Deductive Parsing with an Unbounded Type Lexicon

Konstantinos Kogkalidis, Michael Moortgat, Richard Moot, Giorgos Tziafas

DEDUCTIVE PARSING WITH AN UNBOUNDED TYPE LEXICON

KONSTANTINOS KOGKALIDIS
Utrecht Institute of Linguistics OTS, 3512 JK Utrecht, the Netherlands
k.kogkalidis@uu.nl

MICHAEL MOORTGAT
Utrecht Institute of Linguistics OTS, 3512 JK Utrecht, the Netherlands
m.j.moortgat@uu.nl

RICHARD MOOT
LIRMM, Université de Montpellier, CNRS, France
richard.moot@lirmm.fr

GIORGOS TZIAFAS
University of Groningen, 9700 AB Groningen, the Netherlands
g.tziafas@student.rug.nl

Abstract

We present a novel deductive parsing framework for categorial type logics, modeled as the composition of two components. The first is an attention-based neural supertagger, which assigns words dependency-decorated, contextually informed linear types. It requires no predefined type lexicon, instead utilizing the type syntax to construct types inductively, enabling the use of a richer and more precise typing environment. The type annotations produced are used by the second component, a computationally efficient hybrid system that emulates the inference process of the type logic, iteratively producing a bottom-up reconstruction of the input's derivation-proof and the associated program for compositional meaning assembly. Initial experiments yield promising results for each of the components.

Vol. \jvolume No. \jnumber \jyear
Journal of Applied Logics — IfCoLog Journal of Logics and their Applications

1 Parsing as deduction

The type-logical strand of categorial grammar is based on a proof-theoretic view on natural language syntax and semantics: determining whether a phrase is syntactically well-formed amounts to a process of logical deduction deriving its type from the types of its constituent parts. The Curry-Howard correspondence between proofs and computations then effectively turns this logical deduction into a *program* assembling the meaning of the parsed phrase in a compositional way [10].

What counts as a valid deduction depends on the type logic used. The type logic we aim for is a dependency-enhanced version of the simply typed fragment of Multiplicative Intuitionistic Linear Logic (MILL, [2]), also known as the Lambek-Van Benthem calculus [1].

The type language then has atomic types, and complex types formed with linear implication. Atomic types are assigned to phrases that are considered 'complete', e.g. NP for noun phrase, PRON for pronoun, etc. Complex types, on the other hand, are binary functors that compose with an argument phrase to produce a larger one.

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \to B}{\Gamma, \Delta \vdash B} \to E \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I$$

Table 1: Linear implication: Elimination and Introduction rules

The natural deduction rules for linear implication are shown in Table 1. Judgements are of the form $\Gamma \vdash B$, with $\Gamma = A_1, \ldots, A_n$ a multiset of type assumptions from which one derives conclusion B. Computationally, Implication Elimination corresponds to function application and Introduction to function abstraction on the interpreting meaning spaces. Notice that no intermediate syntax-semantics homomorphism is required to map the syntactic space onto to the semantic space; our types already constitute the type signatures of the latter.

In what follows, we will decorate the linear implication with a fixed set of dependency labels denoting grammatical relations such as subject (su) and object (obj). For instance, $\text{NP} \xrightarrow{\text{su}} \text{S}$ corresponds to a functor that consumes a noun-phrase carrying the subject role to produce a sentence — an intransitive verb. By extending the formula language with modal operators as in [8], it would be possible to treat $A \xrightarrow{\text{d}} B$ formally as $\Diamond_d A \to B$. For the purposes of this paper, we will treat the dependency labels simply as an annotation that provides useful information for the parser.

