
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?

Towards Type-Driven & Neural Textual 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

Towards Type-Driven & Neural Textual 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

$$\begin{array}{c}\text{Syntax} \\ (\mathsf{N})\mathsf{L}_{(\diamond,\Box)}\end{array} \xrightarrow{h_{der}} \begin{array}{c}\text{Derivational Semantics} \\ \mathsf{ILL}_{\multimap}\end{array} \xrightarrow{h_{lex}} \begin{array}{c}\text{Lexical Semantics} \\ \mathsf{IL}_{\rightarrow}\end{array}$$

Syntax-Semantics Interface

Type-logical perspective

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

ACG perspective

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

Syntax-Semantics Interface

Now

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

Syntax
???

CHC

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

Lexical Semantics
???

Abstract syntax with NLP

Grammar

$\mathrm{ILL}_{\multimap}$ plus $\diamond, \square$ modalities for *dependency domain demarkation*.

Types inductively defined by:

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

Rules:

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

$$\frac{\Gamma \vdash \mathtt{x} : A}{\langle \Gamma \rangle^d \vdash \mathtt{x}^{\mathtt{d}} : \diamond^d A} \diamond^d I \qquad \frac{\Gamma \vdash \mathtt{x} : \square^d A}{\langle \Gamma \rangle^d \vdash \mathtt{x}_{\mathtt{d}} : A} \square^d E$$

Abstract syntax with NLP

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

Abstract syntax with NLP

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

I, animals, ducks : np

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

Abstract syntax with NLP

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

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

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

$$\texttt{fly (ducks)}^{\texttt{su}}$$

Abstract syntax with NLP

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

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

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

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

Abstract syntax with NLP

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

I, 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	: $\Diamond^{body}\,(\Diamond^{su}\, np \multimap s) \multimap \Box^{mod}\,(np \multimap np)$		

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

$$\big(\texttt{that}\ (\texttt{swim})^{\texttt{body}}\big)_{\texttt{mod}}\ \ \texttt{animals}$$

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

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

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

Why $\diamond$, $\square$?

- Easier to interpret
- Subsume dependency parsing
- More informative for semantics
- Modalities can regulate non-logical parsing

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

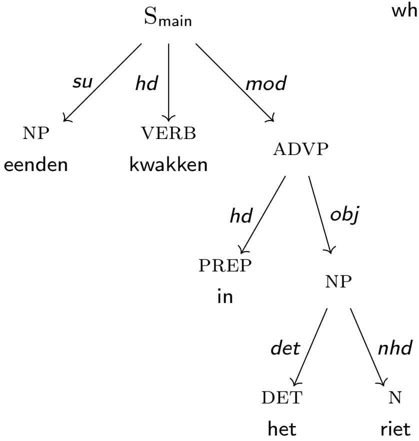
algorithm: graph flooding on dags

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. $\diamond^{\mathit{su}} > \diamond^{\mathit{obj}}$

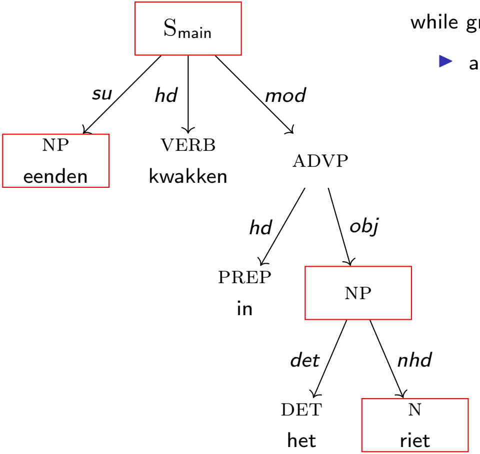
Simple case: trees



“eenden kwakken in het riet”

while graph not fully typed, do:

Simple case: trees

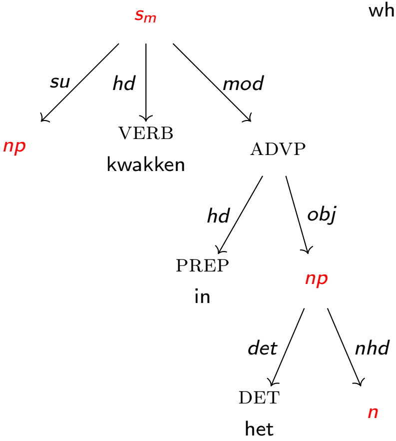


“eenden kwakken in het riet”

while graph not fully typed, do:

