
And Now For Something
Completely Different

The State of Affairs

NLP in the last decade

Over-parameterized data-intensive unsupervised models

2008-2013	Compressed co-occurrence vectors, *n*-grams
2013-2016	"*word2vec*" era: neural vectors
2016-2018	*rnn* based language models (ELMo)
2018-2020	*transformer* based language models (GPT-2, BERT, ...)

The State of Affairs

NLP in the last decade

Over-parameterized data-intensive unsupervised models

2008-2013	Compressed co-occurrence vectors, *n*-grams
2013-2016	"*word2vec*" era: neural vectors
2016-2018	*rnn* based language models (ELMo)
2018-2020	*transformer* based language models (GPT-2, BERT, ...)

..where is syntax?

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

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

Syntax-Semantics Interface

Now

$$
\begin{tikzcd}
\begin{array}{c}\text{Tectogrammar}\\ \mathrm{ILL}_{\multimap,\Diamond,\Box}\end{array} \arrow[d, dotted, leftrightarrow] \arrow[dd, bend left, "\text{CHC}"] \\
\begin{array}{c}\text{Syntax}\\ ???\end{array} \arrow[d, dotted] \\
\begin{array}{c}\text{Derivational Semantics}\\ \mathrm{ILL}_{\multimap,\Diamond,\Box}\end{array} \arrow[d, dotted] \\
\begin{array}{c}\text{Lexical Semantics}\\ ???\end{array}
\end{tikzcd}
$$

Abstract syntax with NLP

Grammar

$\mathrm{ILL}_{\multimap}$ 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 : np

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

Abstract syntax with NLP

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

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

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

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

Abstract syntax with NLP

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

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

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

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

Abstract syntax with NLP

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

animals, ducks	:	np	fly, swim	:	$\Diamond^{su}\, np \multimap s$
like	:	$\Diamond^{obj}\, np \multimap \Diamond^{su}\, np \multimap s$	gracefully	:	$\Box^{mod}\,(s \multimap s)$
that	:	$\Diamond^{body}\,(\Diamond^{su}\, np \multimap s) \multimap \Box^{mod}\,(np \multimap np)$			

$$
\dfrac{\dfrac{\text{animals}}{np}\,\mathcal{L} \qquad \dfrac{\dfrac{\dfrac{\text{that}}{\Diamond^{body}\,(\Diamond^{su}\, np \multimap s) \multimap \Box^{mod}\,(np \multimap np)}\,\mathcal{L} \qquad \dfrac{\begin{array}{c}\vdots\\ \text{swim} \vdash \Diamond^{su}\, np \multimap s\end{array}}{\langle \text{swim} \rangle^{body} \vdash \Diamond^{body}\,(\Diamond^{su}\, np \multimap s)}\,\Diamond^{body} I}{\text{that } \langle \text{swim} \rangle^{su} \vdash \Box^{mod}(np \multimap np)}\multimap E}{\langle \text{that } \langle \text{swim} \rangle^{su} \rangle^{mod} \vdash np \multimap np}\,\Box^{mod} E}{\text{animals } \langle \text{that } \langle \text{swim} \rangle^{body} \rangle^{mod} \vdash np}\multimap E
$$

$$
\blacktriangledown^{mod}\left(\texttt{that}\ \ \triangle^{body}\,\lambda y.(\texttt{swim y})\right)
$$

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

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

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

Why $\mathrm{ILL}_{\multimap,\Diamond,\Box}$?

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

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

Why $\Diamond, \Box$?

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

From parse graphs to ILL$_{\multimap,\Diamond,\Box}$ types

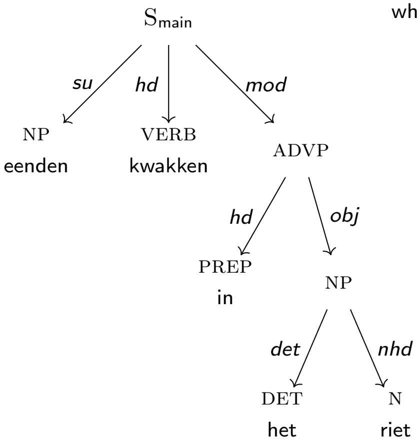
algorithm: graph flooding on dags

init with maps

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

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

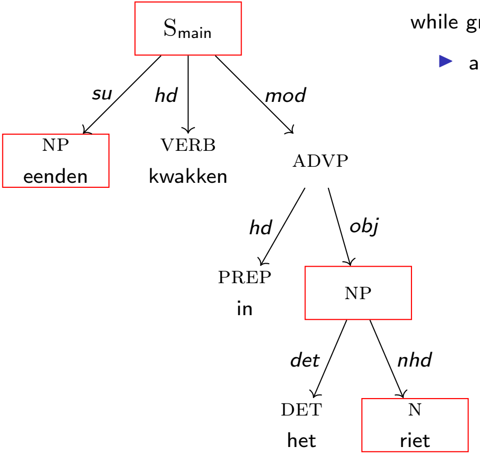
Simple case: trees



"eenden kwakken in het riet"

while graph not fully typed, do:

