$$
\dfrac{\dfrac{\text{animals}}{np}\,\mathcal{L} \qquad \dfrac{\dfrac{\dfrac{\text{that}}{\Diamond^{body}\,(\Diamond^{obj}\, np \multimap s) \multimap \Box^{mod}\,(np \multimap np)}\,\mathcal{L} \qquad \dfrac{\dfrac{\vdots}{\langle \mathrm{I} \rangle^{su}\ \text{like} \vdash \lambda x.(\texttt{like}\ x\ \ \triangle^{su} I) : \Diamond^{obj}\, np \multimap s}}{\langle \langle \mathrm{I} \rangle^{su}\ \text{like} \rangle^{body} \vdash \Diamond^{body}\,(\Diamond^{obj}\, np \multimap s)}\,\Diamond^{body} I}{\text{that}\ \langle \langle \mathrm{I} \rangle^{su}\ \text{like} \rangle^{body} \vdash \Box^{mod}(np \multimap np)}\,\multimap E}{\langle \text{that}\ \langle \langle \mathrm{I} \rangle^{su}\ \text{like} \rangle^{body} \rangle^{mod} \vdash np \multimap np}\,\Box^{mod} E}{\text{animals}\ \langle \text{that}\ \langle \langle \mathrm{I} \rangle^{su}\ \text{like} \rangle^{body} \rangle^{mod} \vdash np}\,\multimap E
$$

$$
\blacktriangledown^{mod} \left(\texttt{that}\ \ \triangle^{body}\ (\lambda x.(\texttt{like x}\ \ \triangle^{su} I))\right)\ \texttt{animals}
$$

Why $ILL_{\multimap,\Diamond,\Box}$?

Why $ILL_{\multimap}$?

- Easier to extract from corpora
- Massive reduction in lexical ambiguity (more later)
- Abstract away from trivial word-order permutations
- Surface syntax matters little to semantics

Why $\mathrm{ILL}_{\multimap,\diamond,\square}$?

Why $\mathrm{ILL}_{\multimap}$?

- Easier to extract from corpora
- Massive reduction in lexical ambiguity (more later)
- Abstract away from trivial word-order permutations
- Surface syntax matters little to semantics

Why $\diamond, \square$?

- More interpretation options
- Subsume dependency parsing
- More informative for semantics
- Modalities can regulate non-logical parsing

Intermezzo: Proof Nets

Proof Nets

A graphical, diagrammatical representation of (intuitionitic) linear logic proofs.

Intermezzo: Proof Nets

Formula decomposition

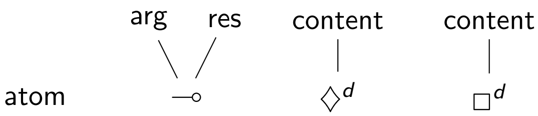
Type formation rules $\equiv$ tree constructors

$$
\begin{array}{lccccccc}
 & & & \text{arg} \quad \text{res} & & \text{content} & & \text{content} \\
 & \text{atom} & & \multimap & & \Diamond^d & & \Box^d \\
\mathcal{T} := & A & \mid & T \multimap T' & \mid & \Diamond^d T & \mid & \Box^d T
\end{array}
$$

Intermezzo: Proof Nets

Formula decomposition

Type formation rules $\equiv$ tree constructors



$$\mathcal{T} := \quad A \quad | \quad T \multimap T' \quad | \quad \Diamond^d T \quad | \quad \Box^d T$$

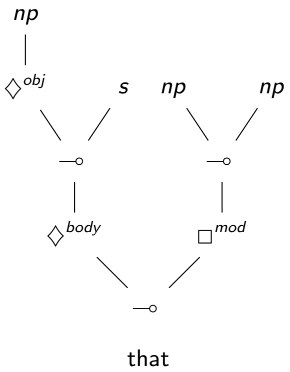
or, overloading $\Diamond$ and $\Box$ as binary operators for brevity

Intermezzo: Proof Nets

Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

$\multimap$ is polarity preserving for the result, reversing for the argument

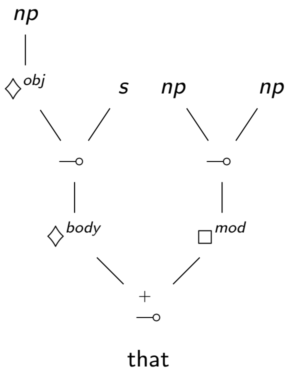


Intermezzo: Proof Nets

Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

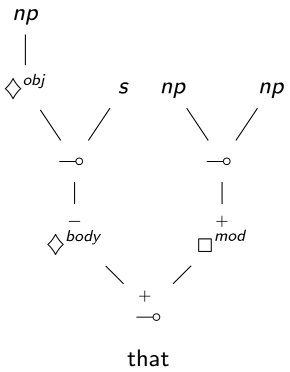
$\multimap$ is polarity preserving for the result, reversing for the argument



Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

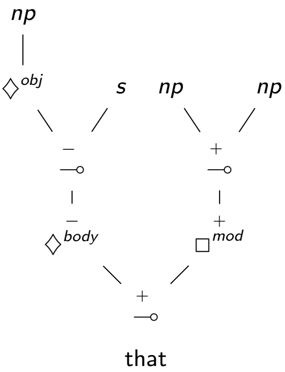
$\multimap$ is polarity preserving for the result, reversing for the argument



Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

- $\multimap$ is polarity preserving for the result, reversing for the argument

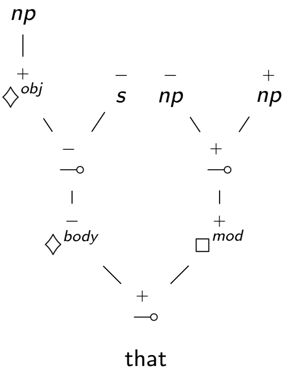


Intermezzo: Proof Nets

Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

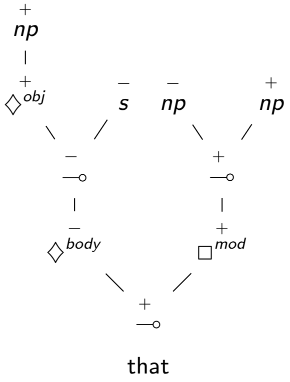
$\multimap$ is polarity preserving for the result, reversing for the argument



Tree Polarization

Let $+$ stand for resources we *have*, $-$ for ones we seek:

