HAL
open science

Neural Proof Nets

Konstantinos Kogkalidis, Michael Moortgat, Richard Moot

To cite this version:

Konstantinos Kogkalidis, Michael Moortgat, Richard Moot. Neural Proof Nets. CoNLL 2020 - 24th Conference on Computational Natural Language Learning, Nov 2020, Virtual, Dominican Republic. pp.26-40, 10.18653/v1/2020.conll-1.3 . lirmm-02952267

HAL Id: lirmm-02952267

https://hal-lirmm.ccsd.cnrs.fr/lirmm-02952267

Submitted on 29 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

Neural Proof Nets

Konstantinos Kogkalidis and **Michael Moortgat** and **Richard Moot**
Utrecht Institute of Linguistics OTS, Utrecht University
LIRMM, Université de Montpellier, CNRS
{k.kogkalidis,m.j.moortgat}@uu.nl, richard.moot@lirmm.fr

arXiv:2009.12702v1 [cs.CL] 26 Sep 2020

Abstract

Linear logic and the linear λ-calculus have a long standing tradition in the study of natural language form and meaning. Among the proof calculi of linear logic, proof nets are of particular interest, offering an attractive geometric representation of derivations that is unburdened by the bureaucratic complications of conventional prooftheoretic formats. Building on recent advances in set-theoretic learning, we propose a neural variant of proof nets based on Sinkhorn networks, which allows us to translate parsing as the problem of extracting syntactic primitives and permuting them into alignment. Our methodology induces a batch-efficient, end-to-end differentiable architecture that actualizes a formally grounded yet highly efficient neuro-symbolic parser. We test our approach on Æthel, a dataset of type-logical derivations for written Dutch, where it manages to correctly transcribe raw text sentences into proofs and terms of the linear λ-calculus with an accuracy of as high as 70%.

1 Introduction

There is a broad consensus among grammar formalisms that the composition of form and meaning in natural language is a resource-sensitive process, with the words making up a phrase contributing exactly once to the resulting whole. The sentence "the Mad Hatter offered" is ill-formed because of a *lack* of grammatical material, "offer" being a ditransitive verb; "the Cheshire Cat grinned Alice a cup of tea" on the other hand is ill-formed because of an *excess* of material, which the intransitive verb "grin" cannot accommodate.

Given the resource-sensitive nature of language, it comes as no surprise that Linear Logic (Girard, 1987), in particular its intuitionistic version ILL, plays a central role in current logic-based grammar formalisms. Abstract Categorial Grammars and Lambda Grammars (de Groote, 2001; Muskens, 2001) use ILL "as-is" to characterize an abstract level of grammatical structure from which surface form and semantic interpretation are obtained by means of compositional translations. Modern typelogical grammars in the tradition of the Lambek Calculus (Lambek, 1958), e.g. Multimodal TLG (Moortgat, 1996), Displacement Calculus (Morrill, 2014), Hybrid TLG (Kubota and Levine, 2020), refine the type language to account for syntactic aspects of word order and constituency; ILL here is the target logic for semantic interpretation, reached by a homomorphism relating types and derivations of the syntactic calculus to their semantic counterparts.

A common feature of the aforementioned formalisms is their adoption of the *parsing-as-deduction* method: determining whether a phrase is syntactically well-formed is seen as the outcome of a process of logical deduction. This logical deduction automatically gives rise to a program for meaning composition, thanks to the remarkable correspondence between logical proof and computation known as the Curry-Howard isomorphism (Sørensen and Urzyczyn, 2006), a natural manifestation of the syntax-semantics interface. The Curry-Howard λ-terms associated with derivations are neutral with respect to the particular semantic theory one wants to adopt, accommodating both the truth-conditional view of formal semantics and the vector-based distributional view (Muskens and Sadrzadeh, 2018), among others.

Despite their formal appeal, grammars based on variants of linear logic have fallen out of favour within the NLP community, owing to a scarcity of large-scale datasets, but also due to difficulties in aligning them with the established high-performance neural toolkit. Seeking to bridge the gap between formal theory and applied practice, we focus on the *proof nets* of linear logic, a lean graphical calculus that does away with the bureau-

cratic symbol-manipulation overhead characteristic of conventional prooftheoretic presentations (§2). Integrating proof nets with recent advances in neural processing, we propose a novel approach to linear logic proof search that eliminates issues commonly associated with higher-order types and hypothetical reasoning, while greatly reducing the computational costs of structure manipulation, backtracking and iterative processing that burden standard parsing techniques (§3).

Our proposed methodology relies on two key components. The first is an encoder/decoder-based supertagger that converts raw text sentences into linear logic judgements by dynamically constructing contextual type assignments, one primitive symbol at a time. The second is a bi-modal encoder that contextualizes the generated judgement in conjunction with the input sentence. The contextualized representations are fed into a Sinkhorn layer, tasked with finding the valid permutation that brings primitive symbol occurrences into alignment. The architecture induced is trained on labeled data, and assumes the role of a formally grounded yet highly accurate parser, which transforms raw text sentences into linear logic proofs and computational terms of the simply typed linear λ-calculus, further decorated with dependency annotations that allow reconstruction of the underlying dependency graph (§4).

2 Background

We briefly summarize the logical background we are assuming, starting with ILL$_{\multimap}$, the implication-only fragment of ILL, then moving on to the dependency-enhanced version ILL$_{\multimap,\Diamond,\Box}$ which we employ in our experimental setup.

2.1 ILL$_{\multimap}$

Formulas (or *types*) of ILL$_{\multimap}$ are inductively defined according to the grammar below:

$$\mathcal{T} ::= \mathrm{A} \mid \mathrm{T}_1 \multimap \mathrm{T}_2$$

Formula A is taken from a finite set of atomic formulas $\mathcal{A} \subset \mathcal{T}$; a complex formula $\mathrm{T}_1 \multimap \mathrm{T}_2$ is the type signature of a transformation that applies on $\mathrm{T}_1 \in \mathcal{T}$ and produces $\mathrm{T}_2 \in \mathcal{T}$, consuming the argument in the process. This view of formulas as non-renewable resources makes ILL$_{\multimap}$ the logic of *linear* functions.[1]

[1] We refer to Wadler (1993) for a gentle introduction.

We can present the inference rules of ILL$_{\multimap}$ together with the associated linear λ-terms in Natural Deduction format. Judgements are sequents of the form $\mathtt{x}_1 : \mathrm{T}_1, \ldots, \mathtt{x}_n : \mathrm{T}_n \vdash \mathtt{M} : \mathrm{C}$. The antecedent left of the turnstile is a *typing environment* (or *context*), a sequence of variables $\mathtt{x}_i$, each given a type declaration T_i. These variables serve as the *parameters* of a program $\mathtt{M}$ of type C that corresponds to the proof of the sequent.

Proofs are built from axioms $\mathtt{x} : \mathrm{T} \vdash \mathtt{x} : \mathrm{T}$ with the aid of two rules of inference:

$$\frac{\Gamma \vdash \mathtt{M} : \mathrm{T}_1 \multimap \mathrm{T}_2 \quad \Delta \vdash \mathtt{N} : \mathrm{T}_1}{\Gamma, \Delta \vdash (\mathtt{M}\,\mathtt{N}) : \mathrm{T}_2} \multimap E \tag{1}$$

$$\frac{\Gamma, \mathtt{x} : \mathrm{T}_1 \vdash \mathtt{M} : \mathrm{T}_2}{\Gamma \vdash \lambda\mathtt{x}.\mathtt{M} : \mathrm{T}_1 \multimap \mathrm{T}_2} \multimap I \tag{2}$$

(1) is the elimination of the implication and models *function application*; it proposes that if from some context Γ one can derive a program $\mathtt{M}$ of type $\mathrm{T}_1 \multimap \mathrm{T}_2$, and from context Δ one can derive a program $\mathtt{N}$ of type T_1, then from the multiset union Γ, Δ one can derive a term $(\mathtt{M}\,\mathtt{N})$ of type T_2.