Simple case: trees

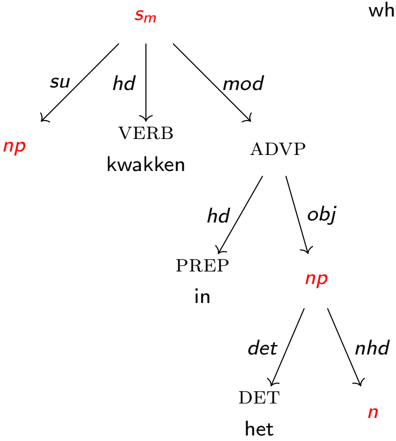


"eenden kwakken in het riet"

while graph not fully typed, do:

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

Simple case: trees

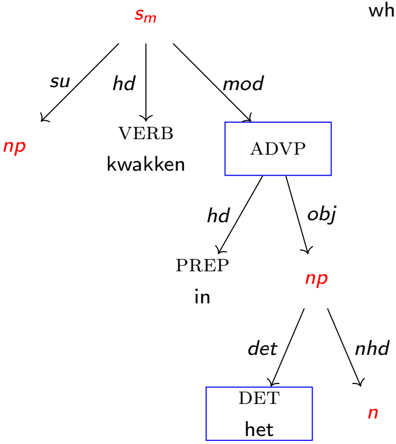


"eenden kwakken in het riet"

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

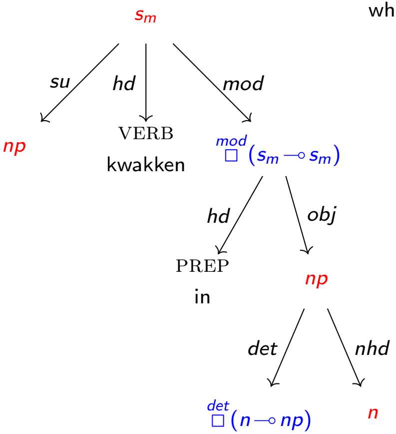


"eenden kwakken in het riet"

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

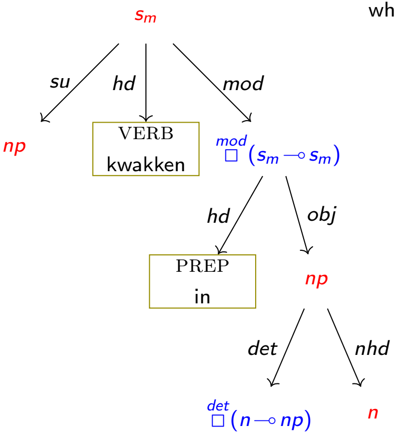


“eenden kwakken in het riet”

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

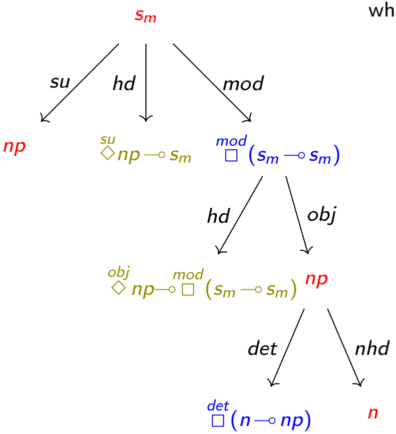


“eenden kwakken in het riet”

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

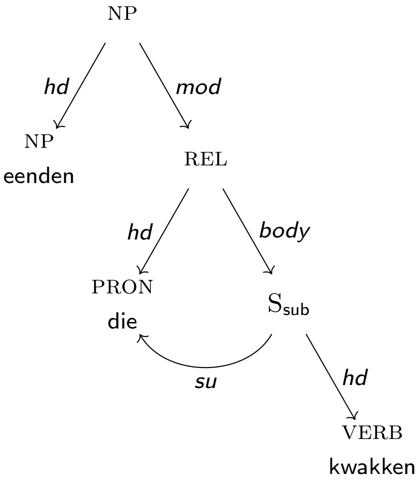


"eenden kwakken in het riet"

while graph not fully typed, do:

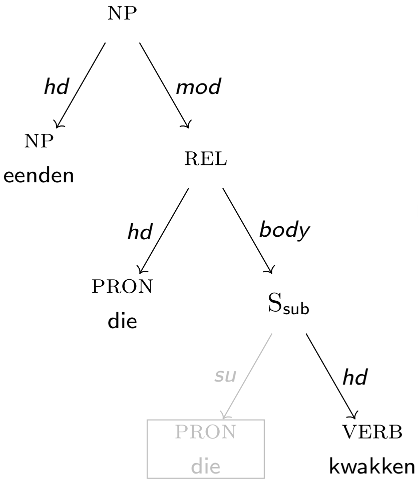
- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed
  type
  if mod: boxed endofunctor of parent
  else:from comp sibs to parent
- assign heads
  incoming head edge
  parent is typed
  no untyped complement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

Harder case: unbounded dependencies



"eenden die kwakken"

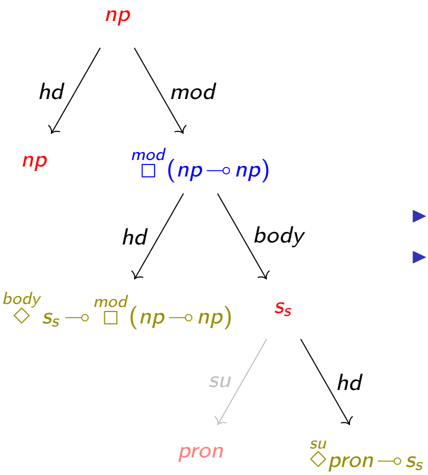
Harder case: unbounded dependencies



"eenden die kwakken"

- detach non-local dependencies

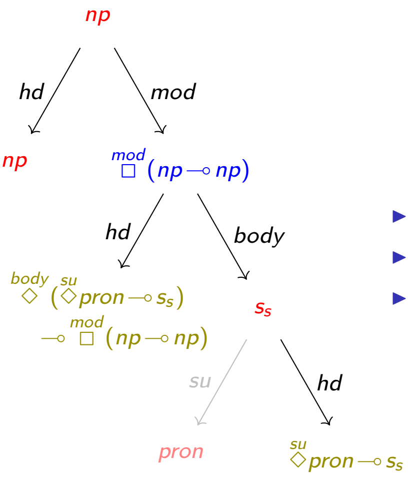
Harder case: unbounded dependencies



"eenden die kwakken"

- detach non-local dependencies
- type trees as before

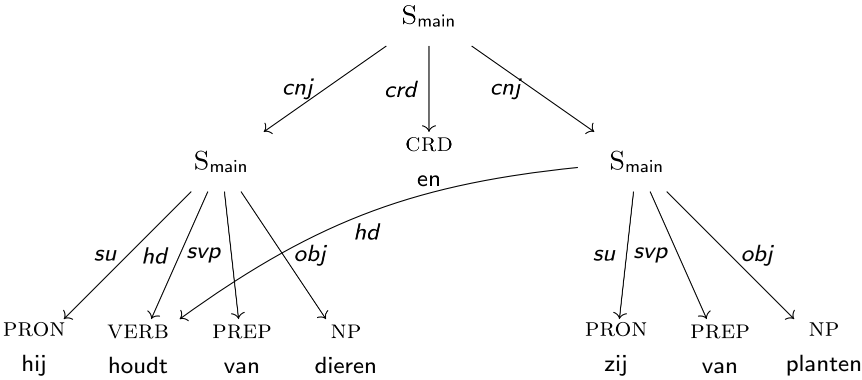
Harder case: unbounded dependencies



"eenden die kwakken"

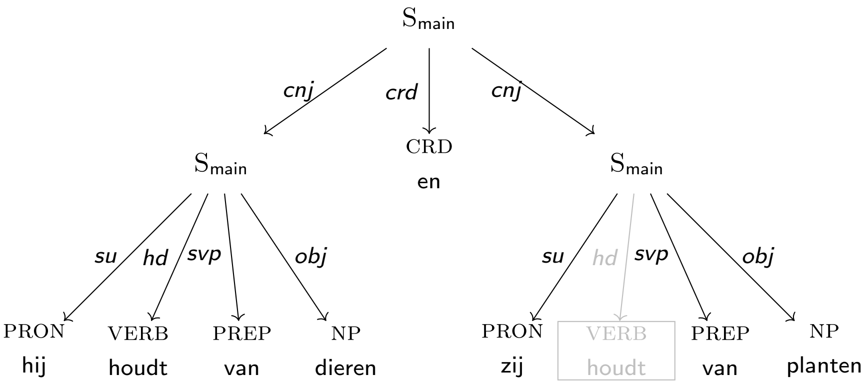
- detach non-local dependencies
- type trees as before
- redact ghosts types from comp sibs

Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

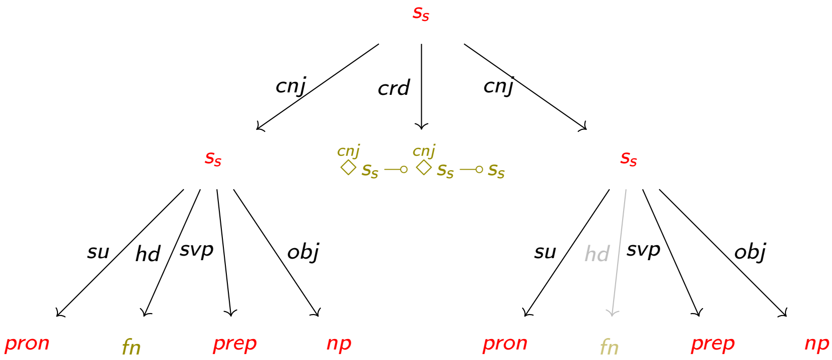
Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

- detach and type trees as usual

Hardest case: Ellipses

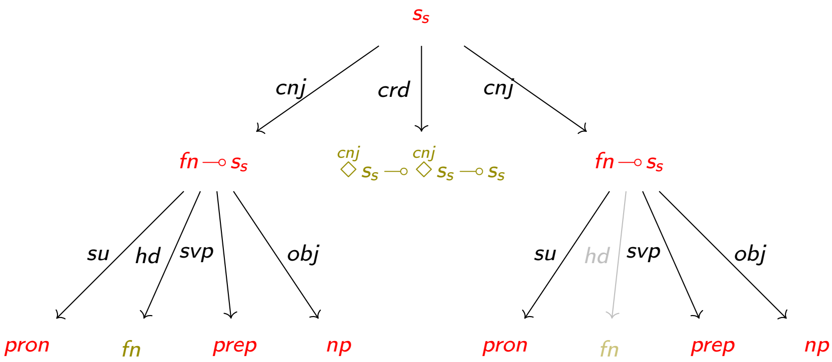


"hij houdt van dieren en zij van planten"

- 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"

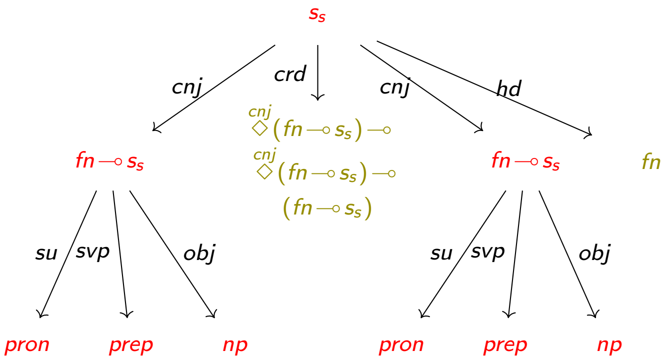
- detach and type trees as usual
- redact missing types from **both** conjuncts

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

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:

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

and

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

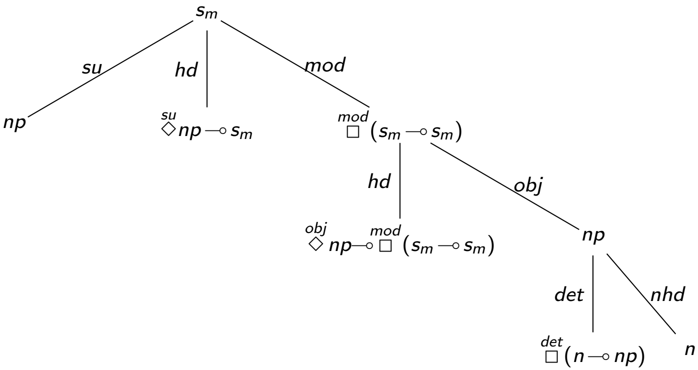
Coordinators as derived types

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,

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

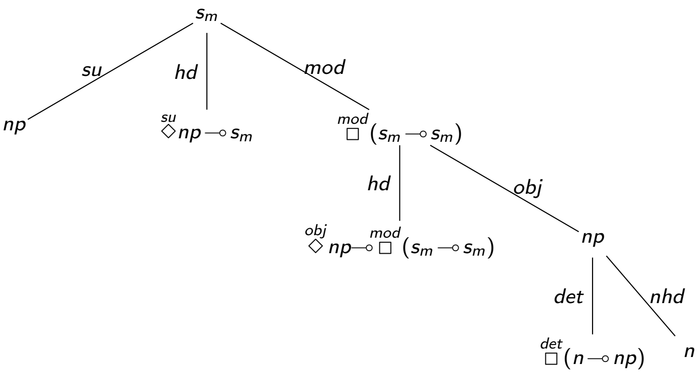
- **Value Raising**
  From $f : \vec{A} \multimap B$ derive $\vec{A} \multimap (B \multimap D) \multimap D$
- **Argument Raising**
  From $f : \vec{A} \multimap B \multimap \vec{C} \multimap D$ derive $\vec{A} \multimap ((B \multimap D) \multimap D) \multimap \vec{C} \multimap D$