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

Simple case: trees

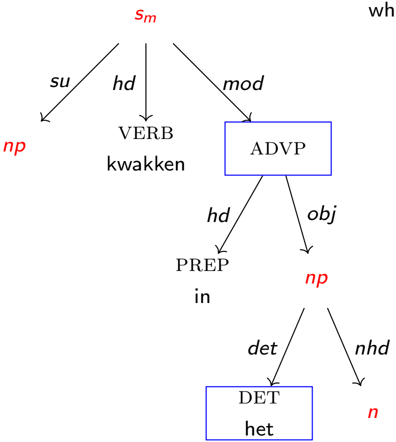


“eenden kwakken in het riet”

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

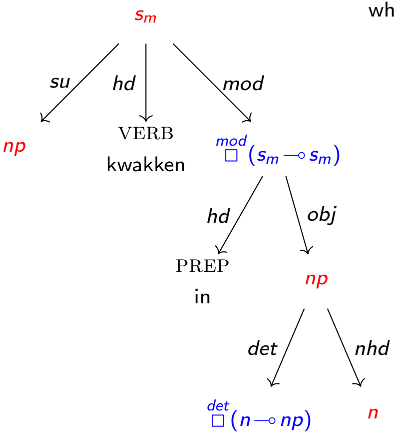


"eenden kwakken in het riet"

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

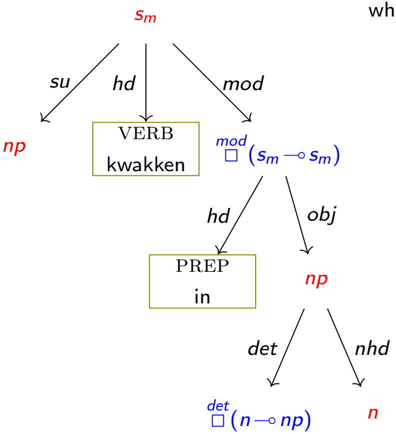


“eenden kwakken in het riet”

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

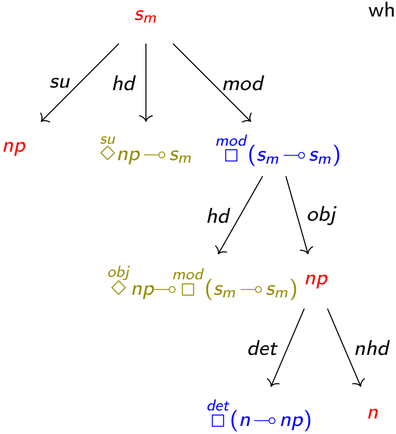


"eenden kwakken in het riet"

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

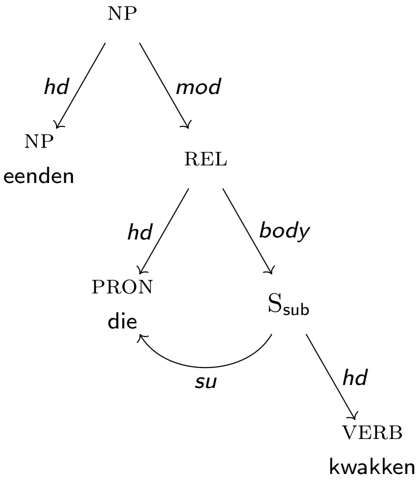


"eenden kwakken in het riet"

while graph not fully typed, do:

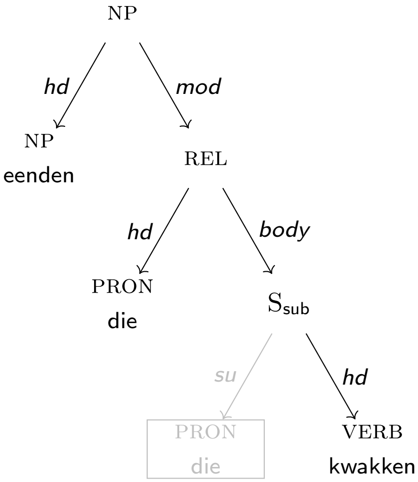
- assign stand-alone nodes
  no incoming adjunct or head edge
  type via the $\mathcal{A}$-map
- assign adjuncts
  incoming adjunct edge
  parent is typed
  type
  if mod: boxed endofunctor of parent
  else:from comp sibs to parent
- assign heads
  incoming head edge
  parent is typed
  no untyped complement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