(2) is the introduction of the implication and models *function abstraction*; it proposes that if from a context Γ together with a type declaration $\mathtt{x} : \mathrm{T}_1$ one can derive a program term $\mathtt{M}$ of type T_2, then from Γ alone one can derive the abstraction $\lambda\mathtt{x}.\mathtt{M}$, denoting a linear function of type $\mathrm{T}_1 \multimap \mathrm{T}_2$.

To obtain a *grammar* based on ILL$_{\multimap}$, we consider the logic in combination with a *lexicon*, assigning one or more type formulas to the words of the language. In this setting, the proof of a sequent $\mathtt{x}_1 : \mathrm{T}_1, \ldots, \mathtt{x}_n : \mathrm{T}_n \vdash \mathtt{M} : \mathrm{C}$ constitutes an algorithm to compute a meaning $\mathtt{M}$ of type C, given by substituting parameters $\mathtt{x}_i$ with lexical meanings $\mathtt{w}_i$. In the type lexicon, atomic types are used to denote syntactically autonomous, stand-alone units (words and phrases); e.g. NP for noun-phrase, S for sentence, etc. Function types are assigned to incomplete expressions, e.g. $\mathrm{NP} \multimap \mathrm{S}$ for an intransitive verb consuming a noun-phrase to produce a sentence, $\mathrm{NP} \multimap \mathrm{NP} \multimap \mathrm{S}$ for a transitive verb, etc.[2] Higher-order types, i.e. types of order greater than 1, denote functions that apply to functions; these give the grammar access to hypothetical reasoning, in virtue of the implication introduction rule.[3] Combined with parametric polymorphism,

[2] Read $\multimap$ as right-associative.

[3] $\mathcal{O}(\mathrm{A})$, the order of an atomic type, equals zero; for function types $\mathcal{O}(\mathrm{T}_1 \multimap \mathrm{T}_2) = \max(\mathcal{O}(\mathrm{T}_1) + 1, \mathcal{O}(\mathrm{T}_2))$.

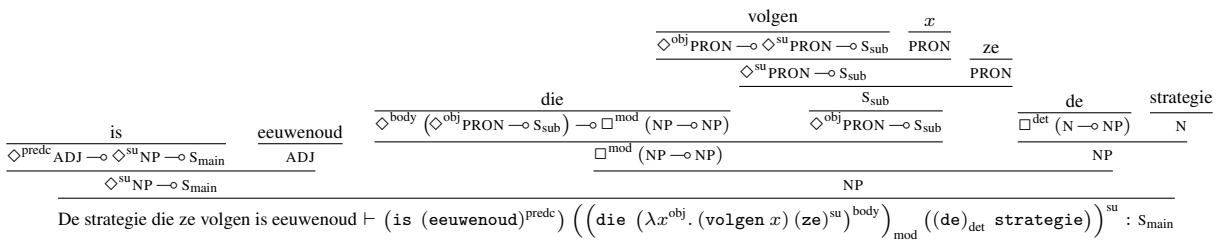

Figure 1: Example derivation and Curry-Howard λ-term for the phrase *De strategie die ze volgen is eeuwenoud* ("The strategy that they follow is ancient") from Æthel sample `dpc-ind-001645-nl-sen.p.12.s.1_1`, showcasing how hypothetical reasoning enables the derivation of an object-relative clause (note how the instantiation of variable x of type PRON followed by its subsequent abstraction creates an argument for the higher-order function assigned to "die"). Judgement premises and rule names have been omitted for brevity's sake.

higher-order types eschew the need for phantom syntactic nodes, enabling straightforward derivations for apparent non-linear phenomena involving long-range dependencies, elliptical conjunctions, wh-movement and the like.

2.2 ILL$_{\multimap,\diamond,\square}$

For our experimental setup, we will be utilizing the Æthel dataset, a Dutch corpus of type-logical derivations (Kogkalidis et al., 2020). Non-commutative categorial grammars in the tradition of Lambek (1958) attempt to directly capture syntactic fine-structure by making a distinction between left- and right-directed variants of the implication. In order to deal with the relatively free word order of Dutch and contrary to the former, Æthel's type system sticks to the directionally non-committed $\multimap$ for function types, but compensates with two strategies for introducing syntactic discrimination. First, the *atomic* type inventory distinguishes between major clausal types $S_{sub}, S_{v1}, S_{main}$, based on the positioning of their verbal head (clause final, clause initial, verb second, respectively). Secondly, *function* types are enhanced with dependency information, expressed via a family of unary modalities $\diamond^d$, $\square^m$, with dependency labels d, m drawn from disjoint sets of complement vs adjunct markers. The new constructors produce types $\diamond^d A \multimap B$, used to denote the *head* of a phrase B that selects for a *complement* A and assigns it the dependency role d, and types $\square^m(A \multimap B)$, used to denote *adjuncts*, i.e. non-head functions that project the dependency role m upon application. Following dependency grammar tradition, determiners and modifiers are treated as non-head functions.

The type enhancement induces a dependency marking on the derived λ-term, reflecting the introduction/elimination of the $\diamond$, $\square$ constructors; each dependency domain has a unique head, together with its complements and possible adjuncts, denoted by superscripts and subscripts, respectively. Figure 1 provides an example derivation and the corresponding λ-term.

A shallow dependency graph can be trivially reconstructed by traversal of the decorated λ-term, recursively establishing labeled edges along the path from a phrasal head to the head of each of its dependants while skipping abstractions; see Figure 4 for an example.

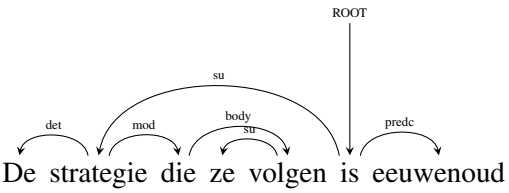

Figure 4: Shallow graph for the term of Figure 1.

2.3 Proof Nets

Despite their clear computational interpretation (Girard et al., 1988; Troelstra and Schwichtenberg, 2000; Sørensen and Urzyczyn, 2006), proofs in natural deduction format are arduous to obtain; reasoning with hypotheticals necessitates a mixture of forward and backward chaining search strategies. The sequent calculus presentation, on the other hand, permits exhaustive proof search via pure backward chaining, but does so at the cost of spurious ambiguity. Moreover, both the above assume a tree-like proof structure, which hinders their parallel processing and impairs compatibility with neural methods. As an alternative, we turn

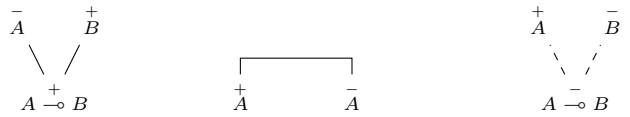

Figure 2: Links for linear logic proof nets. Left/right: positive/negative implication. Center: axiom link.