From graphs to proofs



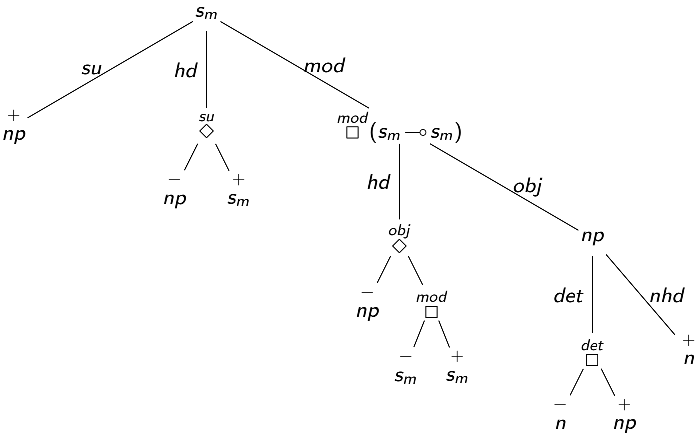
given a typed graph:

From graphs to proofs



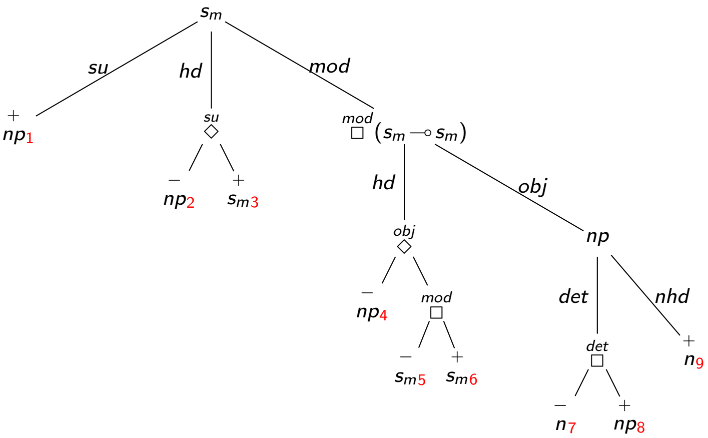
(1) convert types to binary trees and assign polarities