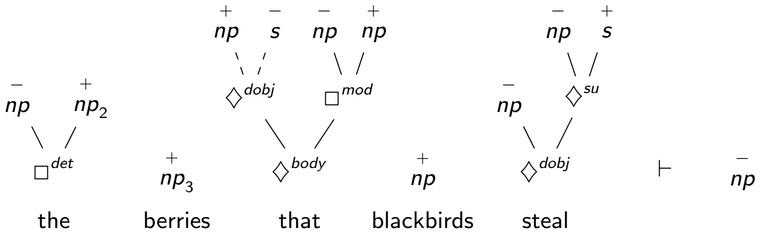
$\multimap$ is polarity preserving for the result, reversing for the argument



Intermezzo: Proof Nets

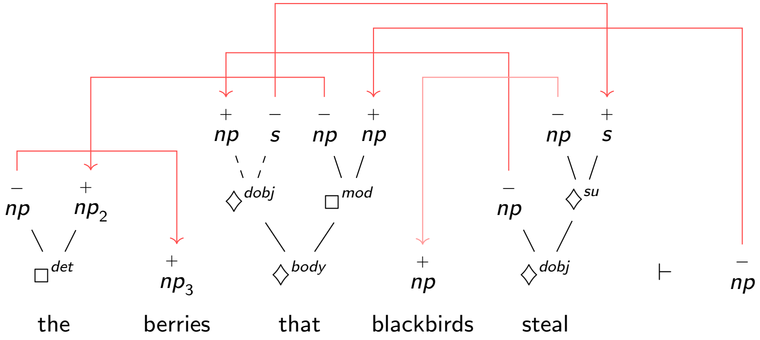
Proof Frame

A sequence of polarized unfolded formulas



Proof Structure

A proof net together with **axiom links**: bijection between pos/neg atoms



Intermezzo: Proof Nets

Traversing a Proof Net

positive (down) mode: start from positive leaf, follow positive nodes to subtree root

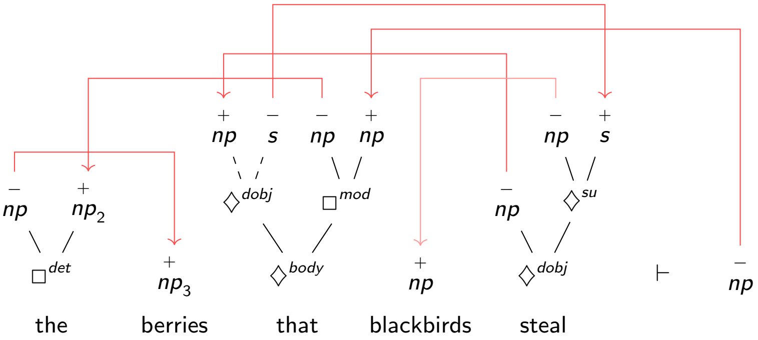
- $\Box$ wrap future with $\blacktriangledown$
- $\Diamond$ wrap variable (context) with $\triangledown$
- $\multimap$ each is an application
  on reaching root, write word or variable name
  and perform negative traversal on negative child of each $\multimap$

negative (up) mode: start from negative root, follow negatives to subtree leaf

- $\Box$ wrap future with $\blacktriangle$
- $\Diamond$ wrap future with $\triangle$
- $\multimap$ each is an abstraction: assign a fresh variable to positive subtree
  on reaching a leaf, cross the axiom link and switch to positive mode

start from conclusion in negative mode

Intermezzo: Proof Nets



From parse graphs to $\text{ILL}_{\multimap,\diamond,\square}$ types

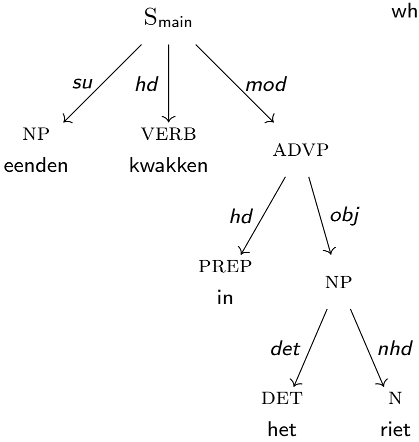
algorithm: graph flooding on dags

init with maps

- from pos & phrasal categories to $\mathcal{A}$
  e.g. $\text{NP} \rightarrow \mathit{np}$, $\text{INF} \rightarrow \mathit{inf}, \ldots$
- from grammatical roles to $\diamond$ (complements) and $\square$ (adjuncts)
  e.g. $\mathit{su} \rightarrow \diamond^{\mathit{su}}, \mathit{obj} \rightarrow \diamond^{\mathit{obj}}, \ldots, \mathit{mod} \rightarrow \square^{\mathit{mod}}, \mathit{det} \rightarrow \square^{\mathit{det}}$

and a strict total order over $\diamond$,
e.g. $\diamond^{\mathit{su}} > \diamond^{\mathit{obj}}$

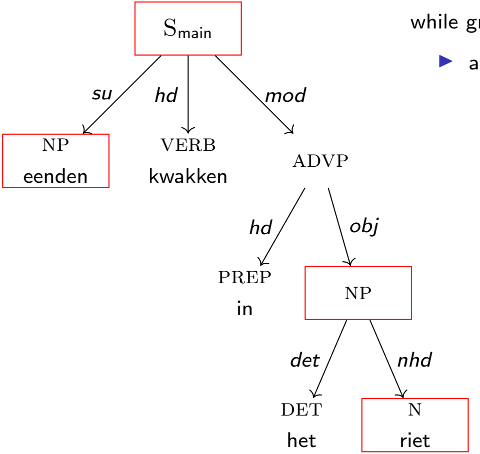
Simple case: trees



"eenden kwakken in het riet"

while graph not fully typed, do:

Simple case: trees

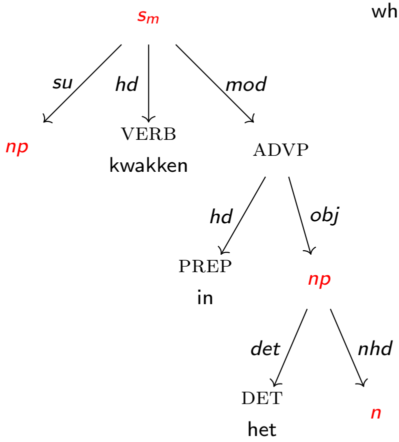


“eenden kwakken in het riet”

while graph not fully typed, do:

- assign stand-alone nodes
  no incoming adjunct or head edge

Simple case: trees

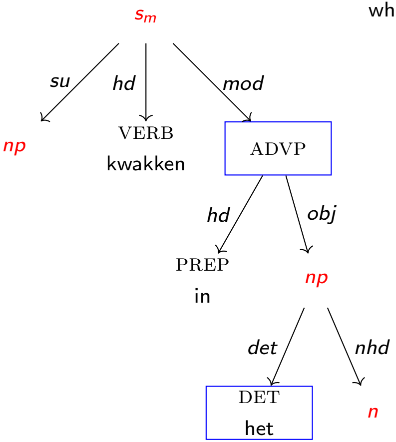


"eenden kwakken in het riet"

while graph not fully typed, do:

- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map

Simple case: trees

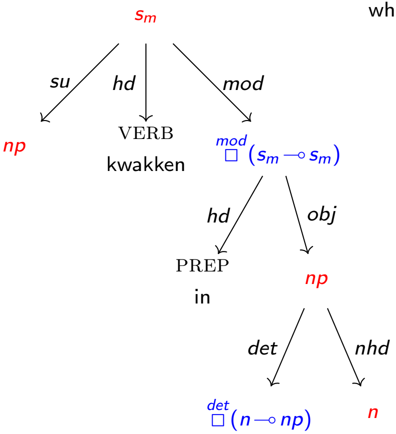


“eenden kwakken in het riet”

while graph not fully typed, do:

- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed

Simple case: trees

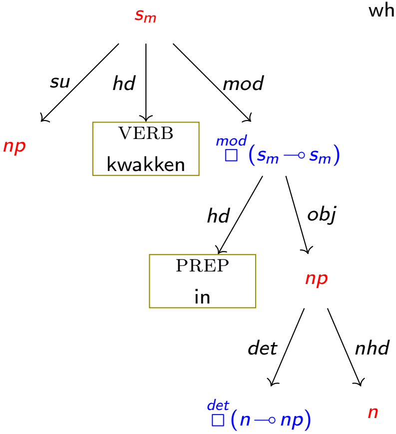


"eenden kwakken in het riet"

while graph not fully typed, do:

- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed
  type
  if mod: boxed endofunctor of parent
  else:from comp sibs to parent

Simple case: trees

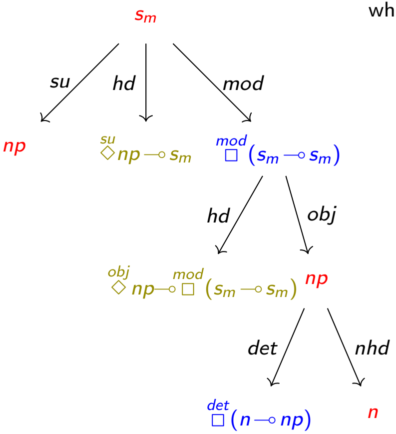


"eenden kwakken in het riet"

while graph not fully typed, do:

- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed
  type
  if mod: boxed endofunctor of parent
  else:from comp sibs to parent
- assign heads
  incoming head edge
  parent is typed
  no untyped complement sibs

Simple case: trees

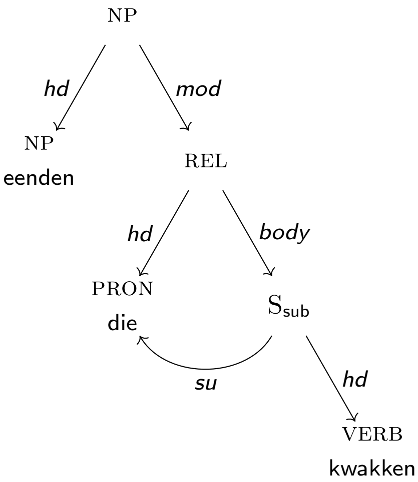


"eenden kwakken in het riet"

while graph not fully typed, do:

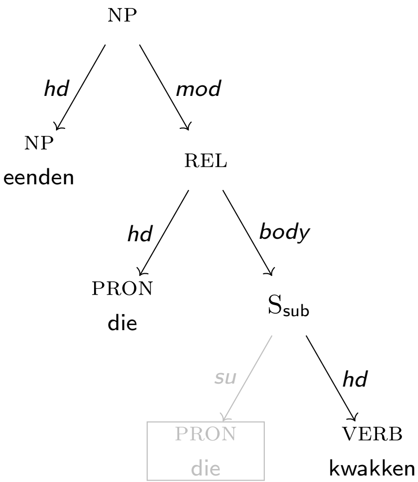
- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed
  type
  if mod: boxed endofunctor of parent
  else:from comp sibs to parent
- assign heads
  incoming head edge
  parent is typed
  no untyped complement sibs
  type from comp sibs to parent

Harder case: unbounded dependencies



"eenden die kwakken"

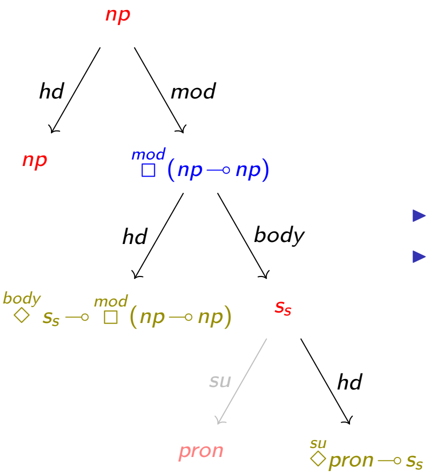
Harder case: unbounded dependencies



"eenden die kwakken"

- detach non-local dependencies

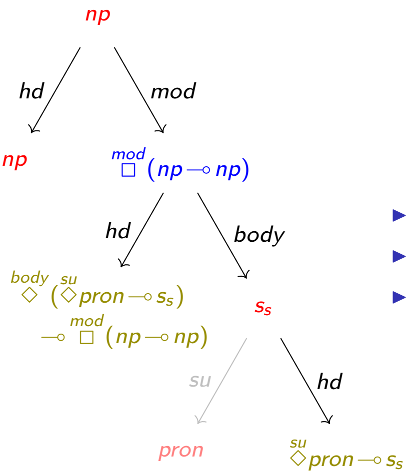
Harder case: unbounded dependencies



- detach non-local dependencies
- type trees as before

"eenden die kwakken"

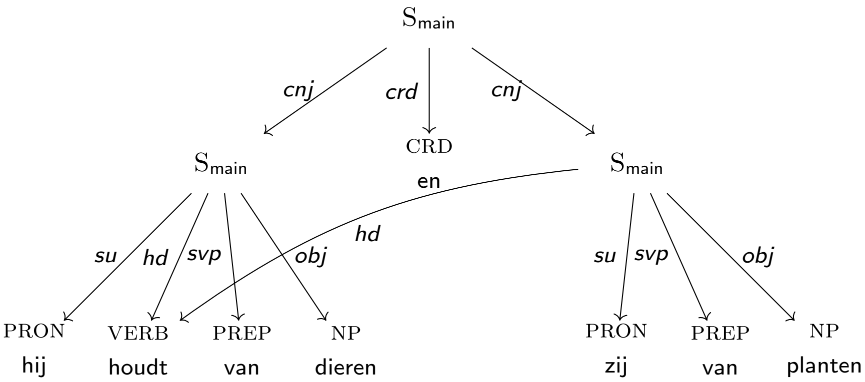
Harder case: unbounded dependencies



"eenden die kwakken"