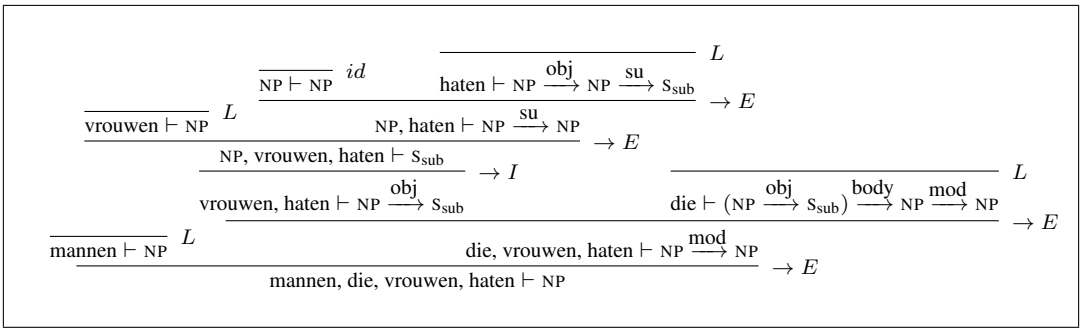

Figure 1: Derivation for the object-relative typing of the example sentence "mannen die vrouwen haten" (*men whom women hate*). Axiom leaves are either marked L for lexical type assignments, or id for identities of hypotheses. S_{sub} is the type of subordinate clauses.

2 Experimental Setup

Our experiments focus on Dutch. The relatively high degree of word order freedom of the language is a feature that makes our choice for a non-directional type logic pay off. Our dataset is a treebank of approximately 65000 annotated sentences of written Dutch, originating from the Lassy-Small corpus [12]. The annotations are DAGs with syntactic category labels at the nodes and dependency labels at the edges; re-entrancy is used for the annotation of unbounded dependencies and related phenomena, obviating the need for phantom syntactic elements (gaps, traces etc).

We extract type annotations by implementing an adaptation of Moortgat and Moot's algorithm [9] . Atomic types are the result of a translation mapping on the part-of-speech tags and phrasal category labels of the Lassy DAGs. Phrasal heads are assigned implicational functor types selecting their dependents. A pre-specified obliqueness hierarchy [5] determines in which order the arguments are consumed in the case of multi-argument functors. The algorithm turns an input DAG into a type-sequence for the lexical items at its yield. The resulting sequences enumerate a total of 5700 unique types, made out of 22 binary connectives and 30 atomic types.

As remarked earlier, our main deviation from standard categorial practice is to collapse the directed implication connectives $/, \backslash$ into a single linear implication. Part of the lost information is re-encoded by naming each implication in accordance to the label of the dependency it is implementing. In practical terms, this means that resolving dependencies no longer needs to be solved explicitly by a future parser, as dependencies are now represented internally within the grammar's type system. Figure 1 depicts an example derivation of our grammar.

3 Supertagging Unbounded Type Lexicons

Our extracted grammar boasts a simple yet rich type system; its fine-grained nature is manifested by the large number of unique types extracted. This has the negative side-effect that most types occur infrequently, making them hard to learn for standard supertaggers. Supertaggers are usually envisioned as neural sequence labeling systems, which iterate over a sentence and assign each of its words a type (i.e. a label) from a finite set. They operate under the assumption that the possible set of labels is bounded and known a priori, thus failing to account for types that are not present in the corpus and struggling with types for which only a small number of training instances is present. We briefly report on earlier work[1], an architecture that does away with the aforementioned problems by making a subtle adjustment to the usual problem formulation.

3.1 The Language of Types

We begin by noting that despite their sparsity, our extracted types are characterized by an important regularity; they are all constructed by a common underlying inductive scheme. This scheme may be viewed as a context-free grammar, in turn defining a language, the words of which are types. Even though the set of types is (potentially) infinite, the grammar's terminal symbols (being either atomic types or binary connectives) are themselves finite, and known in advance. Taking this into account, we orient our architecture towards learning how types are dynamically constructed under the context of an input sentence. Successfully doing so provides the means for supertagging without an explicitly defined lexicon, allowing correct type assignments even for rare and previously unseen types, thereby bypassing the technical issues caused by larger and sparser lexicons.

3.2 Supertagging is Sequence Transduction

With this new insight, we design a neural architecture that treats the supertagging problem as sequence transduction, rather than sequence labeling. Essentially, our goal is a system that accepts a sequence of N words as its input, and produces a sequence of M symbols as output (with $M \geq N$), where the output symbols represent the sequence of formulas (supertags) using a sequence of atomic types, binary connectives, and an auxiliary separator that marks full type boundaries (parentheses are discarded by adopting the polish notation). Neural machine translation architectures naturally lend themselves to this problem specification (another way of putting this is that we want to accomplish a translation from a source natural language onto the language defined by our type grammar). Given the combination of short and long distance dependencies present in the task (stemming from the type-level CFG and

[1] Citation currently under embargo.