From graphs to proofs



(2) assign identifying indices

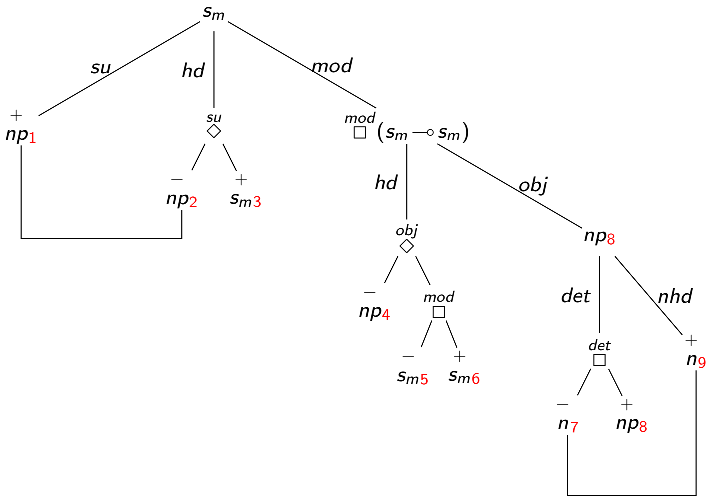
From graphs to proofs



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

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

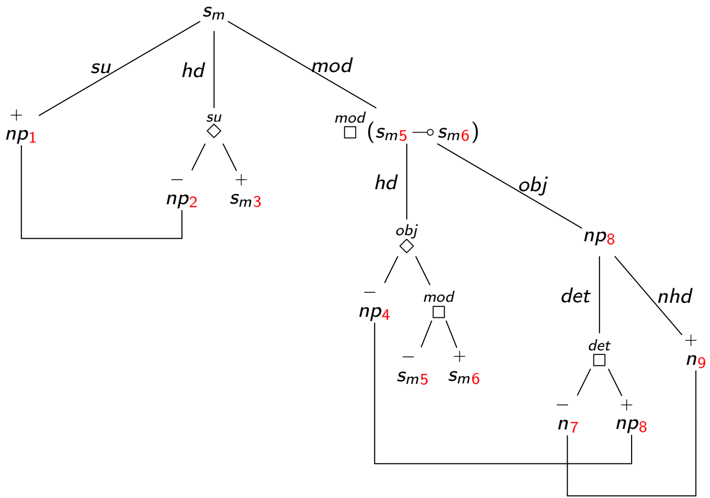
From graphs to proofs



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

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

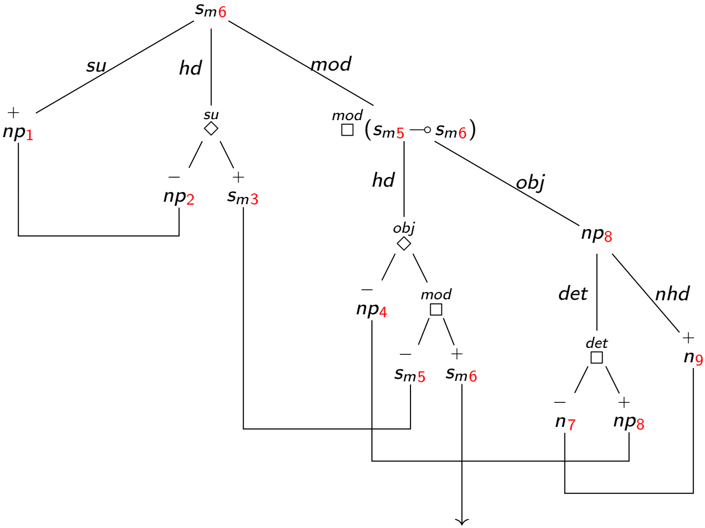
From graphs to proofs



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

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

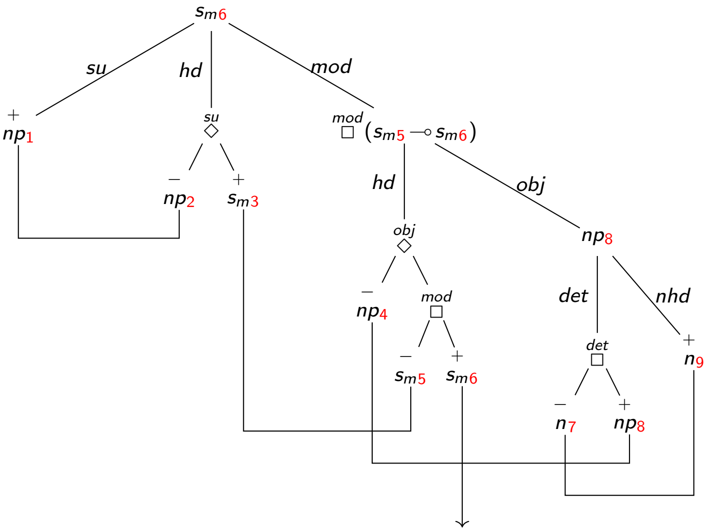
From graphs to proofs



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

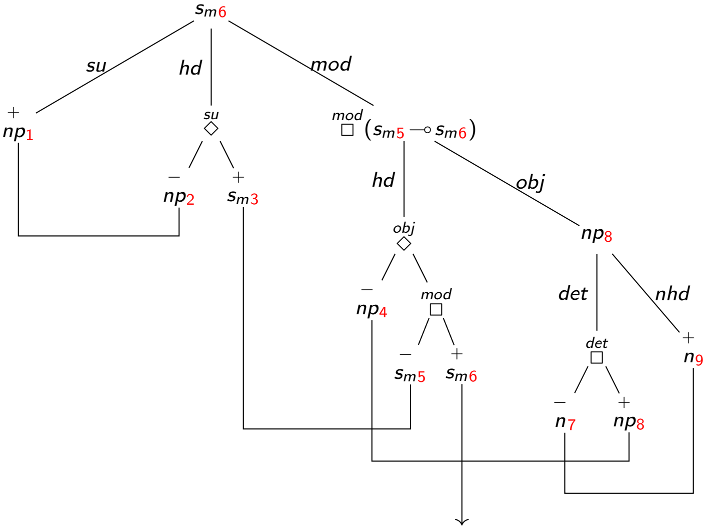
$$\{2 \mapsto 1, 4 \mapsto 8, 5 \mapsto 3, 7 \mapsto 9\}$$

