Geometry-Aware Supertagging with Heterogeneous Dynamic Convolutions

Konstantinos Kogkalidis[◇, □] and **Michael Moortgat**[□]

[◇]Aalto University
[□]Utrecht Institute of Linguistics OTS, Utrecht University
kokos.kogkalidis@aalto.fi, m.j.moortgat@uu.nl

Abstract

The syntactic categories of categorial grammar formalisms are structured units made of smaller, indivisible primitives, bound together by the underlying grammar's category formation rules. In the trending approach of constructive supertagging, neural models are increasingly made aware of the internal category structure. In turn, this enables them to more reliably predict rare and out-of-vocabulary categories, with significant implications for grammars previously deemed too complex to find practical use. In this work, we revisit constructive supertagging from a graph-theoretic perspective, and propose a framework based on heterogeneous dynamic graph convolutions, aimed at exploiting the distinctive structure of a supertagger's output space. We test our approach on a number of categorial grammar datasets spanning different languages and grammar formalisms, achieving substantial improvements over previous state of the art scores.

1 Introduction

Their close affinity to logics and lambda calculi has made categorial grammars a standard tool of trade for the formally-inclined NLP practitioner. Modern flavors of categorial grammar, despite their (sometimes striking) divergences, share a common architecture. At its core, a categorial grammar is a formal system consisting of two parts. First, there is a *lexicon*, a mapping that assigns to each word a set of *categories*. Categories are quasi-logical formulas recursively built out of atomic categories by means of category forming operations. The inventory of category forming operations at the minimum has the ability to express linguistic function-argument structure. If so desired, the inventory can be extended with extra operations, e.g. to handle syntactic phenomena beyond simple concatenation, or to express additional layers of grammatical information. The second component of the grammar is a small set of *inference rules*, formulated in terms of the category forming operations. The inference rules dictate how categories interact and, through this interaction, how words combine to form larger phrases. Parsing thus becomes a process of deduction comparable (or equatable, depending on the grammar's formal rigor) to program synthesis, providing a clean and elegant syntax-semantics interface.

In the post-neural era, these two components allow differentiable implementations. The fixed lexicon is replaced by *supertagging*, a process that contextually decides on the most appropriate supertags (i.e. categories), whereas the choice of which rules of inference to apply is usually deferred to a parser further down the processing pipeline. The highly lexicalized nature of categorial grammars thus shifts the bulk of the weight of a parse to the supertagging component, as its assignments and their internal make-up inform and guide the parser's decisions.

In this work, we revisit supertagging from a geometric angle. We first note that the supertagger's output space consists of a sequence of trees, which has as of yet found no explicit representational treatment. Capitalizing on this insight, we employ a framework based on heterogeneous dynamic graph convolutions, and show that such an approach can yield substantial improvements in predictive accuracy across categories both frequently and rarely encountered during a supertagger's training phase.

2 Background

The supertagging problem revolves around the design and training of a function tasked with mapping each word (in the context of a sentence) to a category, thus inducing a sequence of categories $\{c_1, \ldots, c_n\}$ from a sentence $\{w_1, \ldots, w_n\}$. Existing supertagging architectures differ in how they implement this mapping, with each implementation choice boiling down to (i) which of the temporal and structural dependencies within and between the input and output are taken into consideration, and (ii) how these dependencies are materialized.

Earlier work would utilize solely occurrence counts from a training corpus to independently map word n-grams to their most likely categories, and then attempt to filter out implausible sequences via rule-constrained probabilistic models (Bangalore and Joshi, 1999). The shift from sparse feature vectors to distributed word representations facilitated integration with neural networks and improved generalization on the mapping domain, extending it to rare and previously unseen words (Lewis and Steedman, 2014). Later, the advent of recurrent neural networks offered a natural means of incorporating temporal structure, widening the input receptive field through contextualized word representations on the one hand (Xu et al., 2015), but also permitting an autoregressive formulation of the output generation, whereby the effect of a category assignment could percolate through the remainder of the output sequence (Vaswani et al., 2016). Regardless of implementation specifics, the discriminative paradigm employed by all above works fails to account for the skewness of the data; exceedingly rare categories are practically impossible to learn, and categories absent from the training data are completely ignored.

As an alternative, the recently emerging constructive paradigm seeks to explore the structure hidden *within* categories. By inspecting their formation rules, Kogkalidis et al. (2019) equates categories to CFG derivations, viewing *each* category as a tiny compositional expression, and a category sequence as the concatenation of their flattened depth-first projections. The goal sequence is now incrementally generated on a symbol-by-symbol basis using a transformer-based seq2seq model; a twist which provides the decoder with the means to construct novel categories on demand, bolstering co-domain generalization. The decoder's global receptive field, however, comes at the heavy price of quadratic memory complexity, which also bodes poorly with the elongated output sequences, leading to a slowed down inference speed. Expanding on the idea, Prange et al. (2021) explicates the categories' tree structure, embedding symbols based on their tree positions and propagating contextualized representations through tree edges, using either residual dense connections or a tree-structured GRU. This adaptation completely eliminates the burden of learning *how* the categorial trees are constructed, instead allowing the model to focus on *what* trees to construct, leading to drastically improved performance. Simultaneously, since the decoder is now token-separable, it permits construction of categories for the entire sentence in parallel, speeding up inference and reducing the network's memory footprint. In the process, however, it loses the ability to model interactions between autoregressed nodes belonging to different trees, morally reducing the task once more to sequence classification (albeit now with a dynamic classifier).