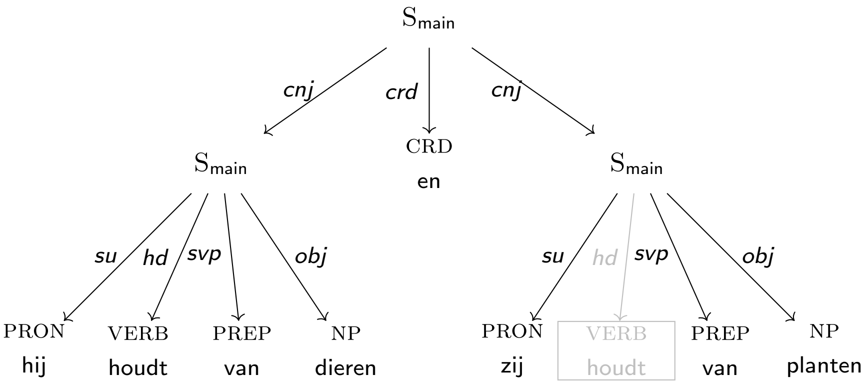
- detach non-local dependencies
- type trees as before
- redact ghosts types from comp sibs

Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

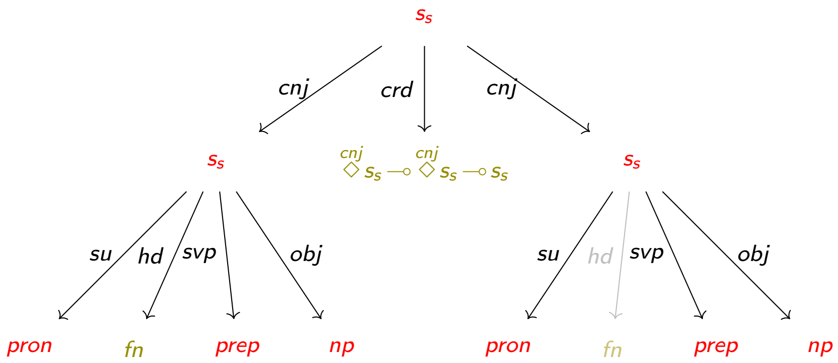
Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

- detach and type trees as usual

Hardest case: Ellipses

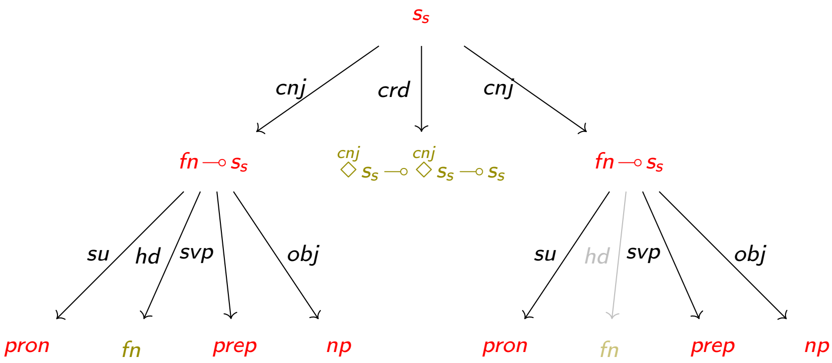


"hij houdt van dieren en zij van planten"

- detach and type trees as usual

$$fn := \overset{svp}{\diamond} prep \multimap \overset{obj}{\diamond} np \multimap \overset{su}{\diamond} pron \multimap s_s$$

Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

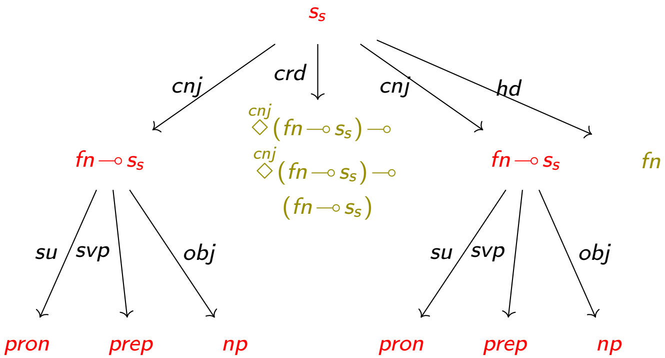
- detach and type trees as usual
- redact missing types from **both** conjuncts

$$fn := \overset{svp}{\diamond} prep \multimap \overset{obj}{\diamond} np \multimap \overset{su}{\diamond} pron \multimap s_s$$

ACG Flashback

- each conjunct represents a tuple of types
  $c = (t_1, t_2, \ldots t_n) \equiv t_1 \otimes t_2 \otimes \ldots \otimes t_n$
- encoded as the higher-order function $(c \multimap r) \multimap r$ and curried into
  $(t_1 \multimap t_2 \multimap \ldots \multimap t_n \multimap r) \multimap r$

Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

- detach and type trees as usual
- redact missing types from **both** conjuncts
- update coord type & attach copies at top level

$$fn := \overset{svp}{\Diamond} prep \multimap \overset{obj}{\Diamond} np \multimap \overset{su}{\Diamond} pron \multimap s_s$$

A glimpse at a higher universe

Second-order IL (system F or polymorphic λ-calculus)

$$\frac{\Gamma \vdash \mathtt{M} : \forall\alpha.\sigma}{\Gamma \vdash \mathtt{M}\tau : \sigma[\tau/\alpha]} \qquad \frac{\Gamma \vdash \mathtt{M} : \sigma}{\Gamma \vdash \Lambda\alpha.\mathtt{M} : \forall\alpha.\sigma}$$

A glimpse at a higher universe

Second-order IL (system F or polymorphic λ-calculus)

$$\frac{\Gamma \vdash \mathtt{M} : \forall\alpha.\sigma}{\Gamma \vdash \mathtt{M}\tau : \sigma[\tau/\alpha]} \qquad \frac{\Gamma \vdash \mathtt{M} : \sigma}{\Gamma \vdash \Lambda\alpha.\mathtt{M} : \forall\alpha.\sigma}$$