From graphs to proofs



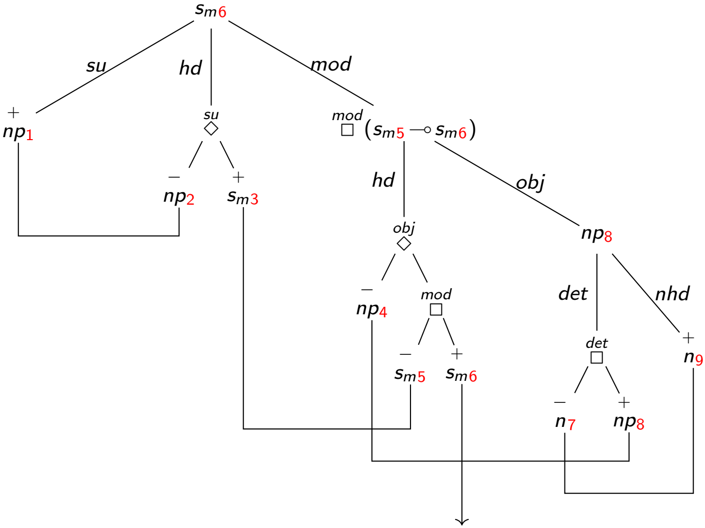
the resulting structure is a *proof net*

From proofs to terms



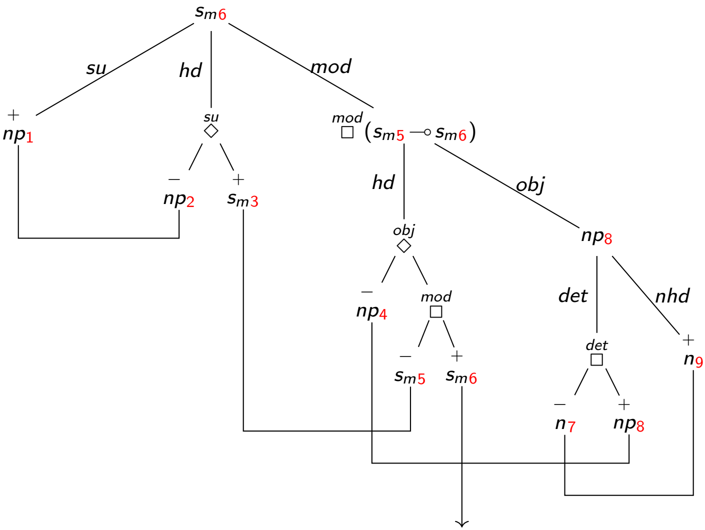
in)

From proofs to terms



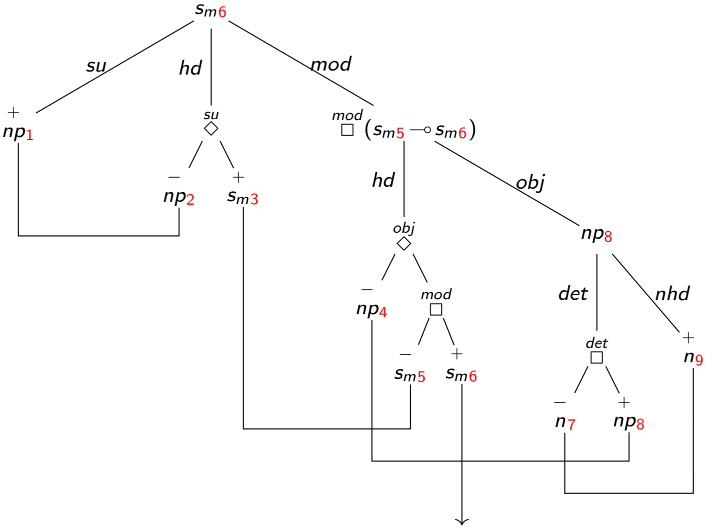
$$in \ \triangle^{obj}\Big(\qquad\qquad \Big)$$