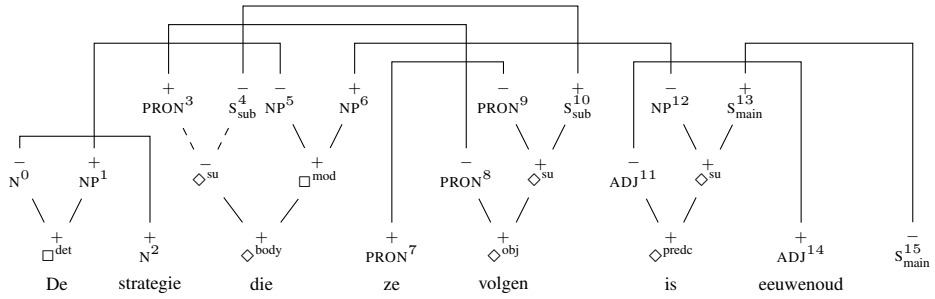

Figure 3: Proof net corresponding to the natural deduction derivation of Figure 1, with modal markings in place of implication arrows. Atomic types at the fringe of the formula decomposition trees are marked with superscript indices denoting their position for ease of identification. During decoding, the proof frame is flattened as the linear sequence: $\big[$[SOS], $\square^{\text{det}}$, N, NP, [SEP], N, [SEP], $\diamond^{\text{body}}$, $\diamond^{\text{su}}$, PRON, S_{sub}, $\square^{\text{mod}}$, NP, NP, [SEP], PRON, [SEP], $\diamond^{\text{obj}}$, ...$\big]$

our attention towards *proof nets* (Girard, 1987), a graphical representation of linear logic proofs that captures hypothetical reasoning in a purely geometric manner. Proof nets may be seen as a parallelized version of the sequent calculus or a multi-conclusion version of natural deduction and combine the best of both words, allowing for flexible and easily parallelized proof search while maintaining the 1-to-1 correspondence with the terms of the linear λ-calculus.

To define ILL proof nets, we first need the auxiliary notion of *polarity*. We assign *positive* polarity to resources we have, *negative* polarity to resources we seek. Logically, a formula with negative polarity appears in *conclusion* position (right of the turnstile), whereas formulas with positive polarity appear in *premise* position (left of the turnstile). Given a formula and its polarity, the polarity of its subformulas is computed as follows: for a positive formula $\text{T}_1 \multimap \text{T}_2$, T_1 is negative and T_2 is positive, whereas for a negative formula $\text{T}_1 \multimap \text{T}_2$, T_1 is positive and T_2 is negative.

With respect to proof search, proof nets present a simple but general setup as follows. (1) Begin by writing down the formula decomposition tree for all formulas in a sequent $\text{P}_1, \ldots \text{P}_n \vdash \text{C}$, keeping track of polarity information; the result is called a *proof frame*. (2) Find a perfect matching between the positive and negative atomic formulas; the result is called a *proof structure*. (3) Finally, verify that the proof structure satisfies the correctness condition; if so, the result is a *proof net*.

Formula decomposition is fully deterministic, with the decomposition rules shown in Figure 2. There are two logical links, denoting positive and negative occurrences of an implication (corresponding to the elimination and introduction rules of natural deduction, respectively). A third rule, called the axiom link, connects two equal formulas of opposite polarity.

To transform a proof frame into a proof structure, we first need to check the *count invariance* property, which requires an equal count of positive and negative occurrences for every atomic type, and then connect atoms of opposite polarity. In principle, we can connect any positive atom to any negative atom when both are of the same type; the combinatorics of proof search lies, therefore, in the axiom connections (the number of possible proof structures scales factorial to the number of atoms). Not all proof structures are, however, proof nets. Validating the correctness of a proof net can be done in linear time (Guerrini, 1999; Murawski and Ong, 2000); a common approach is to attempt a traversal of the proof net, ensuring that all nodes are visited (connectedness) and no loops exist (acyclicity) (Danos and Regnier, 1989). There is an apparent tension here between finding just *a* matching of atomic formulas (which is trivial once we satisfy the count invariance) and finding *the* correct matching, which produces not only a proof net, but also the preferred semantic reading of the sentence.

Deciding the provability of a linear logic sequent is an NP-complete problem (Lincoln, 1995), even in the simplest case where formulas are restricted to order 1 (Kanovich, 1994). Figure 3 shows the proof net equivalent to the derivation of Figure 1.

3 Neural Proof Nets

To sidestep the complexity inherent in the combinatorics of linear logic proof search, we investigate proof net construction from a neural perspective. First, we will need to convert a sentence into a proof frame, i.e. the decomposition of a logical judgement of the form $\mathrm{P}_1, \ldots \mathrm{P}_n \vdash \mathrm{C}$, with P_i the type of word i and C the goal type to be derived. Having obtained a correct proof frame, the problem boils down to establishing axiom links between the set of positive and negative atoms and verifying their validity according to the correctness criteria. We address each of these steps via a functionally independent neural module, and define *Neural Proof Nets* as their composition.

3.1 Proof Frames

Obtaining proof frames is a special case of supertagging, a common problem in NLP literature (Bangalore and Joshi, 1999). Conventional practice treats supertagging as a discriminative sequence labeling problem, with a neural model contextualizing the tokens of an input sentence before passing them through a linear projection in order to convert them to class weights (Xu et al., 2015; Vaswani et al., 2016). Here, instead, we adopt the generative paradigm (Kogkalidis et al., 2019; Bhargava and Penn, 2020), whereby each type is itself perceived as a sequence of primitive symbols.

Concretely, we perform a depth-first-left-first traversal of formula trees to convert types to prefix (Polish) notation. This converts a type to a linear sequence of symbols $s \in \mathcal{V}$, where $\mathcal{V} = \mathcal{A} \cup \mathcal{D}$, the union of atomic types and dependency-decorated modal markings.[4] Proof frames can then be represented by joining individual type representations, separated with an extra-logical token `[SEP]` denoting type breaks and prefixed with a special token `[SOS]` to denote the sequence start (see the caption of Figure 3 for an example). The resulting sequence becomes the goal of a decoding process conditional on the input sentence, as implemented by a sequence-to-sequence model.

[4]Dependency decorations occur only within the scope of an implication, so the two are merged into a single symbol for reasons of length economy.

Treating supertagging as auto-regressive decoding enables the prediction of any valid type in the grammar, improving generalization and eliminating the need for a strictly defined type lexicon. Further, the decoder's comprehension of the type construction process can yield drastic improvements for beam search, allowing distinct branching paths within individual types. Most importantly, it grants access to the atomic sub-formulas of a sequent, i.e. the primitive entities to be paired within a proof net – a quality that will come into play when considering the axiom linking process later on.

3.2 Proof Structures

The conversion of a proof frame into a proof structure requires establishing a correct bijection between positive and negative atoms, i.e. linking each positive occurrence of an atom with a single unique negative occurrence of the same atom.

We begin by first noting that each atomic formula occurrence within a proof frame can be assigned an identifying index according to its position in the sequence (refer to the example of Figure 3). For each distinct atomic type, we can then create a table with rows enumerating negative and columns enumerating positive occurrences of that type, ordered by their indexes. We mark cells indexing linked occurrences and leave the rest empty; tables for our running example can be seen in Figure 5. The resulting tables correspond to a *permutation matrix* $\mathbf{\Pi}_{\mathrm{A}}$ for each atomic type A, i.e. a set of matrices that are square, binary and doubly-stochastic, encoding the permutation over the *chain* (i.e. ordered set) of negative elements that aligns them with the chain of matching positive elements. This key insight allows us to reframe automated proof search as learning the latent space that dictates the permutations between disjoint and non-contiguous sub-sequences of the primitive symbols constituting a decoded proof frame.