In that universe, modifiers and coordinators are polymorphic types:

$$\mathsf{mod} := \Lambda\alpha.\mathtt{w} : \forall\alpha.\Box^{mod}(\alpha \multimap \alpha)$$

and

$$\mathsf{crd} := \Lambda\alpha.\mathtt{w} : \forall\alpha.\Diamond^{cnj}\alpha \multimap \Diamond^{cnj}\alpha \multimap \alpha$$

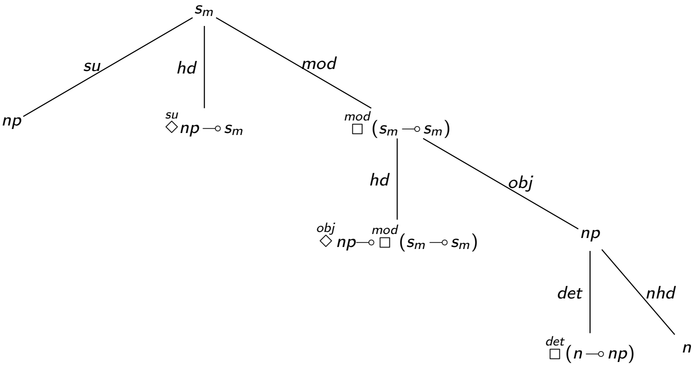
Coordinators as derived types

Elliptical coordinators can also be seen as a transformation of basic types.
If $c = (t_1 \otimes t_2 \otimes \ldots \otimes t_N)$ the conjoined tuples,

$$
\begin{aligned}
crd &= c \multimap c \multimap c \\
&\xrightarrow{vr} c \multimap c \multimap (c \multimap s) \multimap s \\
&\xrightarrow{ar^0} ((c \multimap s) \multimap s) \multimap c \multimap (c \multimap s) \multimap s \\
&\xrightarrow{ar^1} ((c \multimap s) \multimap s) \multimap ((c \multimap s) \multimap s) \multimap (c \multimap s) \multimap s \\
&\equiv ((t_1 \multimap t_2 \multimap \ldots \multimap t_n \multimap s) \multimap s) \multimap \\
&\qquad ((t_1 \multimap t_2 \multimap \ldots \multimap t_n \multimap s) \multimap s) \multimap \\
&\qquad\quad (t_1 \multimap t_2 \multimap \ldots \multimap t_n \multimap s) \multimap s
\end{aligned}
$$

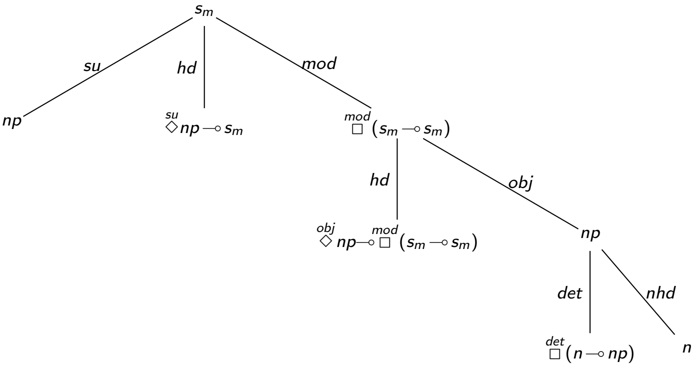
$$
\begin{aligned}
&\Diamond^{cnj} S_{main} \multimap \Diamond^{cnj} S_{main} \multimap S_{main} \\
\xrightarrow{vr^*} &\Diamond^{cnj} S_{main} \multimap \Diamond^{cnj} S_{main} \multimap fn \multimap S_{main} \\
\xrightarrow{ar^1} &\Diamond^{cnj} (fn \multimap S_{main}) \multimap \Diamond^{cnj} S_{main} \multimap fn \multimap S_{main} \\
\xrightarrow{ar^2} &\Diamond^{cnj} (fn \multimap S_{main}) \multimap \Diamond^{cnj} (fn \multimap S_{main}) \multimap fn \multimap S_{main}
\end{aligned}
$$

From graphs to proofs



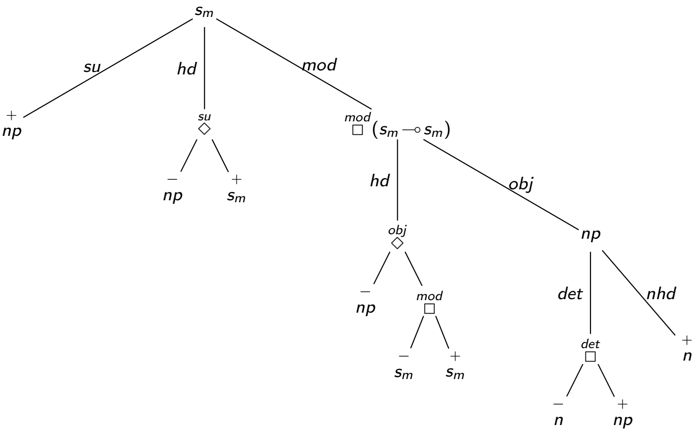
given a typed graph:

From graphs to proofs



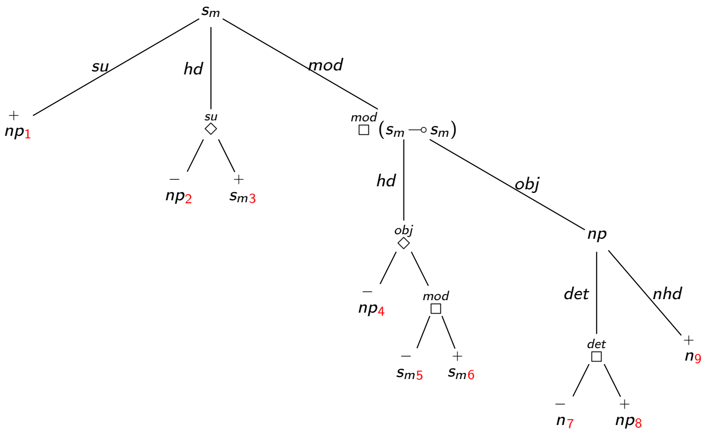
(1) convert types to binary trees and assign polarities