Despite their common goal of accounting for syntactic categories in the zipfian tail, there are tension points between the above two approaches. In flattening categories and concatenating them together, the first breaks the input-to-output alignment and obfuscates the categorial tree structure. In opting for a tree-wise bottom-up decoding, the second forgets about meaningful *inter*-tree output-to-output dependencies. In this paper, we seek to resolve these tension points with a novel, unified and grammar-agnostic supertagging framework based on heterogeneous dynamic graph convolutions. Our architecture combines the merits of explicit tree structures, strong autoregressive properties, near-constant decoding time, and a memory complexity that scales with the input, boasting high performance across the full span of the frequency spectrum and surpassing previously established benchmarks on all datasets considered.

3 Methodology

3.1 Breadth-First Parallel Decoding

Despite seeming at odds, both architectures described fall victim to the same trap of conflating problem-specific structural biases and general purpose decoding orders: one forgets about tree structure in opting for a sequential decoding, whereas the other does the exact opposite, forgetting about sequential structure in opting for a tree-like decoding. We note first that the target output is (a batch of) neither sequences nor trees, but rather *sequences of trees*. Having done that, our task is of a purely technical nature: we simply need to come up with the spatiotemporal dependencies that abide by *both* structural axes, and then a neural architecture that can accommodate them.

Prange et al. (2021) make a compelling case for depth-parallel decoding, given that it's incredibly fast (i.e., not temporally bottlenecked by left-to-right sequential dependencies) but also structurally elegant (trees are only built when/if licensed by non-terminal nodes, ensuring structural correctness virtually for free). Sticking with depth-parallel decoding means necessarily foregoing some autoregressive interactions: we certainly cannot look to the future (i.e., tree nodes located deeper than the current level, since these should depend on the decision we are about to make), but neither to the present (i.e., tree nodes residing in the current level, since these will be all decided simultaneously). This still leaves some leeway as to what could constitute the prediction context. The maximalist position we adopt here is nothing less than the entire past, i.e. *all* the nodes we have so far decoded. Crucially, this extends beyond the ancestry-bound "vertical interactions" of a tree unfolding function implemented *à la* treeRNN, allowing "diagonal" interactions between autoregressed nodes living in different trees.

Such exotic interactions do not follow the inductive biases of any run-of-the-mill architecture, forcing us to turn our attention to structure-aware dynamic convolutions. To make the architecture conducive to learning while keeping its memory footprint in check, we repurpose the encoder's word vectors from initial seeds to recurrent state-tracking vectors that arbitrate the decoding process across both *sequence length* and *tree depth*, respecting the "regularly irregular" structure of the output space. In high level terms, the process can be summarized as an iteration of three alternating stages of message passing rounds.

1. Lexical state vectors are initialized by some external encoder.
2. An empty fringe consisting of blank nodes is instantiated, one such node per word, rooting the corresponding categorial trees.
3. Until a fix-point is reached (there is no longer any fringe):
   - (i) **Node Prediction** States project class weights to their respective fringe nodes in a one-to-many fashion. Depending on the arity of the decoded symbols, a next fringe of unfilled nodes is constructed at the appropriate positions; e.g., binary operators expand the fringe by introducing two new blank nodes located directly above them.
   - (ii) **Autoregressive Feedback** Each state vector receives feedback in a many-to-one fashion, originating from the lexically-aligned nodes just decoded (i.e., the fringe at the previous time step). This way, state vectors are iteratively updated and progressively aggregate information from the tree as it is being dynamically constructed.
   - (iii) **Sequential Feedback** The updated state vectors emit and receive messages to one another in a many-to-many fashion, allowing states to be informed by the decoding progress of their neighbors.

For a visual rendition, refer to Appendix A.

3.2 Architecture

We now move on to detail the individual blocks that together make up the network's pipeline.

3.2.1 Node Embeddings

State vectors are temporally dynamic; they are initially supplied by an external encoder, and are then updated through a repeated sequence of three message passing rounds, described in the next subsections. Tree nodes, on the other hand, are not subject to temporal updates, but instead become dynamically "revealed" by the decoding process. Their representations are computed on the basis of (i) their primitive symbol and (ii) their position within a tree.

Primitive symbol embeddings are obtained from a standard embedding table $W_e : \mathcal{S} \rightarrow \mathbb{R}^{d_n}$ that contains a distinct vector for each symbol in the set of primitives $\mathcal{S}$. When it comes to embedding positions, we are presented with a number of options. It would be straightforward to fix a vocabulary of positions, and learn a distinct vector for each. Such an approach would however lack elegance, as it would impose an ad-hoc bound to the shape of trees that can be encoded (contradicting the constructive paradigm), while also failing to account for the compositional nature of trees. We thus opt for a path-based approach, inspired by and improving upon the idea of Shiv and Quirk (2019). We note first that *paths* over binary branching trees form a semi-group, i.e. they consist of two primitives (namely a left and a right path), and an associative non-commutative binary operator that binds two paths together into a single new one. The archetypical example of a semigroup is matrix multiplication; we therefore instantiate a tensor $P \in \mathbb{R}^{2 \times n_d \times n_d}$ encoding each of the two path primitives as a linear map over symbol embeddings. From the above we can derive a function p that converts positions to linear maps, by performing consecutive matrix multiplications of the primitive weights, as indexed by the binary word of a node's position; e.g. the linear map corresponding to position $12_{10} = 1100_2$ would be $p(12) = P_0 P_0 P_1 P_1 \in \mathbb{R}^{d_n \times d_n}$. We flatten the final map by evaluating it against an initial seed vector ρ_0, corresponding to the tree root.[1] To stabilize training and avoid vanishing or exploding weights, we model paths as *unitary* transformations by parameterizing the two matrices of P to orthogonality using the exponentiation trick on skew-symmetric bases (Bader et al., 2019; Lezcano Casado, 2019).