Permutation matrices are discrete mathematical objects that are not directly attainable by neural models. Their continuous relaxations are, however, valid outputs, approximated by means of the Sinkhorn operator (Sinkhorn, 1964). In essence, the operator and its underlying theorem state that the iterative normalization (alternating between rows and columns) of a square matrix with positive entries yields, in the limit, a doubly-stochastic matrix, the entries of which are *almost* binary. Put differently, the Sinkhorn operator gives rise to a

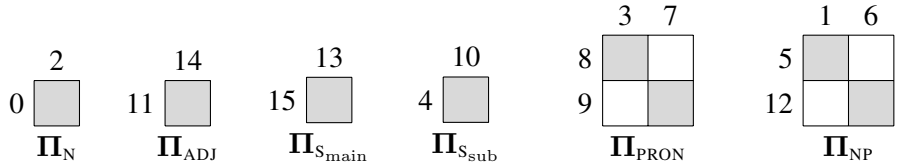

Figure 5: An alternative view of the axiom links of Figure 3, with tables $\mathbf{\Pi}_{\mathrm{N}}$, $\mathbf{\Pi}_{\mathrm{ADJ}}$, $\mathbf{\Pi}_{\mathrm{S_{main}}}$, $\mathbf{\Pi}_{\mathrm{S_{sub}}}$, $\mathbf{\Pi}_{\mathrm{PRON}}$, $\mathbf{\Pi}_{\mathrm{NP}}$ depicting the linked indices and corresponding permutations for each atomic type in the sentence.

non-linear activation function that applies on matrices, pushing them towards binarity and bistochasticity, analogous to a 2-dimensional softmax that preserves assignment (Mena et al., 2018). Moving to the logarithmic space eliminates the positive entry constraint and facilitates numeric stability through the log-sum-exp trick. In that setting, the Sinkhorn-normalization of a real-valued square matrix $\boldsymbol{X}$ is defined as:

$$\mathrm{Sinkhorn}(\boldsymbol{X}) = \lim_{\tau\to\infty} \exp\left(\mathrm{Sinkhorn}^{\tau}(\boldsymbol{X})\right)$$

where the induction is given by:

$$\mathrm{Sinkhorn}^{0}(\boldsymbol{X}) = \boldsymbol{X}$$

$$\mathrm{Sinkhorn}^{\tau}(\boldsymbol{X}) = \mathcal{T}_r\left(\mathcal{T}_r\left(\mathrm{Sinkhorn}^{(\tau-1)}(\boldsymbol{X})\right)^{\top}\right)$$

with $\mathcal{T}_r$ the row normalization in the log-space:

$$\mathcal{T}_r(\boldsymbol{X})_{i,j} = \boldsymbol{X}_{i,j} - log\sum_{r=0}^{N-1} e^{(\boldsymbol{X}_{r,j}-\max(\boldsymbol{X}_{r,:}))}$$

Bearing the above in mind, our goal reduces to assembling a matrix for each atomic type in a proof frame, with entries containing the unnormalized agreement scores of pairs in the cartesian product of positive and negative occurrences of that type. Given contextualized representations for each primitive symbol within a proof frame, scores can be simply computed as the inter-representation *dot-product attention*. Assuming, for instance, $\boldsymbol{I}_{\mathrm{A}}^{+}$ and $\boldsymbol{I}_{\mathrm{A}}^{-}$ the vectors indexing the positions of all a positive and negative occurrences of type A in a proof frame sequence, we can arrange the matrices $\boldsymbol{P}_{\mathrm{A}}, \boldsymbol{N}_{\mathrm{A}} \in \mathbb{R}^{a\times d}$ containing their respective contextualized d-dimensional representations (recall that the count invariance property asserts equal shapes). The dot-product attention matrix containing their element-wise agreements will then be given as $\tilde{\boldsymbol{S}}_{\mathrm{A}} = \boldsymbol{P}_{\mathrm{A}}\boldsymbol{N}_{\mathrm{A}}^{\top} \in \mathbb{R}^{a\times a}$. Applying the Sinkhorn operator, we obtain $\boldsymbol{S}_{\mathrm{A}} = \mathrm{Sinkhorn}(\tilde{\boldsymbol{S}}_{\mathrm{A}})$, which, in our setup, will be modeled as a continuous approximation of the underlying permutation matrix $\mathbf{\Pi}_{\mathrm{A}}$.

3.3 Implementation

Encoder-Decoder We first encode sentences using BERTje (de Vries et al., 2019), a pretrained BERT-Base model (Devlin et al., 2019) localized for Dutch. We then decode into proof frame sequences using a Transformer-like decoder (Vaswani et al., 2017).

Symbol Embeddings In order to best utilize the small, structure-rich vocabulary of the decoder, we opt for lower-dimensional, position-dependent symbol embeddings. We follow insights from Wang et al. (2020) and embed decoder symbols as continuous functions in the complex space, associating each output symbol $s \in \mathcal{V}$ with a magnitude embedding $\boldsymbol{r_s} \in \mathbb{R}^{128}$ and a frequency embedding $\boldsymbol{\omega_s} \in \mathbb{R}^{128}$. A symbol s occurring in position p in the proof frame is then assigned a vector $\tilde{\boldsymbol{v}}_{\boldsymbol{s},\boldsymbol{p}} = \boldsymbol{r_s} e^{j\boldsymbol{\omega_s} p} \in \mathbb{C}^{128}$. We project to the decoder's vector space by concatenating the real and imaginary parts, obtaining the final representation as $\boldsymbol{v}_{\boldsymbol{s},\boldsymbol{p}} = \mathrm{conc}(\Re(\tilde{\boldsymbol{v}}_{\boldsymbol{s},\boldsymbol{p}}), \Im(\tilde{\boldsymbol{v}}_{\boldsymbol{s},\boldsymbol{p}})) \in \mathbb{R}^{256}$.

Tying the embedding parameters with those of the pre-softmax transformation reduces the network's memory footprint and improves representation quality (Press and Wolf, 2017). In duality to the input embeddings, we treat output embeddings as functionals parametric to positions. To classify a token occurring in position p, we first compute a matrix $\boldsymbol{V_p}$ consisting of the local embeddings of all vocabulary symbols, $\boldsymbol{V_p} = \boldsymbol{v}_{:,\boldsymbol{p}} \in \mathbb{R}^{||\mathcal{V}||\times 256}$. The transpose of that matrix acts then as a linear map from the decoder's representation to class weights, from which a probability distribution is obtained by application of the softmax function.

Proof Frame Contextualization Proof frames may generally give rise to more than one distinct proof, with only a portion of those being linguistically plausible. Frames eligible to more than one potential semantic reading can be disambiguated by accounting for statistical preferences, as exhibited by lexical cues. Consequently, we need our

contextualization scheme to incorporate the sentential representation in its processing flow. To that end, we employ another Transformer decoder, now modified to operate with no causal mask, thus allowing all decoded symbols to freely attend over one another regardless of their relative position. This effectively converts it into a *bi-modal encoder* which operates on two input sequences of different length and dimensionality, namely the BERT output and the sequence of proof frame symbols, and constructs contextualized representations of the latter as informed by the former.