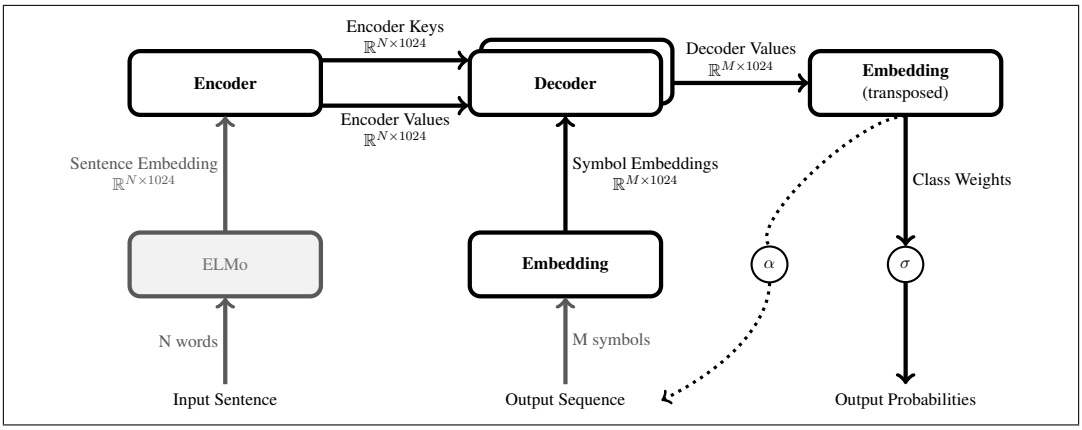

Figure 2: The supertagger's architecture, where σ and α denote the *sigsoftmax* and *argmax* functions respectively, grayed out components indicate non-trainable components and the dotted line depicts the information flow during inference.

the sentence-wide grammar, respectively), we choose to utilize a Transformer network [14] for the task at hand. Transformers rely on self-attention rather than recurrence, a mechanism that allows the network to selectively shift its focus over non-contiguous spans of long, high-dimensional sequences depending on the current context.

Architecture Our model consists of an encoder stack, which is comprised of a frozen (i.e. pretrained) ELMo [13, 3] followed by a single-layer Transformer encoder. As a whole, the encoder creates contextually informed, 1024-dimensional word representations; the pretrained ELMo's output have been shown to achieve high performance in parse-related tasks, and the trainable Transformer encoder accounts for domain adaptation. The model also utilizes a 2-layer Transformer decoder, which processes the atomic symbols produced by the network (up to the current point) along with the encoder's representations. The atomic symbols are vectorized using an embedding layer (again 1024-dimensional), the transpose of which is reused to convert symbolic vectors back into class weights, from which the final output probabilities are obtained via application of the sigsoftmax function [6]. A high-level view of the overall architecture in shown in Figure 2.

3.3 Results

The network was trained on 80% of the type-annotated sentences, validated on half of the remaining sentences, and tested on the remaining 10%. For training, we employed the adaptive scheme proposed by Vaswani et al. [14], with slightly stricter regularization. In the testing phase, the network achieves a competitive overall accuracy, as well as remarkable

generalization potential over rare types and types missing entirely from the training set. Table 2 presents our results across different bins of training set occurrence counts: unseen (0 occurrences), rare (1-10), average (10-100) and high-frequency types ($>$ 100 occurrences).

Type Accuracy (%)				
Overall	Unseen	Rare	Average	High
88.05	19.2	45.68	65.62	89.93

Table 2: Model performance with respect to different bins of type frequencies.

4 Parsing Framework

Considering that a proof is a series of rule applications, we may obtain a proof-faithful parser by simply modeling the grammar's logical rules, namely implication elimination and implication introduction.