The representation $n_{i,k}$ of a tree node $\sigma \in \mathcal{S}$ occupying position k in tree i will then be given as the element-wise product of its tree-positional and content embeddings:

$$n_{i,k} = p(k)(\rho_0) \odot (W_e(\sigma)) \in \mathbb{R}^{d_n}$$

The embedder is then essentially an instantiation of a binary branching unitary RNN (Arjovsky et al., 2016), where the choice of which hidden-to-hidden map to follow at each step depends on the node's position relative to its ancestor.[2] Since paths are shared across trees, their representations are in practice efficiently computed once per batch for each unique tree position during training, and stored as fixed embeddings during inference.

3.2.2 Node Prediction

Assuming at step τ a sequence of globally contextualized states h^τ, we need to use each element h_i^τ to obtain class weights for all of the node neighborhood $\mathcal{N}_{i,\tau}$ consisting of all nodes (if any) of tree i that lie at depth τ. We start by down-projecting the state vector into the node's dimensionality using a linear map W_n. The resulting feature vectors are indistinguishable between all nodes of the same tree – to tell them apart (and obtain a unique prediction for each), we gate the feature vectors against each node's positional embedding. From the latter, we obtain class weights by matrix multiplying them against the transpose of the symbol embedding table (Press and Wolf, 2017):

$$\text{weights}_{i,k} = (p(k)(\rho_0) \odot W_n h_i^\tau)\, W_e^\top$$

[1]In practice, paths are efficiently computed once per batch for each unique tree position during training, and stored as fixed embeddings during inference.

[2]Concurrently, Bernardy and Lappin (2022) follow a similar approach in teaching a unitary RNN to recognize Dyck words, and find the unitary representations learned to respect the compositional properties of the task. Here we go the other way around, using the unitary recurrence exactly because we expect them to respect the compositional properties of the task.

The above weights are converted into a probability distribution over the alphabet symbols $\mathcal{S}$ by application of the $\mathrm{softmax}$ function.

3.2.3 Autoregressive Feedback

We update states with information from the last decoded nodes using a heterogeneous message-passing scheme based on graph attention networks (Veličković et al., 2018; Brody et al., 2021). First, we use a bottleneck layer W_b to down-project the state vector into the nodes' dimensionality. For each position i and corresponding state h_i^τ, we compute a self-loop score:

$$\tilde{\alpha}_{i,\circlearrowleft,\tau} = w_a \cdot (W_b(h_i^\tau) \mid\mid \mathbf{0})$$

where $w_a \in \mathbb{R}^{2d_n}$ a dot-product weight and $\mathbf{0}$ a d_n-dimensional zero vector. Then we use the (now decoded) neighborhood $\mathcal{N}_{i,\tau}$ to generate a heterogeneous attention score for each node $n_{i,k} \in \mathcal{N}_{i,\tau}$:

$$\tilde{\alpha}_{i,k,\tau} = w_a \cdot (h_i^\tau \mid\mid n_{i,k})$$

Scores are passed through a leaky rectifier non-linearity before being normalized to attention coefficients α. These are used as weighting factors that scale the self-loop and input messages, the latter upscaled by a linear map W_m:

$$\tilde{h}_i^\tau = \sum_{n_{i,k} \in \mathcal{N}_{i,\tau}} \alpha_{i,k,\tau} W_m n_{i,k} + \alpha_{i,\circlearrowleft,\tau} h_i^\tau$$

This can also be seen as a dynamic residual connection – $\alpha_{i,\circlearrowleft,\tau}$ acts as a gate that decides how open the state's representation should be to node feedback (or conversely, how strongly it should retain its current values). States receiving no node feedback (i.e. states that have completed decoding one or more time steps ago) are thus protected from updates, preserving their content. In practice, attention coefficients and message vectors are computed for multiple attention heads independently, but these are omitted from the above equations to avoid cluttering the notation.

3.2.4 Sequential Feedback

At the end of the node feedback stage, we are left with a sequence of locally contextualized states $\tilde{h}_i^\tau$. The sequential structure can be seen as a fully connected directed graph, nodes being states (words) and edges tabulated as the square matrix $\mathcal{E}$, with entry $\mathcal{E}_{i,j}$ containing the relative distance between words i and j. We embed these distances into the encoder's vector space using an embedding table

$W_r \in \mathbb{R}^{2\kappa \times d_w}$, where κ the maximum allowed distance, a hyper-parameter. Edges escaping the maximum distance threshold are truncated rather than clipped, in order to preserve memory and facilitate training, leading to a natural segmentation of the sentence into (overlapping) chunks. Following standard practices, we project states into query, key and value vectors, and compute the attention scores between words i and j using relative-position weighted attention (Shaw et al., 2018):

$$\tilde{a}_{i,j} = d_w^{-1/2} \left(W_q \tilde{h}_i^\tau \odot W_r \mathcal{E}_{i,j} \right) \cdot W_k \tilde{h}_j^\tau$$

From the normalized attention scores we obtain a new set of aggregated messages:

$$m'_{i,t} = \sum_{j \in \{0..s\}} \frac{\exp(\tilde{a}_{i,j}) W_v \tilde{h}_j^\tau}{\sum_{k \in \{0..s\}} \exp(\tilde{a}_{i,k})}$$

Same as before, queries, keys, values, edge embeddings and attention coefficients are distributed over many heads. Aggregated messages are passed through a swish-gated feed-forward layer (Dauphin et al., 2017; Shazeer, 2020) to yield the next sequence of state vectors:

$$h_i^{\tau+1} = W_3 \left(\text{swish}_1(W_1 m'_{i,\tau}) \odot W_2 m'_{i,\tau} \right)$$

where $W_{1,2}$ are linear maps from the encoder's dimensionality to an intermediate dimensionality, and vice versa for W_3.