Axiom Linking We index the contextualized proof frame to obtain a pair of matrices for each distinct atomic type in a sentence, easing the complexity of the problem by preemptively dismissing the possibility of linking unequal types; this also alleviates performance issues noted when permuting sets of high cardinality (Mena et al., 2018). Post contextualization, positive and negative items are projected to a lower dimensionality via a pair of feed-forward neural functions, applied token-wise. Normalizing the dot-product attention weights between the above with Sinkhorn yields our final output.

4 Experiments

We train, validate and test our architecture on the corresponding subsets of the Æthel dataset, filtering out samples the proof frames of which exceed 100 primitive symbols. Implementation details and hyper-parameter tables, an illustration of the full architecture, dataset statistics and example parses are provided in Appendix A.[5]

4.1 Training

We train our architecture end-to-end, including all BERT parameters apart from the embedding layer, using AdamW (Loshchilov and Hutter, 2018).

In order to jointly learn representations that accommodate both the proof-frame and the proof-structure outputs, we back-propagate a loss signal derived as the addition of two loss functions. The first is the Kullback-Leibler divergence between the predicted proof frame symbols and the label-smoothed ground-truth distribution (Müller et al., 2019). The second is the negative log-likelihood between the Sinkhorn-activated dot-product weights and the corresponding binary-valued permutation matrices.

Throughout training, we validate by measuring the per-symbol and per-sentence typing accuracy of the greedily decoded proof frame, as well as the linking accuracy under the assumption of an error-free decoding. We perform model selection on the basis of the above metrics and reach convergence after approximately 300 epochs.

4.2 Testing

We test model performance using beam search. For each input sentence, we consider the β best decode paths, with a path's score being the sum of its symbols' log probabilities, counting all symbols up to the last expected `[SEP]` token. Neural decoding is followed by a series of filtering steps. We first parse the decoded symbol sequences, discarding beams containing subsequences that do not meet the inductive constructors of the type grammar. The atomic formulas of the passing proof frames are polarized according to the process of §2.3. Frames failing to satisfy the count invariance property are also discarded. The remaining ones constitute potential candidates for a proof structure; their primitive symbols are contextualized by the bimodal encoder, and are then used to compute soft axiom link strengths between atomic formulas of matching types. Discretization of the output yields a graph encoding a proof structure; we follow the net traversal algorithm of Lamarche (2008) to check whether it is a valid proof net, and, if so, produce the λ-term in the process (de Groote and Retoré, 1996). Terms generated this way contain no redundant abstractions, being in β-normal η-long form.

4.3 Analysis

Table 1 presents a breakdown of model performance at different beam widths. To evaluate model performance, we use the first valid beam of each sample, defaulting to the highest scoring beam if none is available. On the token level, we report *supertagging accuracy*, i.e. the percentage of types correctly assigned. We further measure the percentage of samples satisfying each of the following sentential metrics: 1) *invariance property*, a condition necessary for being eligible to a proof structure, 2) *frame correctness*, i.e. whether the decoded frame is identical to the target frame, meaning all types assigned are the correct ones, 3) *untyped term accuracy*, i.e. whether, regardless of the proof frame,

[5] The implementing code can be found at `github.com/konstantinosKokos/neural-proof-nets`.

Metric (%)	**Beam Size** β					**Baseline**
	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 5$	$\beta = 7$	*alpino*
Token Level						
Types Correct	85.5	91.4	92.4	93.2	93.4	56.2
Sentence Level						
Invariance Correct	87.6	93.4	94.9	96.1	96.6	*n/a*
Frame Correct	57.6	65.3	68.0	69.6	70.2	*n/a*
Term Correct (w/o types)	60.0	65.6	67.7	69.1	69.6	45.7
Term Correct (/w types & deps)	56.9	63.7	65.9	67.1	67.6	30.4

Table 1: Test set model performance broken down by beam size, and baseline comparison.

the untyped λ-term coincides with the true one, and 4) *typed term accuracy*, meaning that both the proof frame and the untyped term are correct.

Numeric comparisons against other works in the literature is neither our prime goal nor an easy task; the dataset utilized is fairly recent, the novelty of our methods renders them non-trivial to adapt to other settings, and ILL-friendly categorial grammars are not particularly common in experimental setups. As a sanity check, however, and in order to obtain some meaningful baselines, we employ the Alpino parser (Bouma et al., 2001). Alpino is a hybrid parser based on a sophisticated handwritten grammar and a maximum entropy disambiguation model; despite its age and the domain difference, Alpino is competitive to the state-of-the-art in UD parsing, remaining within a 2% margin to the last reported benchmark (Bouma and van Noord, 2017; Che et al., 2018). We pair Alpino with the extraction algorithm used to convert its output into $\text{ILL}_{\multimap,\Diamond,\Box}$ derivations (Kogkalidis et al., 2020); together, the two faithfully replicate the data generating process our system has been trained on, modulo the manual correction phase of van Noord et al. (2013). We query Alpino for the globally optimal parse of each sample in the test set (enforcing no time constraints), perform the conversion and log the results in Table 1.

Our model achieves remarkable performance even in the greedy setting, especially considering the rigidity of our metrics. Untyped term accuracy conveys the percentage of sentences for which the function-argument structure has been perfectly captured. Typed term accuracy is even stricter; the added requirement of a correct proof frame practically translates to no erroneous assignments of part-of-speech and syntactic phrase tags or dependency labels. Keeping in mind that dependency information are already incorporated in the proof frame, obtaining the correct proof structure fully subsumes dependency parsing.

The filtering criteria of the previous paragraph yield significant benefits when combined with beam search, allowing us to circumvent logically unsound analyses regardless of their sequence scores. It is worth noting that our metrics place the model's bottleneck at the supertagging rather than the permutation component. Term accuracy closely follows along (and actually surpasses, in the untyped case) frame accuracy. This is further evidenced when providing the ground truth types as input to the parser, in which case term accuracy reaches as high as 85.4%, indicative of the high expressive power of Sinkhorn on top of the the bi-modal encoder's contextualization. On the negative side, the strong reliance on correct type assignments means that a single mislabeled word can heavily skew the parse outcome, but also hints at increasing returns from improvements in the decoding architecture.

5 Related Work

Our work bears semblances to other neural methodologies related to syntactic/semantic parsing. Sequence-to-sequence models have been successfully employed in the past to decode directly into flattened representations of parse trees (Wiseman and Rush, 2016; Buys and Blunsom, 2017; Li et al., 2018). In dependency parsing literature, head selection involves building word representations that act as classifying functions over other words (Zhang et al., 2017), similar to our dot-product weighting between atoms.

Akin to graph-based parsers (Ji et al., 2019; Zhang et al., 2019), our model generates parse structures in the form of graphs. In our case, how-

ever, graph nodes correspond to syntactic primitives (atomic types & dependencies) rather than words, while the discovery of the graph structure is subject to hard constraints imposed by the decoder's output.

Transcription to formal expressions (logical forms, λ-terms, database queries and executable program instructions) has also been a prominent theme in NLP literature, using statistical methods (Zettlemoyer and Collins, 2012) or structurally-constrained decoders (Dong and Lapata, 2016; Xiao et al., 2016; Liu et al., 2018; Cheng et al., 2019). Unlike prior approaches, the decoding we employ here is unhindered by explicit structure; instead, parsing is handled in parallel across the entire sequence by the Sinkhorn operator, which biases the output towards structural correctness while requiring neither backtracking nor iterative processing. More importantly, the λ-terms we generate are not in themselves the product of a neural decoding process, but rather a corollary of the isomorphic relation between $\mathrm{ILL}_{\multimap}$ proofs and linear λ-calculus programs.