With the above in mind, we draw an abstract picture of the main component of our parsing process in Algorithm 1.

Algorithm 1 Parse Step

▷ Performs a backward step of the proof reconstruction.

```
1: procedure ParseStep(Premises)
2:     Goal ← InferGoal(Premises)
3:     while CanIntroduce() do
4:         (premises,Goal) ← ApplyIntro(premises, Goal)
5:     end while
6:     (PremisesLeft, PremisesRight) ← ApplyElim(Premises, Goal)
7:     return (PremisesLeft, PremisesRight)
8: end procedure
```

Concretely, the procedure defined above may be decomposed into several subroutines. First, given the types at the left-hand side of a judgement as assumptions, we may obtain the conclusion, i.e. the proof's goal type at the current step. Barring edge cases, this process is deterministic in virtue of the *count invariance* property [1] of derivable multiplicative sequents that requires balanced numbers of input and output occurrences of atomic subtypes. The next step involves determining whether an introduction rule may be applied. Generally, any time the goal type is a complex type, we can safely replace it by its result (i.e. the type on the right side of the main implication connective) while adding its argument

(i.e. the type on the left side of the main connective) onto the list of premises, a functionality implemented by the APPLYINTRO routine. This is not the case, however, if the main connective carries a name corresponding to a modifier label, or the argument to be introduced was just eliminated at the immediately prior step of the backwards proof search (thus avoiding infinite loops); CANINTRODUCE keeps track of these two conditions and informs the parser accordingly. Finally, given that our complex types are the signatures of binary functors, elimination may be treated as the splitting of a sequence (PREMISES) into two disjoint, possibly non-contiguous, subsequences (PREMISESLEFT and PREMISESRIGHT), where the elements of the first together form the argument that is to be consumed by the function formed by the elements of the second.

In practical terms, we first provide our system with a phrase (the full sequence of premises), out of which the phrasal type is derived. After zero or more introduction applications, the phrase is split into two disjoint (possibly non-contiguous) sequences of premises. If any of the resulting sequences has an length of one, it corresponds to an axiom leaf; otherwise, it is a valid input candidate upon which the same process may start anew. This iteration progressively yields a binary branching tree, that is in one-to-one correspondence with the underlying proof constructed bottom-up (with introduction rules telescoped).

4.1 Elimination is Binary Sequence Classification

At this point, recall that our grammar specification assumes associativity and commutativity as universally holding, a design choice that limits the type system's complexity and enhances its learnability. The non-directionality of $\rightarrow$ means that the splitting done by APPLYELIM is not deterministic. In practice, the type assignments made by the supertagger (even if fully correct) may admit more proofs (or parses) than linguistically desired. To constrain the search space over parses to just those that are linguistically plausible but also constitute valid proofs, our parser needs to resolve ambiguities by incorporating both lexical and type-level information. Hence, rather than treating PREMISES as a sequence of types, we instead expand it to sequence of pairs of words and types. This allows us to distinguish between elements that share the same type but are anchored to different lexical items — an addition that, together with our dependency-decorated types, allows for a preferential treatment of particular words with respect to certain dependency roles under different contexts.

With this in mind, we choose to model APPLYELIM as a neural function. Under its current specification, the function's task is to split a sequence into two disjoint subsequences; or equivallently, to assign a binary label for each item within the sequence. Binary sequence classification is an established problem in machine learning literature, which points to the direct applicability of standard recurrent architectures. Our network is a standard variant of such an architecture; we allow a bidirectional, two-layer deep Gated Recurrent Unit (GRU)[4] to iterate over the vectorial representations of the input sequence. A linear

transformation then converts the high-dimensional vectors of the recurrent unit onto a class weight vector in $\mathbb{R}^2$, which are converted onto class probabilities by the softmax function.