3.2.5 Putting Things Together

We compose the previously detailed components into a single layer, which acts a sequence-wide, recurrent-in-depth decoder. We insert skip connections between the input and output of the message-passing and feed-forward layers (He et al., 2016), and subsequently normalize each using root mean square normalization (Zhang and Sennrich, 2019).

4 Experiments

We employ our supertagging architecture in a range of diverse categorial grammar datasets spanning different languages and underlying grammar formalisms. In all our experiments, we bind our model to a monolingual BERT-style language model used as an external encoder, fine-tuned during training (Devlin et al., 2018). In order to homogenize the tokenization between the one directed by each dataset and the one required by the encoder, we make use of a simple localized attention aggregation scheme. The subword tokens together comprising a single word are independently projected to scalar values through a shallow feed-forward layer. Scalar values are softmaxed within their local group to yield attention coefficients over their respective BERT vectors, which are then summed together, in a process reminiscent of a cluster-wide attentive pooling (Li et al., 2016). In cases of data-level tokenization treating multiple words as a single unit (i.e. assigning one type to what BERT perceives as many words), we mark all words following the first with a special `[MWU]` token, signifying they need to be merged to the left. This effectively adds an extra output symbol to the decoder, which is now forced to do double duty as a sequence chunker. To avoid sequence misalignments and metric shifts during evaluation, we follow the merges dictated by the ground truth labels, and consider the decoder's output as correct only if all participating predictions match, assuming no implicit chunking oracles.

4.1 Datasets

We conduct experiments on the two variants of the English CCGBank, the French TLGbank and the Dutch Æthel proofbank. A high-level overview of the datasets is presented in Table 1, and short descriptions are provided in the following paragraphs. We refer the reader to the corresponding literature for a more detailed exposition.

	CCGbank		***TLGbank***	***Æthel***
	original	*rebank*		
Primitives	37	40	27	81
Zeroary	35	38	19	31
Binary	2	2	8	50
Categories	1323	1619	851	5762
in train	1286	1575	803	5146
depth avg.	1.94	1.96	1.99	1.82
depth max.	6	6	7	35
Test Sentences	2407	2407	1571	5770
length avg.	23.00	24.27	27.58	16.52
Test Tokens	55371	56395	44302	95331
Frequent (100+)	54825	55690	43289	91503
Uncommon (10-99)	442	563	833	2639
Rare (1-9)	75	107	149	826
Unseen (OOV)	22	27	31	363

Table 1: Bird's eye view of datasets employed and relevant statistics. Test tokens are binned according to their corresponding categories' occurrence count in the respective dataset's training set. Token counts are measured before pre-processing. Unique primitives for the type-logical datasets are counted after binarization.

CCGBank The English CCGbank (original) (Hockenmaier and Steedman, 2007) and its refined version (rebank) (Honnibal et al., 2010) are resources of Combinatory Categorial Grammar (CCG) derivations obtained from the Penn Treebank (Taylor et al., 2003). CCG (Steedman and Baldridge, 2011) builds lexical categories with the aid of two binary slash operators, capturing forward and backward function application. Some additional rules lent from combinatory logic (Curry et al., 1958) permit constrained forms of type raising and function composition, allowing categories to remain relatively short and uncomplicated while keeping parsing complexity in check. The key difference between the two versions lies in their tokenization and the plurality of categories assigned, the latter containing more assignments and a more fine-grained set of syntactic primitives, which in turn make it a slightly more challenging evaluation benchmark.

French TLGbank The French type-logical treebank (Moot, 2015) is a collection of proofs extracted from the French treebank (Abeillé et al., 2003). The theory underlying the resource is that of Multi-Modal Typelogical Grammars (Moortgat, 1996); annotations are deliberately made compatible with Displacement Calculus (Morrill et al., 2011) and First-Order Linear Logic (Moot and Piazza, 2001) at the cost of a small increase in lexical sparsity. In short, the vocabulary of operators is extended with two modalities that find use in licensing or restricting the applicability of rules related to non-local syntactic phenomena. To adapt their representation to our framework, we cast unary operators into pseudo-binaries by inserting an artificial terminal tree in a fixed slot within them. Due to the absence of predetermined train/dev/test splits, we randomize them with a fixed seed at a 80/10/10 ratio and keep them constant between repetitions.

Æthel Our last experimental test bed is Æthel (Kogkalidis et al., 2020a), a dataset of type-logical proofs for written Dutch sentences, automatically extracted from the Lassy-Small corpus (Noord et al., 2013). Æthel is geared towards *semantic* parsing, which means categories employ linear implication $\multimap$ as their single binary operator. An additional layer of dependency information is realized via unary modalities, now lifted to *classes* of operators distinguishing complement and adjunct roles. The grammar assigns concrete instances of polymorphic coordinator types, as a result containing more and sparser categories (some of which distinctively tall); considering also its larger vocabulary of primitives, it makes for a good stress test for our approach. We experiment with the latest available version of the dataset (version `1.0.0a5` at the time of writing). Same as before, we impose a regular tree structure, this time by merging adjunct (resp. complement) markers with the subsequent (resp. preceding) binary operator, which makes for an unambiguous and invertible representational translation.