In machine learning literature, Sinkhorn-based networks have been gaining popularity as a means of learning latent permutations of visual or synthetic data (Mena et al., 2018) or imposing permutation invariance for set-theoretic learning (Grover et al., 2019), with so far limited adoption in the linguistic setting (Tay et al., 2020; Swanson et al., 2020). In contrast to prior applications of Sinkhorn as a final classification layer, we use it over chain element representations that have been mutually contextualized, rather than set elements vectorized in isolation. Our benchmarks, combined with the assignment-preserving property of the operator, hint towards potential benefits from adopting it in a similar fashion across other parsing tasks.

6 Conclusion

We have introduced neural proof nets, a data-driven perspective on the proof nets of $\mathrm{ILL}_{\multimap}$, and successfully employed them on the demanding task of transcribing raw text to proofs and computational terms of the linear λ-calculus. The terms construed constitute type-safe abstract program skeletons that are free to interpret within arbitrary domains, fulfilling the role of a practical intermediary between text and meaning. Used as-is, they can find direct application in logic-driven models of natural language inference (Abzianidze, 2016).

Our architecture marks a departure from other parsing approaches, owing to the novel use of the Sinkhorn operator, which renders it both fully parallel and backtrack-free, but also logically grounded. It is general enough to apply to a variety of grammar formalisms inheriting from linear logic; if augmented with Gumbel sampling (Mena et al., 2018), it can further a provide a probabilistic means to account for derivational ambiguity. Viewed as a means of exposing deep tecto-grammatic structure, it paves the way for graph-theoretic approaches at syntax-aware sentential meaning representations.

Acknowledgements

We would like to thank the anonymous reviewers for their detailed feedback, which helped improve the presentation of the paper. Konstantinos and Michael are supported by the Dutch Research Council (NWO) under the scope of the project "A composition calculus for vector-based semantic modelling with a localization for Dutch" (360-89-070).

References

Lasha Abzianidze. 2016. Natural solution to FraCaS entailment problems. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*, pages 64–74, Berlin, Germany. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450v1*.

Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.

Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.

Gosse Bouma and Gertjan van Noord. 2017. Increasing return on annotation investment: The automatic construction of a Universal Dependency treebank for Dutch. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 19–26, Gothenburg, Sweden. Association for Computational Linguistics.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide-coverage computational analysis of dutch. In *Computational linguistics in the Netherlands 2000*, pages 45–59. Brill Rodopi.

Nicolaas Govert de Bruijn. 1979. Wiskundigen, let op uw Nederlands. *Euclides*, 55(juni/juli):429–435.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2019. Learning an executable neural semantic parser. *Computational Linguistics*, 45(1):59–94.

Vincent Danos and Laurent Regnier. 1989. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43.

Jean-Yves Girard. 1987. Linear logic. *Theoretical computer science*, 50(1):1–101.

Jean-Yves Girard, Yves Lafont, and P. Taylor. 1988. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 252–259.

Philippe de Groote and Christian Retoré. 1996. On the semantic readings of proof-nets. In *Proceedings Formal grammar*, pages 57–70, Prague, Czech Republic. FoLLI.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*.

Stefano Guerrini. 1999. Correctness of multiplicative proof nets is linear. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 454–263. IEEE Computer Science Society.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485.

Max I. Kanovich. 1994. The complexity of horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69(2-3):195–241.

Konstantinos Kogkalidis, Michael Moortgat, and Tejaswini Deoskar. 2019. Constructive type-logical supertagging with self-attention networks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 113–123.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020. ÃĘthel: Automatically extracted typelogical derivations for dutch. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5259–5268, Marseille, France. European Language Resources Association.

Ysuke Kubota and Robert Levine. 2020. *Type-Logical Syntax*. MIT Press.

François Lamarche. 2008. Proof nets for intuitionistic linear logic: Essential nets. Research report, INRIA Nancy.

Joachim Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214.

Patrick Lincoln. 1995. Deciding provability of linear logic formulas. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, pages 109–122. Cambridge University Press.

Jiangming Liu, Shay B Cohen, and Mirella Lapata. 2018. Discourse representation structure parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 429–439.

Ilya Loshchilov and Frank Hutter. 2018. Fixing weight decay regularization in adam.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning latent permutations with Gumbel-Sinkhorn networks. In *International Conference on Learning Representations*.

Michael Moortgat. 1996. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4):349–385.

Glyn Morrill. 2014. A categorial type logic. In *Categories and Types in Logic, Language, and Physics - Essays Dedicated to Jim Lambek on the Occasion of His 90th Birthday*, volume 8222 of *Lecture Notes in Computer Science*, pages 331–352. Springer.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems*, pages 4696–4705.

Andrzej S. Murawski and C.-H. Luke Ong. 2000. Dominator trees and fast verification of proof nets. In *Logic in Computer Science*, pages 181–191.

Reinhard Muskens. 2001. Lambda grammars and the syntax-semantics interface. In *Proceedings of the 13th Amsterdam Colloquium*, pages 150–155.

Reinhard Muskens and Mehrnoosh Sadrzadeh. 2018. Static and dynamic vector semantics for lambda calculus models of natural language. *Journal of Language Modelling*, 6(2):319–351.

Gertjan van Noord, Gosse Bouma, Frank van Eynde, Daniel de Kok, Jelmer van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste. 2013. Large scale syntactic annotation of written dutch: Lassy. In *Essential speech and language technology for Dutch*, pages 147–164. Springer, Berlin, Heidelberg.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163.

Dirk Roorda. 1991. *Resource Logics: Proof-theoretical Investigations*. Ph.D. thesis, Universiteit van Amsterdam.

Richard Sinkhorn. 1964. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879.

Morten Heine Sørensen and Pawel Urzyczyn. 2006. *Lectures on the Curry-Howard isomorphism*. Elsevier.

Kyle Swanson, Lili Yu, and Tao Lei. 2020. Rationalizing text matching: Learning sparse alignments via optimal transport. *arXiv preprint arXiv:2005.13111*.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020. Sparse sinkhorn attention. *arXiv preprint arXiv:2002.11296v1*.

Anne Sjerp Troelstra and Helmut Schwichtenberg. 2000. *Basic Proof Theory*, 2 edition, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with lstms. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. BERTje: A Dutch BERT model. *arXiv preprint arXiv:1912.09582v1*.

Philip Wadler. 1993. A taste of linear logic. In *International Symposium on Mathematical Foundations of Computer Science*, pages 185–210. Springer.

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. 2020. Encoding word order in complex embeddings. In *International Conference on Learning Representations*.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. Ccg supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420v1*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 665–676.

A Appendix

A.1 Model

Table 2 presents model hyper-parameters, as selected by greedy grid search. An illustration of the model can be seen in Figure 6.