To create vectorial representations of our sequence elements, we first apply the ELMo used by the supertagger onto each word of a sentence. A word's vector $\overrightarrow{w}$ is then given by ELMo(word, context), where context refers to the initial sentence (prior to any eliminations). By using a contextual embedding that is informed by the full sentence and letting it persist unchanged throughout the proof, we can provide the parser with an implicit notion of the proof's "past" while drastically reducing the computational costs associated with calculating the word vectors at each step. Next, we convey the type-level information by associating each unique type with a vector. Recalling that complex types (in Polish notation) are sequences of atomic types and binary connectives, for which we already have embeddings as produced by the supertagger, we construct complex type embeddings by iterating another GRU over the vector sequences that correspond to each complex type. A word-type pair's vector is then simply the concatenation of its word and type vectors. In cases where an element participating in an elimination is not lexically grounded (i.e. types generated by prior introduction rules), we simply set its word vector to zeros. Finally, in order to inform the network of the conclusion type (which we have already derived), we further concatenate its vector onto each word-type pair, essentially converting the recurrence into a function that is parametric with respect to the goal type.

A visual presentation of the network is shown in Figure 3.

Training To train the network, we precompute the contextualized vectors for each word participating in an elimination. Since the word vectors do not change within any single proof, we may then treat each elimination as an independent data point, allowing multiple eliminations (possibly from different sentences) to be processed in parallel. This gives us the ability to avoid complex data structure manipulation during training time, abolishing the need for CPU instructions that insert computational overhead. In effect, the neural component of the parser is based solely on highly optimized tensor operations and requires no more than a couple of minutes to train, despite its high expressivity. This marks a significant improvement over modern parsing architectures, which commonly involve a stack containing partial derivations which is continuously written to and read from, in both the temporal axis (the parse steps) and the sentential axis (the neural iteration over the words).

Inference The inference process is identical to training, except for the fact that the input is no longer an independent sample, but rather the production of a previous application of the network (or, the initial phrase). End-to-end inference has quadratic complexity with respect to the input length (linear number of eliminations, linear iteration complexity per elimination).

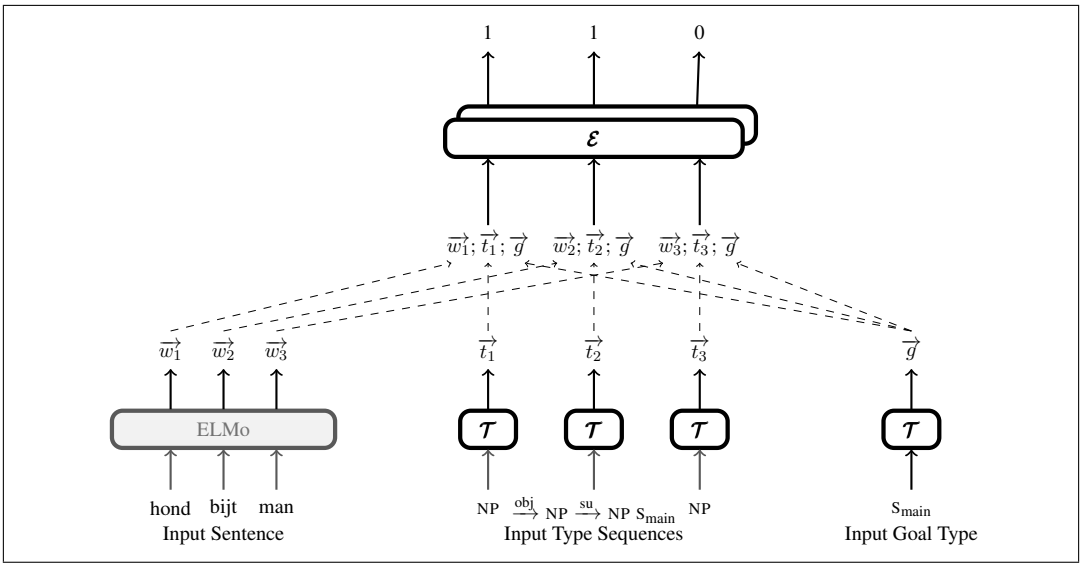

Figure 3: The APPLYELIM neural architecture, where $\mathcal{T}$ refers to the type-level GRU, and $\mathcal{E}$ to the premise-level GRU. Vector concatenation is denoted by ;. For the example word input "hond bijt man", "man" is the element to be eliminated, therefore getting the label 0.