4.2 Implementation

We implement our model using PyTorch Geometric (Fey and Lenssen, 2019), which provides a high-level interface to efficient low-level protocols, facilitating fast and pad-free graph manipulations. We share a single hyper-parameter setup across all experiments, obtained after a minimal logarithmic search over sensible initial values. Specifically, we set the node dimensionality d_n to 128 with 4 heterogeneous attention heads and the state dimensionality d_w to 768 with 8 homogeneous attention heads. We train using AdamW (Loshchilov and Hutter, 2018) with a batch size of 16, weight decay of 10^{-2}, and a learning rate of 10^{-4}, scaled by a linear warmup and cosine decay schedule over 25 epochs. During training we provide strict teacher forcing and apply feature and edge dropout at 20% chance. Our loss signal is derived as the label-smoothed negative log-likelihood between the network's prediction and the ground truth label (Müller et al., 2019). We procure pretrained base-sized BERT variants from the transformers library (Wolf et al., 2020): RoBERTa for English (Liu et al., 2019), BERTje for Dutch (de Vries et al., 2019) and CamemBERT for French (Martin et al., 2020), which we fine-tune during training, scaling their learning rate by 10% compared to the decoder.

4.3 Results

We perform model selection on the basis of validation accuracy, and gather the corresponding test scores according to the frequency bins of Table 1. Table 2 presents our results compared to relevant published literature. Evidently, our model surpasses established benchmarks in terms of overall accuracy, matching or surpassing the performance of both traditional supertaggers on common categories and constructive ones on the tail end of the frequency distribution.

We observe that the relative gains appear to scale with respect to the task's complexity. In the original version of the CCGbank, our model is only slightly

model	accuracy (%)				
	overall	frequent	uncommon	rare	unseen
CCG (original)					
Symbol Sequential LSTM /w n-gram oracles (Liu et al., 2021)	95.99	96.40	65.83	8.65[!]	
Cross-View Training (Clark et al., 2018)	96.10	–	–	–	n/a
Recursive Tree Addressing (Prange et al., 2021)	96.09	96.44	68.10	**37.40**	**3.03**
BERT Token Classification (Prange et al., 2021)	96.22	**96.58**	70.29	23.17	n/a
Attentive Convolutions (Tian et al., 2020)	**96.25**	**96.64**	71.04	n/a	n/a
Heterogeneous Dynamic Convolutions (this work)	**96.29**$\pm$0.04	**96.61**$\pm$0.04	**72.06**$\pm$0.72	34.45$\pm$1.58	**4.55**$\pm$2.87
CCG (rebank)					
Symbol Sequential Transformer[†] (Kogkalidis et al., 2019)	90.68	91.10	63.65	34.58	**7.41**
TreeGRU (Prange et al., 2021)	94.62	95.10	64.24	25.55	2.47
Recursive Tree Addressing (Prange et al., 2021)	94.70	95.11	68.86	**36.76**	4.94
Token Classification (Prange et al., 2021)	94.83	95.27	68.68	23.99	n/a
Heterogeneous Dynamic Convolutions (this work)	**95.07**$\pm$0.04	**95.45**$\pm$0.04	**71.40**$\pm$1.15	**37.19**$\pm$1.81	3.70$\pm$0.00
French TLGbank					
ELMo & LSTM Classification (Moot, 2019)	93.20	95.10	75.19	25.85	n/a
BERT Token Classification[‡]	**95.93**	**96.44**	81.39	47.45	n/a
Heterogeneous Dynamic Convolutions (this work)	**95.92**$\pm$0.01	96.40$\pm$0.01	**81.48**$\pm$0.97	**55.37**$\pm$1.00	**7.26**$\pm$2.67
Æthel					
Symbol Sequential Transformer[⋆] (Kogkalidis et al., 2020b)	83.67	84.55	64.70	50.58	**24.55**
BERT Token Classification[‡]	93.52	**94.83**	71.85	38.06	n/a
Heterogeneous Dynamic Convolutions (this work)	**94.08**$\pm$0.02	**95.16**$\pm$0.01	**75.55**$\pm$0.02	**58.15**$\pm$0.01	18.37$\pm$2.73

[!] Accuracy over both bins, with a frequency-truncated training set (authors claim no difference when using the full set).
[†] Numbers from Prange et al. (2021).
[‡] Our replication.
[⋆] Model trained and evaluated on an older dataset version and tree sequences spanning less than 140 nodes in total.

Table 2: Model performance across datasets and compared to recent studies. Numbers are taken from the papers cited unless otherwise noted. For our model, we report averages and standard deviations over 6 runs. Bold face fonts indicate (within standard deviation of) highest performance.

superior to the next best performing model (in turn only marginally superior to the token-based classification baseline), whereas in the rebank version the absolute difference is one order of magnitude wider. The effect is even further pronounced for the harder type-logical datasets, which are characterized by a longer tail, leading to performance comparable to CCGbank's for the French TLGbank (despite it being significantly smaller and sparser), and a 10% absolute performance leap for Æthel (despite its unusually tall and complex types). We attribute this to increased returns from performance in the rare and uncommon bins; there is a synergistic effect between the larger population of these bins pronouncing even minor improvements, and acquisition of rarer categories apparently benefiting from the plurality of their respective bins in a self-regularizing manner. Put simply, learning sparse categories is *easier* and *matters more* for grammars containing many rare categories.