Harder case: unbounded dependencies



"eenden die kwakken"

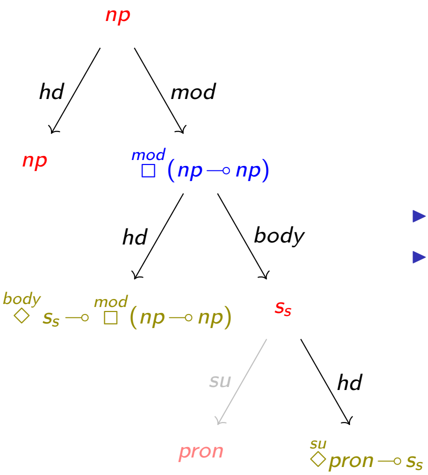
Harder case: unbounded dependencies



"eenden die kwakken"

- detach non-local dependencies

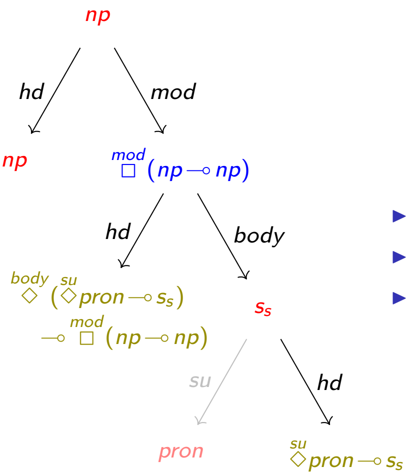
Harder case: unbounded dependencies



- detach non-local dependencies
- type trees as before

"eenden die kwakken"

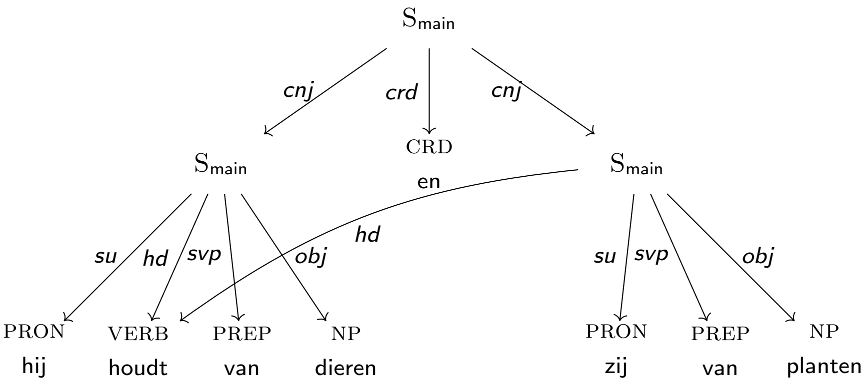
Harder case: unbounded dependencies



"eenden die kwakken"

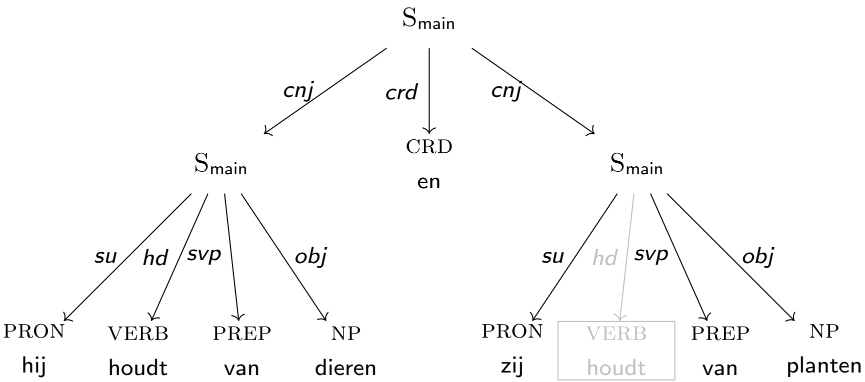
- detach non-local dependencies
- type trees as before
- redact ghosts types from comp sibs

Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

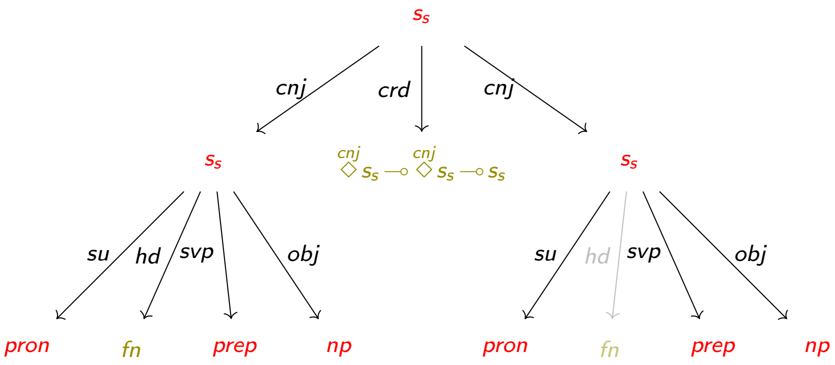
Hardest case: Ellipses



"hij houdt van dieren en zij van planten"

- detach and type trees as usual

Hardest case: Ellipses

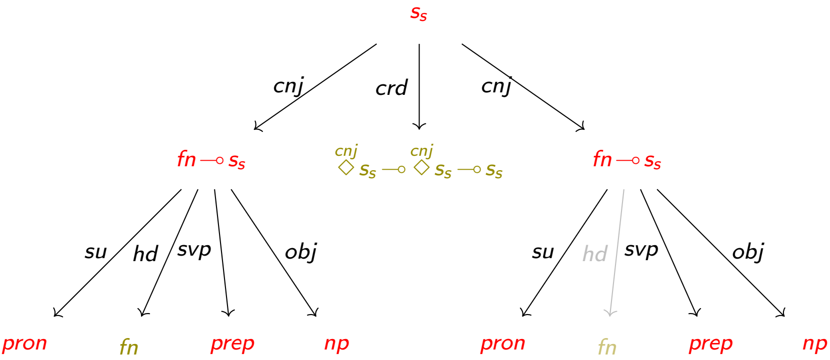


"hij houdt van dieren en zij van planten"

- 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"

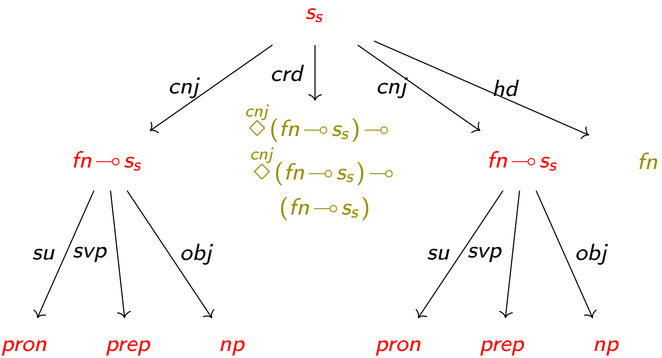
- detach and type trees as usual
- redact missing types from **both** conjuncts

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

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)

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

A glimpse at a higher universe

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

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

In that universe, modifiers and coordinators are polymorphic types:

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

and

$$\mathsf{w} := \Lambda\alpha.\lambda\mathtt{xy}.((\mathtt{w}\ (\mathtt{x})^{\mathtt{cnj}})\ (\mathtt{y})^{\mathtt{cnj}}) : \forall\alpha.\Diamond^{cnj}\alpha \multimap \Diamond^{cnj}\alpha \multimap \alpha$$