Parameter	Value
BERTje (*BERT-Base*)	
# Layers	12
# Self-attention heads	12
Feed-forward dimensionality	3 072
Feed-forward activation	GELU
Input/output dimensionality	768
Vocabulary size	30 000
Decoder	
# Layers	3
# Self-attention heads	8
# Encoder-attention heads	8
Feed-forward dimensionality	512
Input/output dimensionality	256
Vocabulary size	58
Bi-modal Encoder	
# Layers	1
# Self-attention heads	8
# Encoder-attention heads	8
Feed-forward dimensionality	512
Feed-forward activation	GELU
Input/output dimensionality	256
Pre-Sinkhorn Transformations	
Input/Feed-forward dimensionality	256
Feed-forward activation	GELU
Output dimensionality	32
Output activation	LayerNorm

Table 2: Model hyper-parameters

A.2 Optimization

We train with an adaptive learning rate following Vaswani et al. (2017), such that the learning rate at optimization step i is given as:

$$768^{-0.5} \cdot \min\left(i^{-0.5},\ i \cdot \text{warmup\_steps}^{-1.5}\right)$$

For BERT parameters, learning rate is scaled by 0.1. We freeze the oversized word embedding layer to reduce training costs and avoid overfitting. Optimization hyper-parameters are presented in Table 3.

We provide strict teacher guidance when learning axiom links, whereby the network is provided with the original proof frame symbol sequence instead of the predicted one. To speed up computation, positive and negative indexes are arranged per-length rather than type for each batch; this allows us to process symbol transformations, dot-product attentions and Sinkhorn activations in parallel for many types across many sentences. During training, we set the number of Sinkhorn iterations to 5; lower values are more difficult to reach convergence with, hurting performance, whereas higher values can easily lead to vanishing gradients, impeding learning (Grover et al., 2019).

Parameter	Value
Batch size	32
Warmup epochs	5
Weight decay	10^{-5}
Weight decay (BERT)	0
LR scale (BERT)	0.1
LR scale (BERT embedding)	0
Dropout rate	0.1
Label smoothing	0.1

Table 3: Optimizer hyper-parameters

A.3 Data

Figure 7 presents cumulative distributions of dataset statistics. The kept portion of the dataset corresponds to roughly 97% of the original, enumerating 55 683 training, 6 971 validation and 6 957 test samples.

A.4 Performance

Table 4 summarizes the model's performance in terms of untyped term accuracy over the test set in the greedy setting, binned according to input sentence lengths. Table 5 presents input-output pairs from sample sentences not included in the dataset.

Sentence Length	Total	Correct	(%)
1 – 5	808	743	92
5 – 10	1 491	1 104	74
10 – 15	1 576	919	58
15 – 20	1 206	501	42
20 –	592	154	26

Table 4: Test set model performance broken down by sentence length.

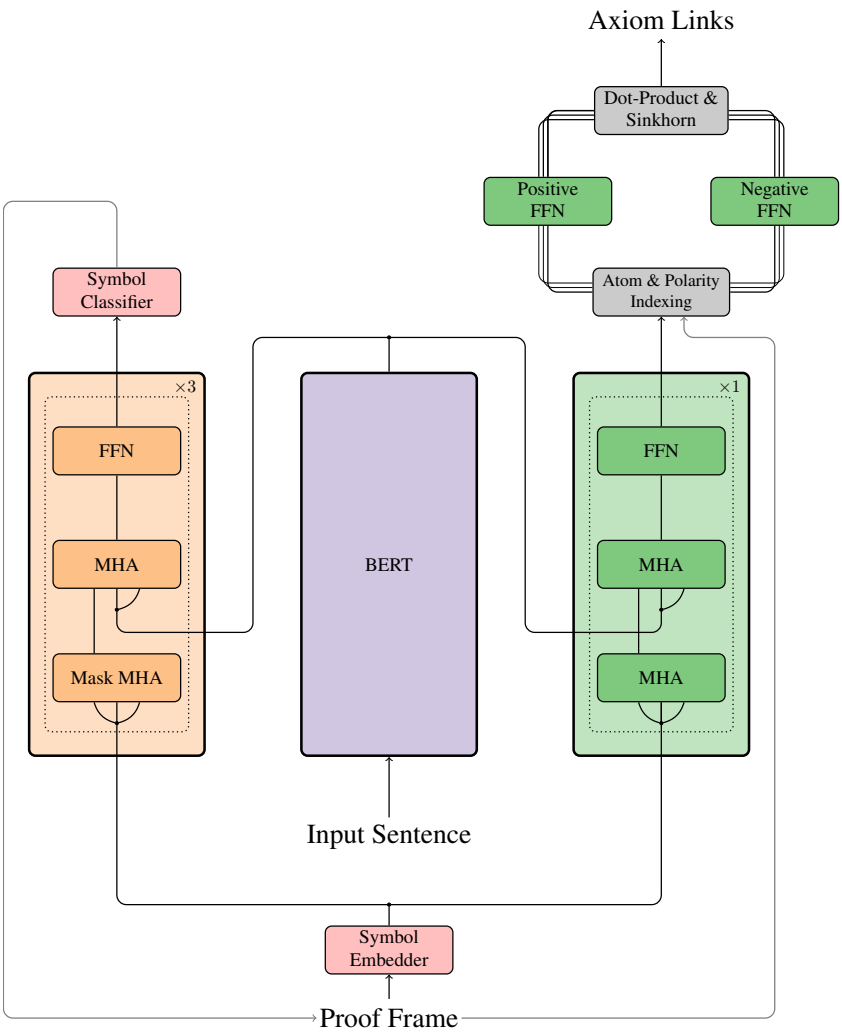

Figure 6: Schematic diagram of the full network architecture. The supertagger (orange, left) iteratively generates a proof frame by attending over the currently available part of it plus the full input sentence. The axiom linker (green, right) contextualizes the complete proof frame by attending over it as well as the sentence. Representations of atomic formulas are gathered and transformed according to their polarity, and their Sinkhorn-activated dot-product attention is computed. Discretization of the result yields a permutation matrix denoting axiom links for each unique atomic type in the proof frame. The final output is a proof structure, i.e. the pair of a proof frame and its axiom links.

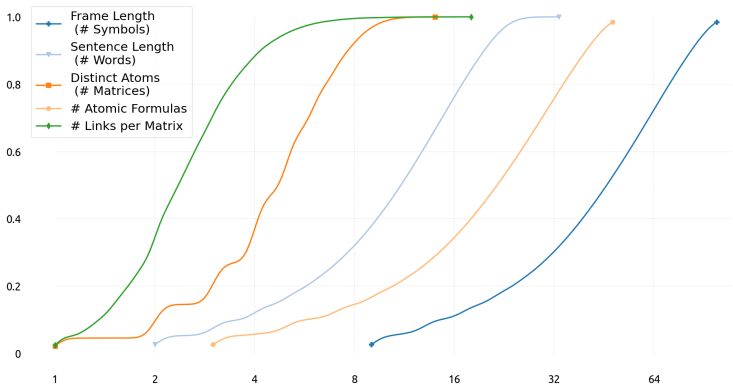

Figure 7: log2-transformed cumulative distributions of symbol and word lengths, counts of atomic formulas, matrices and matrix sizes from the portion of the dataset trained on.

De voorafgaande stukjes over Wiskundige Omgangstaal hadden het vooral over het samenspel tussen woorden en formules.
"The preceding articles on the Mathematical Vernacular mainly focused on the interplay between words and formules."