From proofs to terms



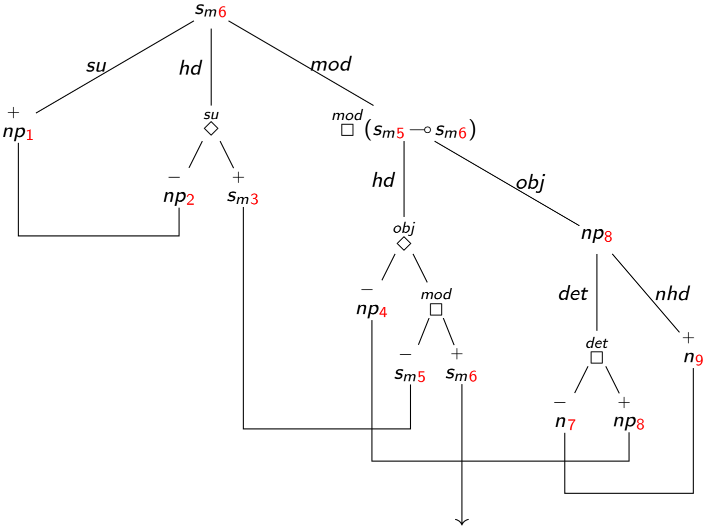
$in\ \triangle^{obj}(\quad het \quad)$

From proofs to terms



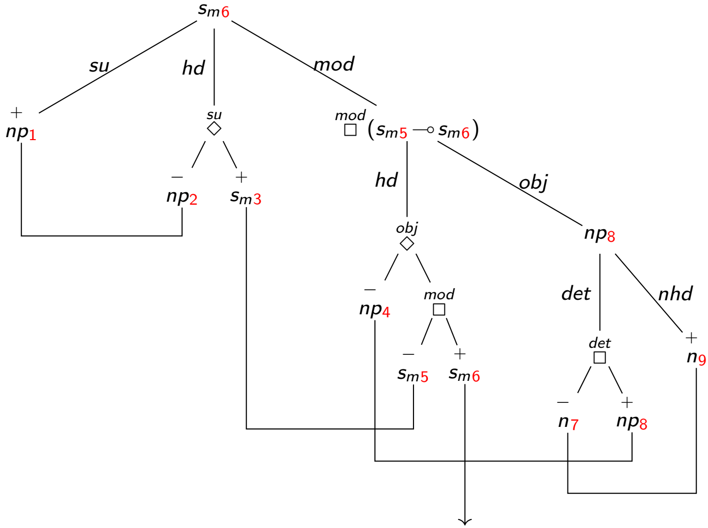
$$in \; \triangle^{obj}\left(\blacktriangledown^{det} het \qquad \right)$$

From proofs to terms



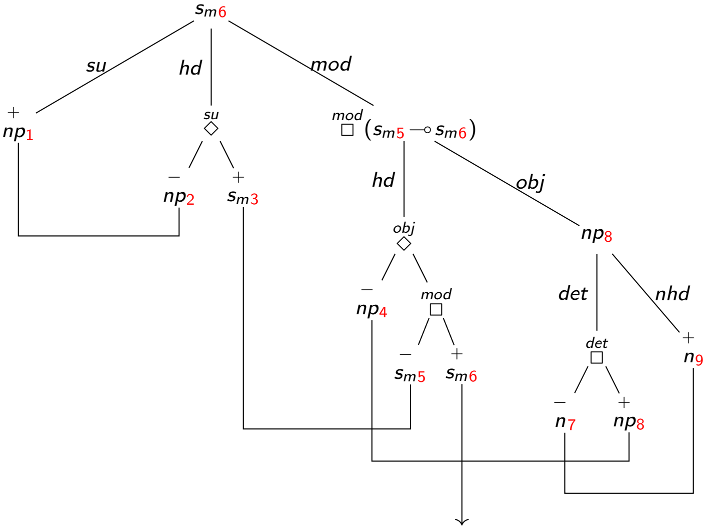
$$in\ \triangle^{obj}\left(\blacktriangledown^{det}\,het\ \ riet\right)$$

From proofs to terms



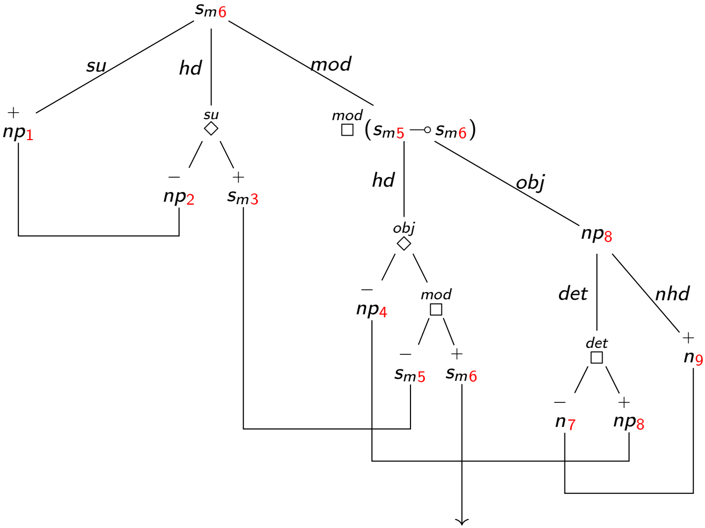
$$\blacktriangledown^{mod}\left(in\ \triangle^{obj}\left(\blacktriangledown^{det} het\ riet\right)\right)$$

From proofs to terms



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

From proofs to terms



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

DIY

- download data
- clone this
  or:
  ```
  python -m pip install
  git+https://github.com/konstantinosKokos/lassy-tlg-extraction@0.3.dev0#egg=LassyExtraction
  ```
- do stuff
- ???
- profit