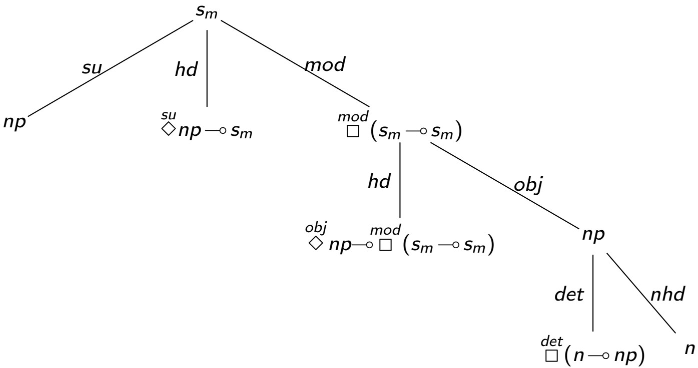
Coordinators as derived types

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,

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

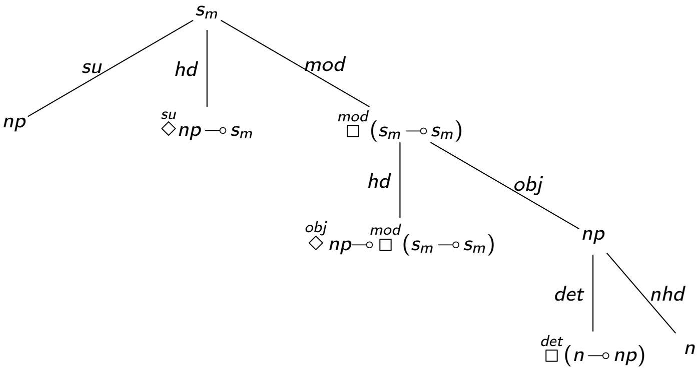
- **Value Raising**
  From $f : \vec{A} \multimap B$ derive $\vec{A} \multimap (B \multimap D) \multimap D$
- **Argument Raising**
  From $f : \vec{A} \multimap B \multimap \vec{C} \multimap D$ derive $\vec{A} \multimap ((B \multimap D) \multimap D) \multimap \vec{C} \multimap D$

From graphs to proofs



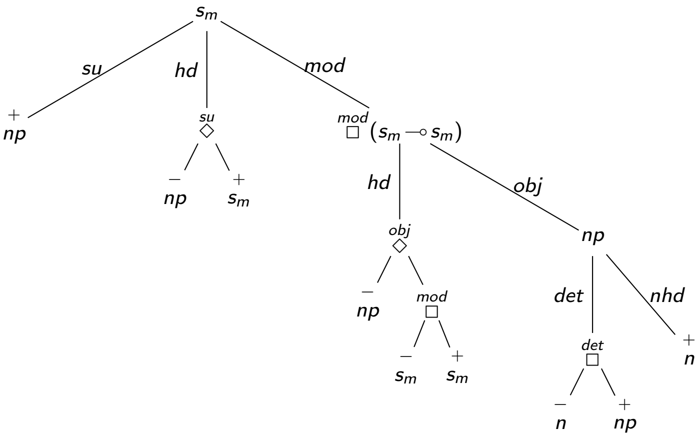
given a typed graph:

From graphs to proofs



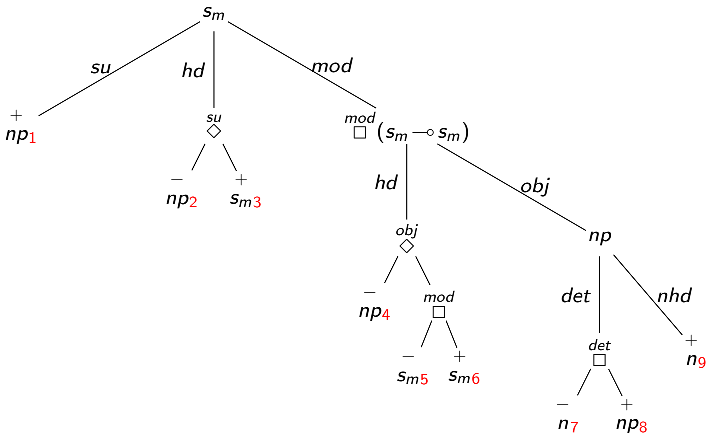
(1) convert types to binary trees and assign polarities

