Spinning Raw Text into Lambda Terms with Graph Attention

Konstantinos Kogkalidis[◇], Michael Moortgat[◇], Richard Moot[□]
◇ Institute for Language Sciences, Utrecht University | □ LIRMM, Université de Montpellier, CNRS

Introduction

Categorial Grammars & the **Holy Trinity** :

Logic	**Computer Science**	**Linguistics**
Propositional Constant	Base Type	Syntactic Category
Inference Rule	Term Rewrite	Phrase Formation
Axiom	Variable	Word
Provability	Type Inhabitation	Grammaticality
Deduction	Program Synthesis	Parsing

Standard pipeline:
parse surface form $\Longrightarrow$ convert to λ $\Longrightarrow$ downstream task

This work:
parse deep λ form $\Longrightarrow$ downstream task

Type Grammar

A **semantics-first**, **type-driven** grammar for the 21st century.
Combines:
i. a linear functional core
ii. a set of residuated modalities

to capture:
i. grammatical function-argument structures
ii. dependency-domain annotations ($\longleftarrow$ new fancy feature!)

✓ cooler than CCG!
✓ type checks!

Grammatical Types are inductively defined as:

$A, B := p$ # base categories, e.g. NP
$\mid \diamondsuit_d A$ # d-marked *complements*, e.g. $\diamondsuit_{su}NP$
$\mid A \multimap B$ # grammatical functions, e.g. $\diamondsuit_{su}NP \multimap S$
$\mid \Box_d A$ # d-marked *adjuncts*, e.g. $\Box_d(NP \multimap NP)$

Inference Rules assert grammaticality and provide recipes for compositional meaning assembly in the form of λ expressions:

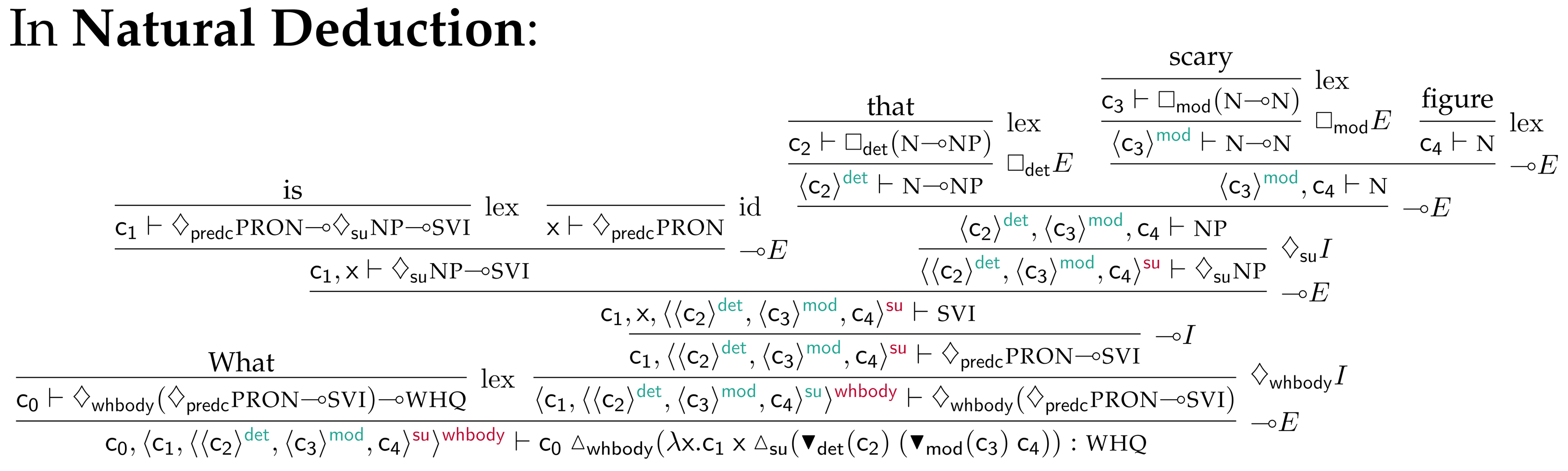
$$\frac{}{x : A \vdash x : A}\ \text{id} \qquad \frac{(c \mapsto A) \in \mathcal{L}}{c : A \vdash c : A}\ \text{lex}$$

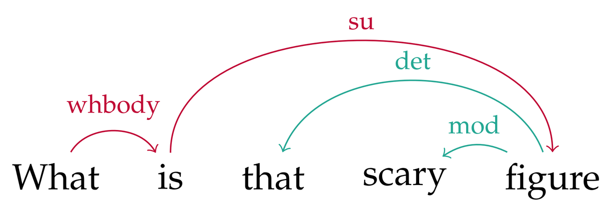
$$\frac{\Gamma \vdash s : A \multimap B \quad \Delta \vdash t : A}{\Gamma, \Delta \vdash s\ t : B}\ \multimap E \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \multimap B}\ \multimap I$$

$$\frac{\Gamma \vdash s : \Box_\delta A}{\langle\Gamma\rangle^\delta \vdash \blacktriangledown_\delta s : A}\ \Box_\delta E \qquad \frac{\Gamma \vdash s : A}{\langle\Gamma\rangle^\delta \vdash \vartriangle_\delta s : \diamondsuit_\delta A}\ \diamondsuit_\delta I$$