From graphs to proofs



(2) assign identifying indices

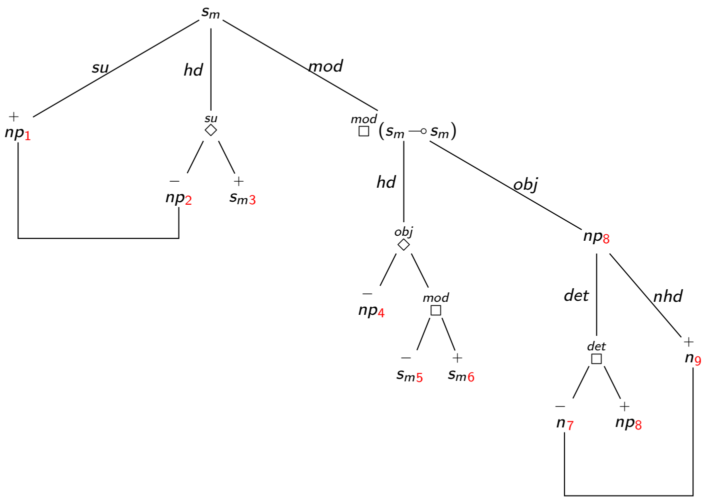
From graphs to proofs



(3) traverse upwards, identifying pos/neg atoms and propagating indices

$$\{2 \mapsto ?, 4 \mapsto ?, 5 \mapsto ?, 7 \mapsto ?\}$$

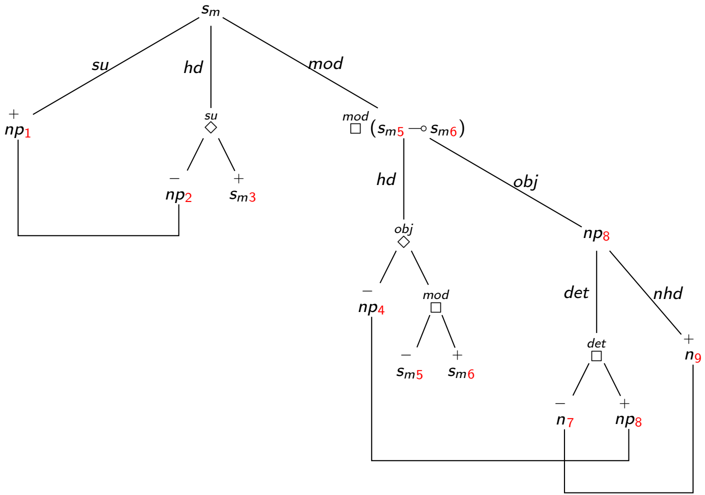
From graphs to proofs



(3) traverse upwards, identifying pos/neg atoms and propagating indices

$$\{2 \mapsto 1, 4 \mapsto ?, 5 \mapsto ?, 7 \mapsto 9\}$$

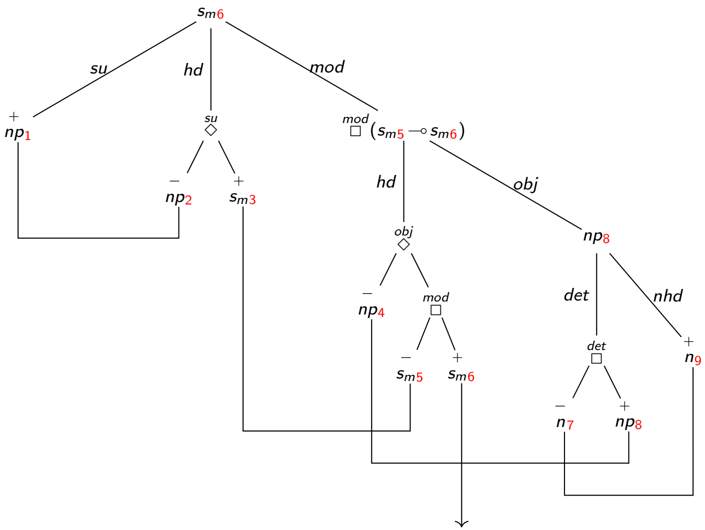
From graphs to proofs



(3) traverse upwards, identifying pos/neg atoms and propagating indices

$$\{2 \mapsto 1, 4 \mapsto 8, 5 \mapsto ?, 7 \mapsto 9\}$$

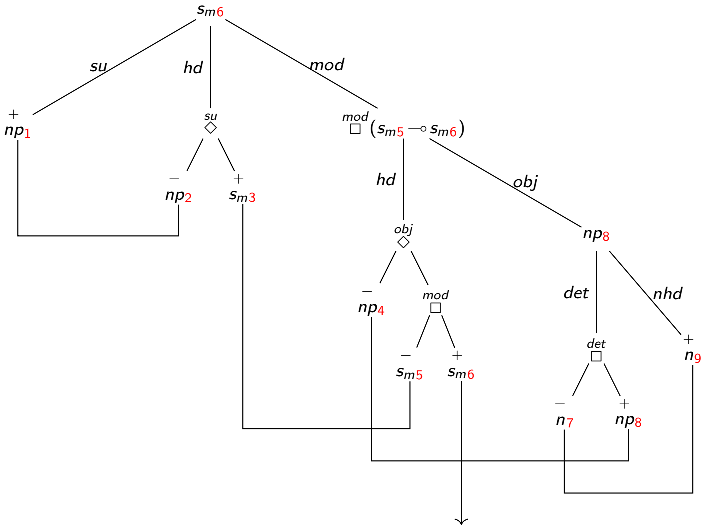
From graphs to proofs



(3) traverse upwards, identifying pos/neg atoms and propagating indices

$$\{2 \mapsto 1, 4 \mapsto 8, 5 \mapsto 3, 7 \mapsto 9\}$$

From graphs to proofs



the resulting structure is a *proof net*

$$\blacktriangledown^{mod}\left(in\ \triangle^{obj}\left(\blacktriangledown^{det} het\ riet\right)\right)\ (kwakken\ \triangle^{su} eenden)$$

..continuing

The agenda:

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations

Lexical type ambiguity

Type assignments are often *ambiguous* and context-dependent:

very realistic lexicon		
running ::	$\overset{mod}{\Box}(np \multimap np)$	*running dog*
	inf	*I like running*
like ::	$\overset{obj}{\Diamond} np \multimap \overset{su}{\Diamond} np \multimap s_m$	*ducks like seeds*
	$\overset{obj}{\Diamond} np \multimap \overset{su}{\Diamond} pron \multimap s_m$	*I like ducks*
	$\overset{obj}{\Diamond} inf \multimap \overset{su}{\Diamond} np \multimap s_m$	*ducks like swimming*
	$\overset{obj}{\Diamond} inf \multimap \overset{su}{\Diamond} pron \multimap s_m$	*I like swimming*
	$\overset{obj}{\Diamond} np \multimap \overset{mod}{\Box}(s_m \multimap s_m)$	*I swim like a duck*
	$\ldots$	