4.2 Results

We run preliminary experiments on a subset of our data to evaluate the framework's potential. We limit our experiments to sentences involving types of at most order 2 and no conjunctions (more on that in Section 5). The resulting dataset counts 33000 sentences (approximately half of the original), out of which 340000 instaces of eliminations are generated. We train on the first 80% of the sentences, and report results on the remaining 20%. In order to assess the parser in isolation, we use the gold extracted types rather than the types assigned by the supertagger.

Implementation Details We use a 1-layer unidirectional GRU with an input and hidden dimensionality of 1024 as our type embedder. For the premise-level GRU, we set its hidden dimensionality to 256, its number of layers to 2 and apply a recurrent dropout of 0.5 for regularization. We train our network using a cross-entropy loss and an AdamW optimizer [7] with a learning rate of 10^{-3} and a weight decay of 10^{-4}.

Ablations We perform a number of ablations to gain an understanding of the relative influence of each extra source of information to the model. Table 3 reports our results.

Model	Full	Full-g	Full-g-t	Full-g-w
Accuracy	**88.6**	85.2	73.82	73.59

Table 3: Percentage of eliminations correctly analyzed by each model, where Full refers to the model described in 4.1, and -g, -t, -w refer to a model where the goal, type and word-level inputs have been removed respectively.

Analysis As all other parsing subroutines are largely deterministic, the neural elimination module is the factor that decides the upper boundary of our system's performance. Evidently, the model achieves competitive accuracy scores despite its simple formulation and implementation. Further, it manages to successfully incorporate all informational sources, as suggested by its increased performance when allowed access to more inputs. When trained only on word or type information, it still manages adequate accuracy scores, attesting to the high quality of both the word vectors as yielded by the pretrained ELMo, and also the rich informational content of the atomic embeddings as produced by the supertagger.

5 Conclusion & Future Work

We have presented an end-to-end parsing framework, which allows the construction of type-logical derivations from input sentences. Our contributions are threefold. First, we have showcased how non-directional implication types may be enriched with dependency label information, integrating part of the parser's overhead into the grammar's type system. Second, we have demonstrated that a reformulation of the supertagging problem significantly increases the generalization potential of tagging architectures, allowing for larger and more fine-grained categorial grammars to be efficiently acquired. Finally, we have presented a novel parsing algorithm capable of resolving the inherent ambiguity of our type system by incorporating a combination of word and type level information. In reducing the neural aspect of parsing to an iterated process of binary sequence classification, our system eliminates the need for coupling a neural network with complex data structures, significantly simplifying the overall architecture. As a result, it is remarkably fast to both train and execute, while still achieving highly competitive results.

Overall, our work proposes a straightforward, pragmatic treatment for languages exhibiting word order freedom. Aside from acting as a competent parser, the system's output (being ILL proofs) facilitate large-scale practical experimentation on data-driven semantic compositionality. Our types have a direct correspondence with simply-typed λ-calculus terms; given a means of translating these terms onto structured sets (e.g. vector spaces) and function spaces, proofs may immediately be interpreted into programs of syntactically-informed meaning assembly and their results practically evaluated.

During our experimentation, a few compromises were made to allow for a faster validation of our methods. These will need to be addressed before a more ambitious expansion is attempted.

Conjunctions To avoid overpopulating the lexicon with various instantiations of conjunction types, we have adopted a meta-notation for each polymorphic instance that describes the union type of all multi-argument functors of the form $X \xrightarrow{cnj} X \ldots \xrightarrow{cnj} X$ as $X^{\star} \xrightarrow{cnj} X$, where the star means 2 or more arguments. A proper logical treatment of coordination polymorphism would require extension of the type logic with facilities for iteration, as in [11]. Not having to specify the concrete number of arguments of a coordinator eases the supertagging phase, but adds a challenge in the parsing phase. The conjunction type now needs to be propagated upwards in the proof unchanged, until the point where no more items are valid arguments for it. This change, if treated carefully, does not have any major implications for the rest of the framework.