Finally, to investigate the relative impact of each network component, we conduct an ablation study where message passing components are removed from their network in their entirety. Removing the state feedback component collapses the network into a token-wise separable recurrence, akin to a graph-featured RNN without a hidden-to-hidden affine map. Removing the node feedback component turns the network into a Universal Transformer (Dehghani et al., 2018) composed with a dynamically adaptive classification head. Removing both is equatable to a 1-to-many contextualized token classification that is structurally unfolded in depth. Our results, presented in Table 3, verify first a positive contribution from both components, indicating the importance of both information sharing axes. In three out of the four datasets, the relative gains of incorporating state feedback outweigh those of node feedback, and are most pronounced in the case of Æthel, likely due to its positionally agnostic types. With the exception of CCGrebank, relinquishing both kinds of feedback largely underperforms having either one, experimentally affirming their compatibility.

	-sf	-nf	-sf-nf
CCG *(original)*	-0.05	-0.01	-0.08
CCG *(rebank)*	-0.12	-0.04	-0.07
French TLGbank	-0.13	-0.14	-0.23
Æthel	-0.24	-0.12	-0.37

Table 3: Absolute difference in overall accuracy when removing the state and node feedback components (averages of 3 repetitions).

5 Related Work

Our work bears semblance and owes credit to various contemporary lines of work. From the architectural angle, we perceive our work as an application-specific offspring of weight-tied architectures, dynamic graph convolutions and structure-aware self-attention networks. The depth recurrence of our decoder is inspired by weight-tied architectures (Dehghani et al., 2018; Bai et al., 2019) and their graph-oriented variants (Li et al., 2016), which model neural computation as the fix-point iteration of a single layer against a structured input, thus allowing for a dynamically adaptive computation "depth" – albeit with a constant parameter count. Analogously to structure-aware self-attention networks (Zhu et al., 2019; Cai and Lam, 2020) and graph attentive networks (Veličković et al., 2018; Yun et al., 2019; Ying et al., 2021; Brody et al., 2021), our decoder employs standard query/key and fully-connected attention mechanisms injected with structurally biased representations, either at the edge or at the node level. Finally, akin to dynamic graph approaches (Liao et al., 2019; Pareja et al., 2020), our decoder forms a closed loop system that autoregressively generates its own input, in the process becoming exposed to subgraph structures that drastically differ between time steps.

From the application angle, our proposal is a refinement of and a continuation to recent advances in categorial grammar supertagging. Similar to the transition from words to subword units (Sennrich et al., 2016), constructive supertaggers seek to bolster generalization by disassembling syntactic categories into smaller indivisible units, thereby incorporating structure at a finer granularity scale. The original approach of Kogkalidis et al. (2019) employed seq2seq models to directly translate an input text to a flattened projection of a categorial sequence, demonstrating that the correct prediction of categories unseen during training is indeed feasible. Prange et al. (2021) improved upon the process through the explicit accounting of the tree structure embedded within categorial types, while Liu et al. (2021) explored the orthogonal approach of employing a transition-based "parser" over individual categories. Outside the constructive paradigm, Tian et al. (2020) employed graph convolutions over sentential edges built from static, lexicon-based preferences. Our approach is a bridge between prior works; our modeling choice of structure-aware graph convolutions boasts the merits of ex+plicit sentential and tree-structured edges, a structurally constrained, valid-by-construction output space, favorable memory and time complexities, partial autoregressive context flows, end-to-end differentiability with no vocabulary requirements, and minimal rule-based structure manipulation.

6 Conclusion

We have proposed a novel supertagging methodology, where both the linear order of the output sequence and the tree-like structure of its elements is made explicit. To represent the different information sources (sentential word order, subword contextualized vectors, tree-sequence order and intra-tree edges) and their disparate sizes and scales, we turned to heterogeneous graph attention networks. To capture the autoregressive dependencies between different trees, we formulated the task as a dynamic graph completion process, aligning each subsequent temporal step with a higher order tree node neighborhood and predicting them in parallel across the entire sequence. We tested our methodology on four different datasets spanning three languages and as many grammar formalisms, establishing new state of the art scores in the process. Through our ablation studies, we showed the importance of incorporating both *intra*- and *inter*-tree context flows, to which we attribute our system's performance.

Other than architectural adjustment and optimizations, several interesting ideas present themselves as promising research avenues. First, it is worthwhile to consider adaptations of our framework to either allow an efficient integration of more "exotic" context pathways, e.g. sibling node interactions, or alter the graph's decoding order altogether. On a related note, for formalisms faithful to the linear logic roots of categorial grammars, it seems reasonable to anticipate that the goal graph can be compactified by collapsing primitive nodes of opposite polarity according to their interactions, unifying the tasks of supertagging and parsing with a single end-to-end framework.

Practice aside, our results pose further evidence that lexical sparsity, historically deemed the categorial grammar's curse, might well just require a change of perspective to tame and deploy as the answer to the very problem it poses.

Limitations

Despite its objective success, our methodology is not without limitations. Most importantly, our model trades inference speed for an incompatibility with local greedy algorithms like beam search. Put plainly, obtaining more than the "best" category assignment per word is not straightforward, which can potentially negatively impact the downstream parser's coverage. A possible solution would involve branching across multiple tree-slices (i.e. sequences of partial assignments) rather than single predictions, but efficiently computing scores and comparing between complex structures is uncharted territory and not trivial to implement. Note, however, that the issue is not unique to our system but common to all decoders that perform multiple assignments concurrently.