Lexical type ambiguity

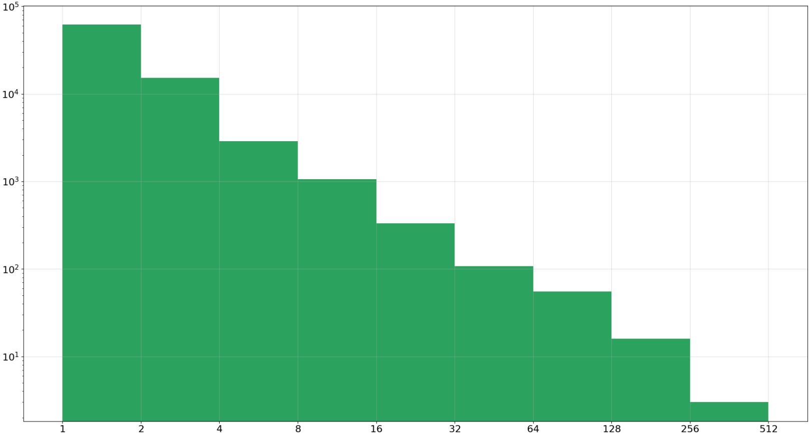
generally:

$$w :: t_1 \& t_2 \& t_3 \& \ldots \& t_n$$

more refined grammar $\Longrightarrow$

- ☹ harder lexical disambiguation (more n per word)
- ☺ easier parsing (*if* one starts from correct type)

Lexical type ambiguity



\# types (*log2-transformed*) per word

Supertagging

General idea

Given a sentence $w_1, w_2, \ldots, w_n$
find the type sequence $t_1, t_2, \ldots t_n$ that maximimizes

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

where θ the trainable parameter space

Paleolithic ages

Indipendence Assumption

$$max(p(t_1, t_2, \dots t_n | w_1, w_2, \dots w_n, \theta)) \approx \prod_{i=1}^{n} max(p(t_i, w_i, \theta))$$

Implemented as naive bayes/maximum entropy models, later using feed-forward networks on distributional vectors

- ☺ Beats hand-writing a lexicon
- ☹ Small (if any) context/receptive field
- ☹ No domain generalization
- ☹ Severe sample sparsity when using windows

Discriminative approach

Markov assumption

$$p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta) \approx \prod_{i=1}^{n} p(t_i | w_1, w_2, \ldots w_n, \theta)$$

Implemented as contextualized token classification using RNNs

- ☺ global context (lossy)
- ☺ domain generalization
- ☹ single answer
- ☹ sample sparsity
- ☹ no codomain generalization (closed world assumption)

Generative approach

~~Markov assumption~~

$$p(t_1, t_2, \dots t_n | w_1, w_2, \dots w_n, \theta)$$
$$\approx \prod_{i=1}^{n} p(t_i | t_1, t_2, \dots, t_{i-1}, w_1, w_2, \dots w_n, \theta)$$

(could be) implemented as a transducer/seq2seq model

- ☺ global context (lossless)
- ☺ many answers
- ☹ sample sparsity
- ☹ no codomain generalization

Generative approach: one more step

The type syntax:

$$cod(\mathcal{L}) := A \mid \overset{d}{\Diamond} T \multimap T' \mid \overset{d}{\Box}(T \multimap T')$$

corresponds to a simple cfg (in prefix notation) with meta-rules:

$$\begin{aligned} S &\rightarrow A & \forall A \in \mathcal{A} \\ S &\rightarrow \overset{d}{\Diamond} S\ S & \forall d \in \mathrm{comps} \\ S &\rightarrow \overset{d}{\Box} S\ S & \forall d \in \mathrm{adjns} \end{aligned}$$

i.e. each type t_i can be written as a sequence of primitive symbols $\vec{\sigma}$

Generative approach: one more step

The type syntax:

$$cod(\mathcal{L}) := A \mid \overset{d}{\Diamond} T \multimap T' \mid \overset{d}{\Box}(T \multimap T')$$

corresponds to a simple cfg (in prefix notation) with meta-rules:

$$\begin{aligned} S &\to A && \forall A \in \mathcal{A} \\ S &\to \overset{d}{\Diamond} S\ S && \forall d \in \mathrm{comps} \\ S &\to \overset{d}{\Box} S\ S && \forall d \in \mathrm{adjns} \end{aligned}$$

i.e. each type t_i can be written as a sequence of primitive symbols $\vec{\sigma}$

$$\begin{aligned} & p(t_1, t_2, \ldots t_n | w_1, w_2, \ldots w_n, \theta) \\ \approx & \prod_{i=1}^{n} p(\sigma_i | \sigma_1, \sigma_2, \ldots, \sigma_{i-1}, w_1, w_2, \ldots w_n, \theta) \end{aligned}$$

- ☺ each type contains many subtypes (less sparsity)
- ☺ any valid types can be inductively constructed (open codomain)

Supertagging today

? types are **trees** – but tree transduction requires trees decoded in isolation

Supertagging today

- types are **trees** – but tree transduction requires trees decoded in isolation
- longer sequence length $\Longrightarrow$ memory and time cost, degraded beam search

Supertagging today

- types are **trees** – but tree transduction requires trees decoded in isolation
- longer sequence length $\implies$ memory and time cost, degraded beam search
- harder to train: the system has to learn a grammar within the grammar

Supertagging today

- types are **trees** – but tree transduction requires trees decoded in isolation
- longer sequence length $\implies$ memory and time cost, degraded beam search
- harder to train: the system has to learn a grammar within the grammar
- left-right decoding does not make use of *easy* types first

Supertagging today

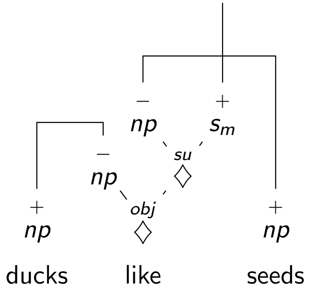
- types are **trees** – but tree transduction requires trees decoded in isolation
- longer sequence length $\implies$ memory and time cost, degraded beam search
- harder to train: the system has to learn a grammar within the grammar
- left-right decoding does not make use of *easy* types first
- logical constraints are not easy to integrate with neural decoding

Supertagging today

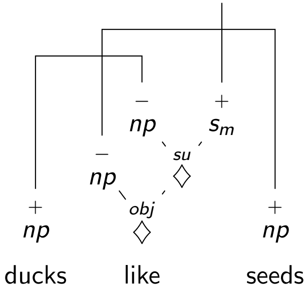
- ? types are **trees** – but tree transduction requires trees decoded in isolation
- ? longer sequence length $\Longrightarrow$ memory and time cost, degraded beam search
- ? harder to train: the system has to learn a grammar within the grammar
- ? left-right decoding does not make use of *easy* types first
- ? logical constraints are not easy to integrate with neural decoding
- ! new possibilities for parsing and meaning representation