Beyond Second Order Our extraction algorithm produces higher-order types that are of maximal order three, whereas our current parser implementation may treat only up to second order types. If at any point during the proof search the goal type is third order, applying an introduction rule yields a lexically ungrounded second order type. Such types may give rise to derivational ambiguity; they are not arguments to be consumed, but rather functors that consume other arguments; if multiple candidate arguments are present, there is no way of uniquely determining which one they are applied to. This is an intrinsic feature of higher order types rather than a defect of the parsing algorithm. Resolving the derivational ambiguity would require either inserting a predefined preference bias to be learned by the network (either implicitly or using an additional component responsible for deciding the formula in focus), or decorating the types in question (for instance using unary modalities) to guide their proof-theoretic behaviour.

Different Networks Aside from considering the above points, there are a few possible directions that we would be interested in further exploring. The architecture of APPLYELIM is one of those. An immediate addition which is very likely to further increase performance would be to incorporate the Transformer encoder (as already trained by the supertagger) into the word vectorization process, further improving our network's representations with no added parameters. More benefits might be reaped if the system is re-purposed as a structured search over entire proof spans (rather than just elimination steps in isolation); such a change would output more than one unique proof outputs, each associated with a different probability score, allowing for multiple syntactic analyses in cases of real (i.e. non-spurious) ambiguity. Further, a comparison to alternative architectures (i.e. encoder-

decoder networks) or established parsing practices (i.e. shift/reduce-based parsers) would shed light on the comparative strengths and weaknesses of our approach.

Evaluation Our parser was trained using gold tags obtained directly from the extraction process. Realistically, it would need to perform even under the presence of partially erroneous tags, as produced by the supertagger. To increase its robustness against wrong tags, the training process could emulate these by confounding the input types in a controlled manner (alternatively, the two could be jointly trained on a shared portion of the dataset). Finally, to estimate concrete high and low bounds for the end-to-end accuracy of proof reconstruction, the supertagger's output would need to be used directly as the parser's input.

References

[1] van Benthem, J.: Language in action. J. Philosophical Logic **20**(3), 225–263 (1991)

[2] Benton, N., Bierman, G., de Paiva, V., Hyland, M.: A term calculus for intuitionistic linear logic. In: Bezem, M., Groote, J.F. (eds.) Typed Lambda Calculi and Applications. pp. 75–90. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)

[3] Che, W., Liu, Y., Wang, Y., Zheng, B., Liu, T.: 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. pp. 55–64. Association for Computational Linguistics, Brussels, Belgium (October 2018), http://www.aclweb.org/anthology/K18-2005

[4] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar (Oct 2014). https://doi.org/10.3115/v1/D14-1179, https://www.aclweb.org/anthology/D14-1179

[5] Dowty, D.: Grammatical relations and Montague grammar. In: Jacobson, P., Pullum, G. (eds.) The nature of syntactic representation, pp. 79–130. Reidel (1982)

[6] Kanai, S., Fujiwara, Y., Yamanaka, Y., Adachi, S.: Sigsoftmax: Reanalysis of the softmax bottleneck. In: Advances in Neural Information Processing Systems. pp. 284–294 (2018)

[7] Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101 (2017)

[8] Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language and Information **5**(3-4), 349–385 (1996)

[9] Moortgat, M., Moot, R.: Using the spoken Dutch corpus for type-logical grammar induction. In: Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02). European Language Resources Association (ELRA) (2002), http://www.lrec-conf.org/proceedings/lrec2002/pdf/209.pdf

[10] Moot, R., Retoré, C.: The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics, vol. 6850. Springer (07 2012). https://doi.org/10.1007/978-3-642-31555-8

[11] Morrill, G., Valentín, O.: Computational coverage of TLG: nonlinearity. CoRR **abs/1706.03033** (2017)

[12] van Noord, G., Schuurman, I., Vandeghinste, V.: Syntactic annotation of large corpora in STEVIN. In: LREC 2006 Proceedings. 5th Edition of the International Conference on Language Resources and Evaluation. European Language Resources Association (ELRA) (01 2006)

[13] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). pp. 2227–2237. Association for Computational Linguistics, New Orleans, Louisiana (Jun 2018). https://doi.org/10.18653/v1/N18-1202, https://www.aclweb.org/anthology/N18-1202

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)

Received \jreceived