$((\texttt{hadden} :: \text{PP} \multimap \text{PRON} \multimap \text{S}_{\text{main}}\ ((\texttt{vooral} :: \text{PP} \multimap \text{PP})_{\text{mod}}\ (\texttt{over} :: \text{NP} \multimap \text{PP}\ ((\texttt{tussen} :: \text{WW} \multimap \text{NP} \multimap \text{NP}\ ((\texttt{en} :: \text{NP} \multimap \text{NP} \multimap \text{NP}\ (\texttt{woprden} :: \text{NP})^{\text{cnj}})\ (\texttt{formules} :: \text{NP})^{\text{cnj}})^{\text{obj}})_{\text{mod}}\ ((\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{samenspel} :: \text{N}))^{\text{obj}}))^{\text{pc}}\ (\texttt{het} :: \text{PRON})^{\text{obj}})\ ((\texttt{over} :: \text{NP} \multimap \text{NP} \multimap \text{NP}\ (\texttt{Wiskundige\_Omgangstaal} :: \text{NP})^{\text{obj}})_{\text{mod}}\ ((\texttt{voorafgaande} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ ((\texttt{De} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{stukjes} :: \text{N})))^{\text{su}}$

In het wiskundig Nederlands worden vaak dezelfde fouten gemaakt als in het gewone Nederlands.
"The same mistakes are often made in mathematical Dutch as in common Dutch." $(\texttt{worden} :: \text{PPART} \multimap \text{NP} \multimap \text{S}_{\text{main}}\ ((\texttt{vaak} :: \text{PPART} \multimap \text{PPART})_{\text{mod}}\ ((\texttt{In} :: \text{NP} \multimap \text{PPART} \multimap \text{PPART}\ ((\texttt{wiskundig} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ ((\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{Nederlands} :: \text{N}))^{\text{obj}})_{\text{mod}}\ \texttt{gemaakt} :: \text{PPART}))^{\text{vc}})\ ((\texttt{dezelfde} :: \text{CP} \multimap \text{N} \multimap \text{NP}\ (\texttt{als} :: \text{PP} \multimap \text{CP}\ (\texttt{in} :: \text{NP} \multimap \text{PP}\ ((\texttt{gewone} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ ((\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{Nederlands} :: \text{N}))^{\text{obj}})^{\text{cmp\_body}})^{\text{obcomp}})_{\text{det}}\ (\texttt{fouten} :: \text{N})^{\text{su}}$

In het wiskundige taalgebruik is er meestal een scheiding aan te brengen tussen de echte wiskundige taal en de taal waarmee we over die wiskundige taal of over het wiskundige bedrijf spreken.
"In mathematical discourse, there is usually a distinction to be made between the real mathematical language and the language with which we speak about the mathematical language or about the mathematical practice."
–

Probeer zinnen steeds zo te stellen dat ze alleen op de door de schrijver bedoelde wijze zijn terug te lezen.
"Try to always formulate sentences in such a way that they can only be read in the manner intended by the author."
–

In het Nederlands kunnen vele zinnen wat volgorde betreft omgegooid worden.
"In Dutch, many sentences can be restructured as far as order is concerned."

$(\texttt{kunnen} :: \text{INF} \multimap \text{NP} \multimap \text{S}_{\text{main}}\ (\texttt{worden} :: \text{PPART} \multimap \text{INF}\ ((\texttt{wat} :: (\text{PRON} \multimap \text{S}_{\text{sub}}) \multimap \text{PPART} \multimap \text{PPART}\ \lambda x_0^{\text{su}}.((\texttt{betreft} :: \text{N} \multimap \text{PRON} \multimap \text{S}_{\text{sub}}\ (\texttt{volgorde} :: \text{N})^{\text{obj}})\ x_0)^{\text{rel\_body}})_{\text{mod}}\ ((\texttt{In} :: \text{NP} \multimap \text{PPART} \multimap \text{PPART}\ ((\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{Nederlands} :: \text{N})^{\text{obj}})_{\text{mod}}\ \texttt{omgegooid} :: \text{PPART}))^{\text{vc}})^{\text{vc}})\ ((\texttt{vele} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ \texttt{zinnen} :: \text{NP})^{\text{su}}$

In het Nederlands kunnen vaak twee zinnen tot èèn kortere worden samengetrokken.
"In Dutch, two sentences can often be merged into a shorter one."

$(\texttt{kunnen} :: \text{INF} \multimap \text{NP} \multimap \text{S}_{\text{main}}\ (\texttt{worden} :: \text{PPART} \multimap \text{INF}\ ((\texttt{vaak} :: \text{PPART} \multimap \text{PPART})_{\text{mod}}\ ((\texttt{In} :: \text{NP} \multimap \text{PPART} \multimap \text{PPART}\ ((\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{Nederlands} :: \text{N})^{\text{obj}})_{\text{mod}}\ (\texttt{samengetrokken} :: \text{PP} \multimap \text{PPART}\ (\texttt{tot} :: \text{NP} \multimap \text{PP}\ ((\texttt{èèn} :: \text{ADJ} \multimap \text{NP})_{\text{det}}\ \texttt{kortere} :: \text{ADJ})^{\text{obj}})^{\text{ld}}))^{\text{vc}})^{\text{vc}})\ ((\texttt{twee} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{zinnen} :: \text{N})^{\text{su}}$

Populaire taal is vaak minder beveiligd tegen dubbelzinnigheid dan nette taal, en het mengsel van beide talen is nòg gevaarlijker.
"Informal language is often less protected against ambiguity than formal language, and the mixture of both languages is even more dangerous."

$(\texttt{en} :: \text{S}_{\text{main}} \multimap \text{S}_{\text{main}} \multimap ((\texttt{is} :: \text{PPART} \multimap \text{NP} \multimap \text{S}_{\text{main}}\ ((\texttt{minder} :: \text{CP} \multimap \text{PPART} \multimap \text{PPART}\ (\texttt{dan} :: \text{NP} \multimap \text{CP}\ ((\texttt{nette} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ \texttt{taal} :: \text{NP})^{\text{cmp\_body}})^{\text{obcomp}})_{\text{mod}}\ ((\texttt{vaak} :: \text{PPART} \multimap \text{PPART})_{\text{mod}}\ (\texttt{beveiligd} :: \text{PP} \multimap \text{PPART}\ ((\texttt{tegen} :: \text{NP} \multimap \text{PP}\ (\texttt{dubbelzinnighead} : \text{NP})^{\text{obj}})^{\text{pc}})))^{\text{vc}}\ ((\texttt{Populaire} :: \text{NP} \multimap \text{NP})_{\text{mod}}\ \texttt{taal} :: \text{NP})^{\text{su}})^{\text{cnj}})\ ((\texttt{is} :: \text{AP} \multimap \text{NP} \multimap \text{S}_{\text{main}}\ ((\texttt{nòg} :: \text{AP} \multimap \text{AP})_{\text{mod}}\ \texttt{gevaarlijker} :: \text{AP})^{\text{predc}}\ ((\texttt{van} :: \text{NP} \multimap \text{NP} \multimap \text{NP}\ ((\texttt{beide} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{talen} :: \text{N})^{\text{obj}})_{\text{mod}}\ (\texttt{het} :: \text{N} \multimap \text{NP})_{\text{det}}\ \texttt{mengsel} :: \text{N}))^{\text{su}})^{\text{cnj}}$

Table 5: Greedy parses of the opening sentences of the first seven paragraphs of de Bruijn (1979), in the form of type- and dependency-annotated λ expressions. Two of them (3 & 4) yield no valid proof net; the remaining five are both valid and correct.