From graphs to proofs



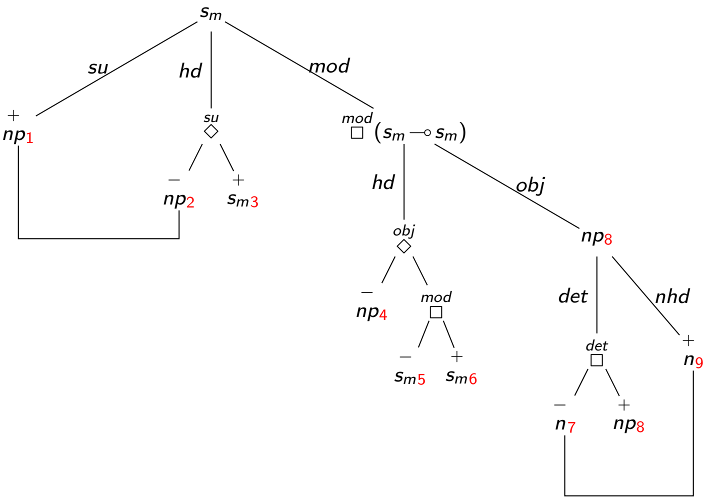
(2) assign identifying indices

From graphs to proofs



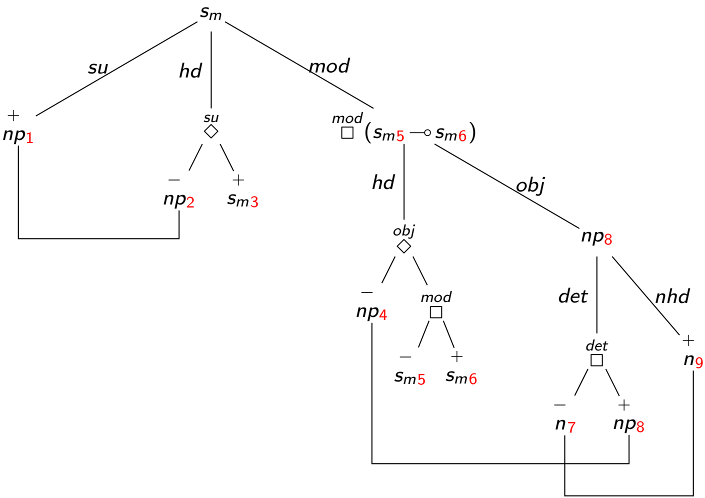
(3) traverse upwards, identifying pos/neg atoms and propagating indices

From graphs to proofs



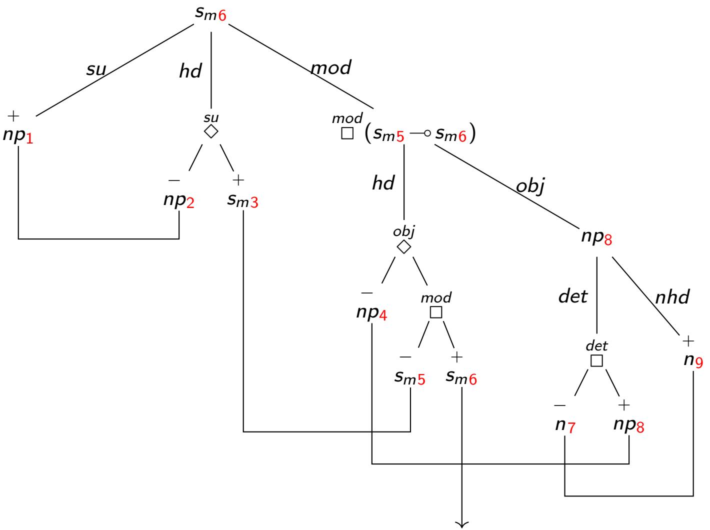
(3) traverse upwards, identifying pos/neg atoms and propagating indices

From graphs to proofs



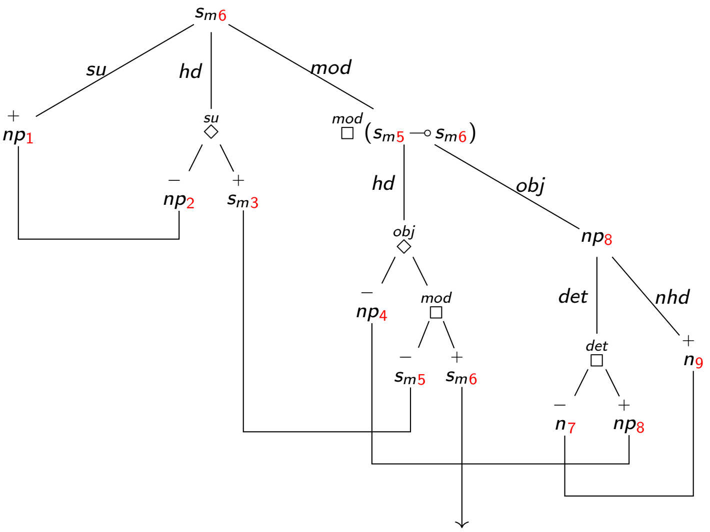
(3) traverse upwards, identifying pos/neg atoms and propagating indices

From graphs to proofs



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

From graphs to proofs



the resulting structure is a *proof net*