Proof Representations

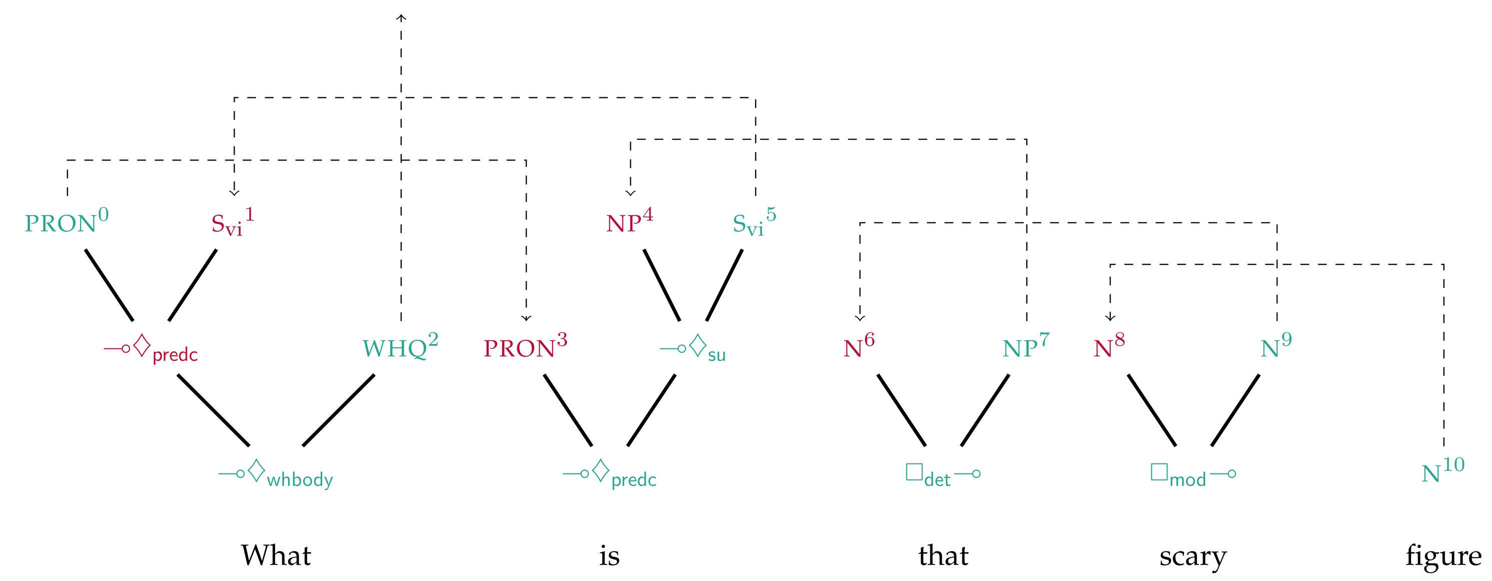
In **Natural Deduction**:





a **proof-tree**
✓ clear computational interpretation
✓ trivial interface to a dependency tree
✓ correct by construction
✗ takes time to build (induction in *depth*)
✗ odd structure for ML (can't batch!)

As a **Proof Net**:



a **proof-graph**
✓ each type decomposed to a bicolor tree
✓ functional relations specified as a bijection
✓ fully parallel, easy to vectorize
✗ computationally intractable ($n!$ combinations)
✗ requires formal verification

$\Pi_N :=$

	6	8
9	✓	
10		✓

System Architecture

spind$^2\lambda$e

spind$^2\lambda$e parses into dependency-decorated λ expressions

A neat packaging of:

1. **a graph-based supertagger**
   i. learns (word $\mapsto$ type) mapping as a graph generation task
   ii. can correctly produce new types on demand
   iii. SOTA across datasets (multi-framework, multi-lingual)
   iv. length-parallel, depth-linear decoding (i.e. constant)
2. **an OT-based proof search module**
   i. learns proof search as a node matching task
   ii. fully parallel in batch/depth/length
   iii. no iteration (unlike shift-reduce)
3. **a mini type checker**
   i. Python DSL to write/manipulate well-typed parses
   ii. handles proof net $\longleftrightarrow$ nat. ded. $\longleftrightarrow$ λ-term conversions
   iii. asserts validity of neural output
   iv. user-friendly hooks

Experiments

Current implementation trained/evaluated on Æthel (~70,000 proof-derivations of written Dutch):

parsability (some proof obtainable)	coverage (some proof obtained)
86.83	84.94
types correct (correct proof obtainable)	**accuracy** (correct proof obtained)
56.88	55.30

- faster & stricter than conventional parsers, just as accurate
- proof search bottlenecked by supertagging

Try It Out/Read More

source code,
installation instructions
& usage examples

arXiv preprint