Proof Ambiguity

The type system (being non-directional) permits too many parses:



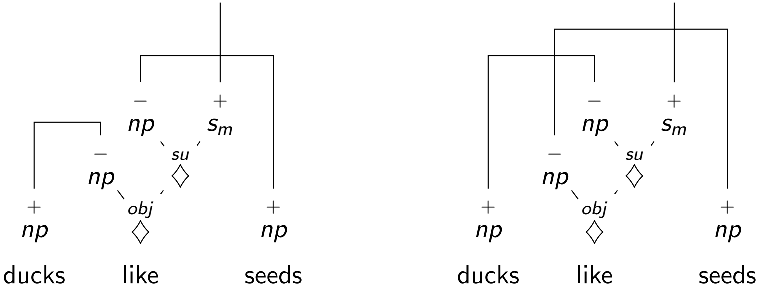
$\texttt{like (ducks)}^{\texttt{obj}}\ \texttt{(seeds)}^{\texttt{su}}$



$\texttt{like (seeds)}^{\texttt{obj}}\ \texttt{(ducks)}^{\texttt{su}}$

Proof Ambiguity

The type system (being non-directional) permits too many parses:



`like (ducks)`^{obj} `(seeds)`^{su} `like (seeds)`^{obj} `(ducks)`^{su}

How can we select the correct one?

Neural Proof Nets

the berries that blackbirds steal

1. pass the sentence through the supertagger

Neural Proof Nets

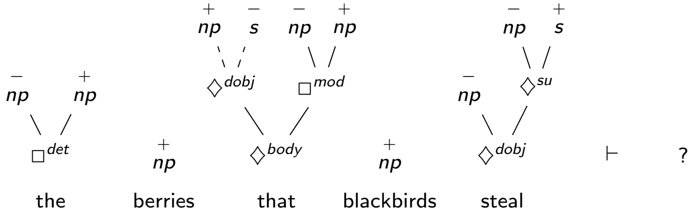
$$\Box^{det}, np, np, \#, np, \#, \Diamond^{body}, \Diamond^{dobj}, np, s, \Box^{mod}, np, np, \#, \Diamond^{dobj}, np, \Diamond^{su}, np, s$$

$\uparrow$

the berries that blackbirds steal

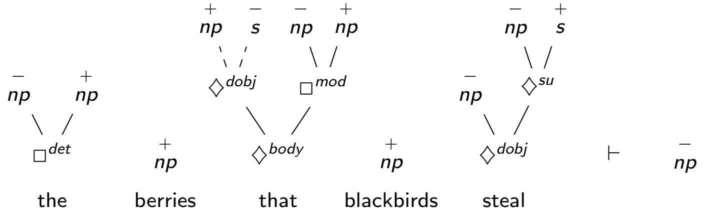
2. parse types to trees and assign polarity information

Neural Proof Nets



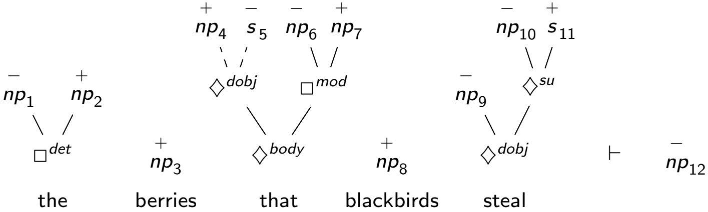
3. find conclusion as the singleton $\mathcal{A}^+ - \mathcal{A}^-$

Neural Proof Nets



4. index pos/neg occurrences and arrange in a table

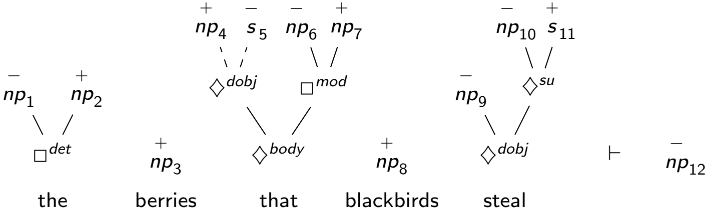
Neural Proof Nets



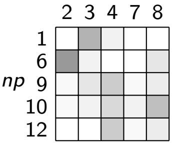
5. fill table with pair-wise agreement scores

		2	3	4	7	8
	1					
	6					
np	9					
	10					
	12					

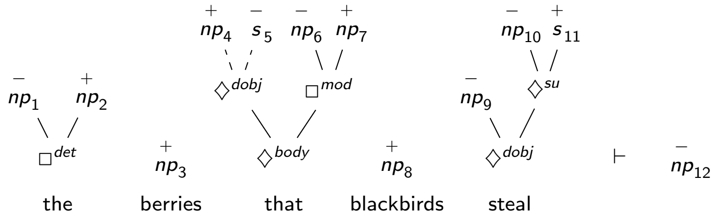
Neural Proof Nets



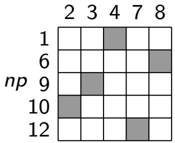
6. discretize result with Sinkhorn



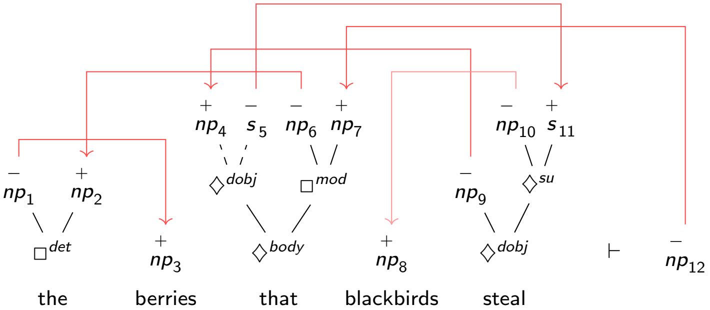
Neural Proof Nets



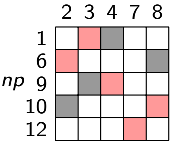
7. train against ground-truth axiom links



Neural Proof Nets



7. train against ground-truth axiom links



The future

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations

The future

- λ Choosing the logic
- λ Making a dataset: proofs and lexical type assignments
- λ Learning the type assignment process
- λ Navigating the proof space
- λ Syntax-aware & type-correct text representations
  - meaning representation from proofs: graph traversal/encoding
  - types as recipes/functions on word embeddings, pure interpretations of the system
  - ...
  - [your project/internship/thesis idea here]

diy: github.com/konstantinosKokos/neural-proof-nets