Parallel or not, all autoregressive decoders assume an order on their output: the standard left-to-right order (which makes sense for text) has become the de facto choice for most applications. The order we have chosen to employ here is structurally faithful to our output, but is neither the only one, nor necessarily the most natural one. In that sense, the entanglement between structural bias (i.e. from the graph operations and representations) and decoding priority (i.e. the order in which trees become revealed) is a practical decision rather than a deep one – a better operationalization could for instance employ an insertion-style operation on the graph-structured output to yield an "easy-first" geometric tagger. We await further developments and community insights on that front.

Finally, the system carries the standard risks of any NLP architecture reliant on machine learning, namely linguistic biases inherited from the unsupervised pretraining of the incorporated language models, and annotation biases derived from the supervised training over human-labeled data.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In *Treebanks*, pages 165–187. Springer.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2016. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR.

Philipp Bader, Sergio Blanes, and Fernando Casas. 2019. Computing the matrix exponential with an optimized Taylor polynomial approximation. *Mathematics*, 7(12):1174.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32.

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

Jean-Philippe Bernardy and Shalom Lappin. 2022. Assessing the unitary rnn as an end-to-end compositional model of syntax. *arXiv preprint arXiv:2208.05719*.

Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Haskell Brooks Curry, Robert Feys, William Craig, J Roger Hindley, and Jonathan P Seldin. 1958. *Combinatory Logic*, volume 1. North-Holland Amsterdam.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941.

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:1912.09582*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Matthew Honnibal, James R Curran, and Johan Bos. 2010. Rebanking CCGbank for improved np interpretation. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 207–215.

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, Florence, Italy. Association for Computational Linguistics.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020a. ÆTHEL: Automatically extracted typelogical derivations for Dutch. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5257–5266, Marseille, France. European Language Resources Association.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020b. Neural proof nets. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 26–40, Online. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Mario Lezcano Casado. 2019. Trivializations for gradient-based optimization on manifolds. *Advances in Neural Information Processing Systems*, 32.

Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. 2016. Gated graph sequence neural networks. In *Proceedings of ICLR'16*.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, 32.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Yufang Liu, Tao Ji, Yuanbin Wu, and Man Lan. 2021. Generating CCG categories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13443–13451.

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

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online. Association for Computational Linguistics.

Michael Moortgat. 1996. Multimodal linguistic inference. *JoLLI*, 5(3/4):349–385.

Richard Moot. 2015. A type-logical treebank for French. *Journal of Language Modelling Vol*, 3(1):229–264.

Richard Moot. 2019. Reconciling vectors with proofs for natural language processing. Compositionality in formal and distributional models of natural language semantics, 26th Workshop on Logic, Language, Information and Computation (WoLLIC 2019). Retrieved from `https://richardmoot.github.io/Slides/WoLLIC2019.pdf`.

Richard Moot and Mario Piazza. 2001. Linguistic applications of first order intuitionistic linear logic. *Journal of Logic, Language and Information*, 10(2):211–232.

Glyn Morrill, Oriol Valentín, and Mario Fadda. 2011. The displacement calculus. *Journal of Logic, Language and Information*, 20(1):1–48.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? *Advances in neural information processing systems*, 32.

Gertjan van Noord, Gosse Bouma, Frank Van Eynde, Daniël 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.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the long tail with tree-structured decoding of complex categories. *Transactions of the Association for Computational Linguistics*, 9:243–260.

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, Valencia, Spain. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Noam Shazeer. 2020. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. *Advances in Neural Information Processing Systems*, 32.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. In Robert Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224. Wiley-Blackwell.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. The Penn treebank: an overview. *Treebanks*, pages 5–22.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging Combinatory Categorial Grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, Online. Association for Computational Linguistics.

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, San Diego, California. Association for Computational Linguistics.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

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, Beijing, China. Association for Computational Linguistics.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems*, 32.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. 2019. Modeling graph structure in transformer for better AMR-to-text generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5459–5468, Hong Kong, China. Association for Computational Linguistics.

A Visualization of the decoding process

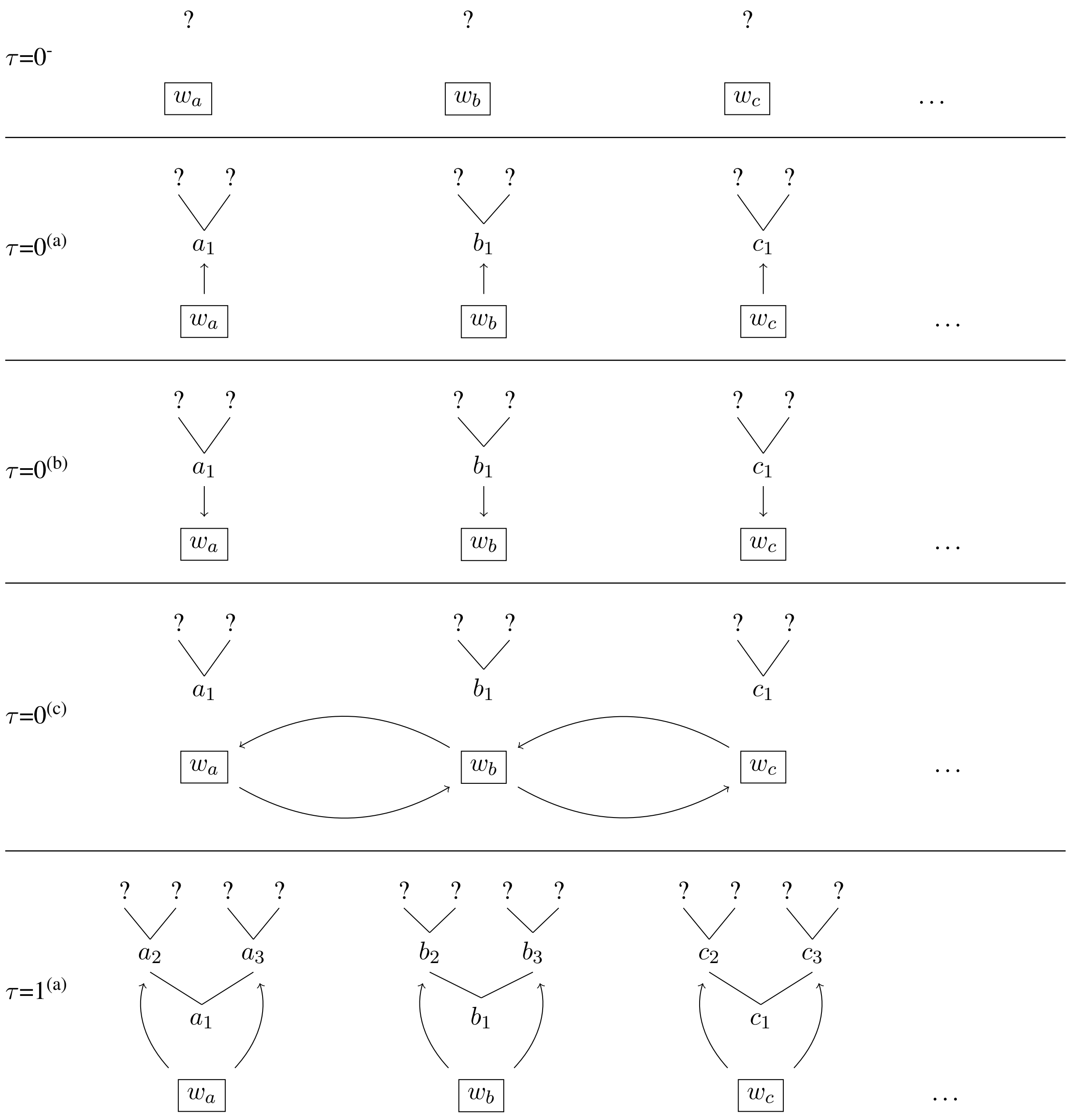
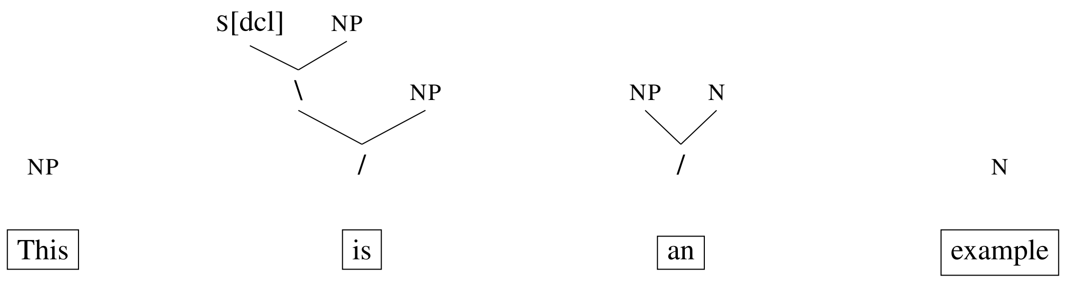
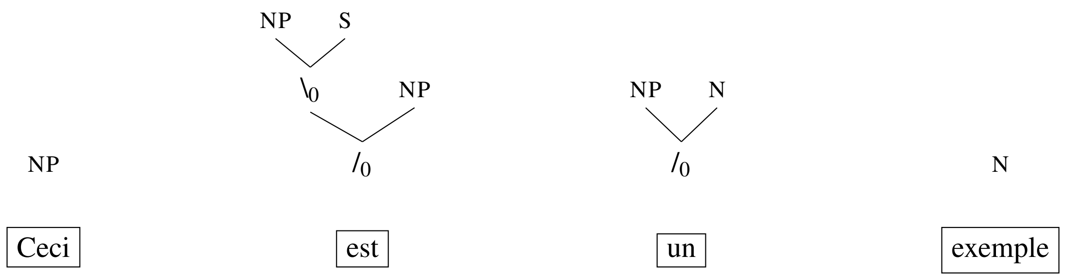


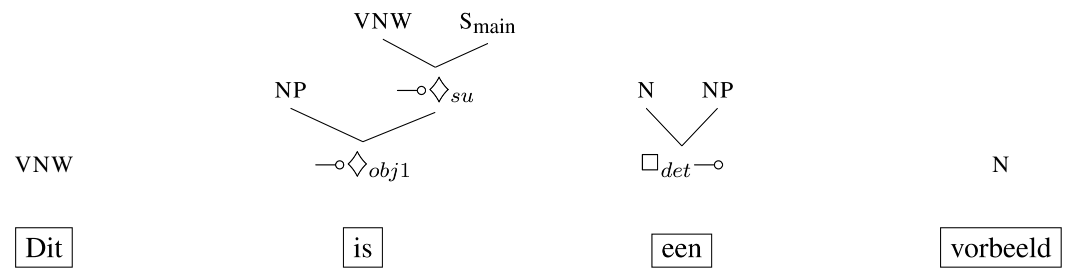
Figure 1: A frame by frame view of the first decoding step, where the abstract canvas assumes words w_a, w_b, w_c $\ldots$, rooting fully binary trees a, b, $c\ldots$, with nodes enumerated in a breadth-first fashion. For an intuition on what a concrete canvas might look like, refer to Figure 2.



(a) CCGbank



(b) French TLGbank



(c) Æthel

Figure 2: Artificial but concrete canvas examples for the three grammars experimented on.