Constraint Grammar as a Hand-Crafted Transformer

Anssi Yli-Jyrä
Helsinki Centre for Digital Humanities (HELDIG)
P.O. Box 24, 00014 University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Abstract

The differences between the rule-based NLP such as CG and the deep neural networks, such as the Transformer (Vaswani et al., 2017) are so striking that it is really hard to see any relevant conceptual links between them. However, this paper sketches a thought experiment that assumes an equivalent input-output behaviour by both systems and aligns certain structural aspects of the computation behind a practical Constraint Grammar with the computation structure of Transformer. Based on this scene, several findings are presented that state some functional similarities in the computation graphs of the systems.

1 Introduction

The Transformer architecture (simply *Transformer*) (Vaswani et al., 2017), and Constraint Grammar parsing framework (simply *CG*) are currently in the opposite ends of the continuum for different NLP technologies (Table 1). The main contrasts between these relate to the representation of word senses and the way in which the systems implement machine learning. Learning in both systems is *error-driven*, but CG can use *transformation-based learning* algorithms (Brill, 1995; Lager, 2001) that differ greatly from the backpropagation algorithm used as a part of the gradient descent optimisation of Transformer. A more striking, but superficial difference is the way how the systems traditionally represent their lexicons. A Transformer network (*a* Transformer) assumes a lexicon of word parts. This maps the word parts, or tokens, directly to a high-dimensional vector space. A Constraint Grammar parser (*a* CG) has typically access to a finite-state based lexicon that assigns, to each word, a set of morpho-syntactic readings and categories. These sets are called *cohorts*.

Since there is no obvious link between constraint grammars and deep neural networks, the two methods are seldom studied in parallel. If there are some links, they are hardly ever been pointed out. One harmful consequence of this state of affairs is that it is not known how to combine these technologies in a synthetic design. Therefore, it is especially valuable to investigate how these unrelated technologies could be aligned and even married with one another. Accordingly, the aim of the current paper is to start a discussion that seeks for cross-design understanding and synthesis. This discussion may lead to ideas that allow us to create NLP that takes advantage of both expert knowledge and big data.

2 Premise and Methodology

Talman et al. (2019) compared the performance of a CG-based machine translation (MT) system and a Transformer-based sys-

Table 1: Some contrasts

	Transformer	*CG*
based on:	neural networks	restarting automata
data need:	large	moderate
tokens:	word parts	inflected words forms
input sequence:	word embeddings	cohorts of readings
word senses:	continuous	discrete tag sequences
features:	learned feature representations	template or lattice based features
special use:	pretrained embeddings	gold annotation
learning:	backpropagation and stochastic gradient descent (SGD)	composition and transformation-based learning (TBL)

tem, bringing them thus to the same table for comparison in terms of their respective translation quality. Our aim is to consider the theoretical consequences of a different comparison that is based on a thought experiment, a powerful method mastered by G. Galilei and A. Einstein in particular.

The Premise of Equivalence. According to our thought experiment, we assume the imaginary situation where both systems would happen to compute the same, nontrivial function. Although this state of affairs is unlikely to be generally achievable, there is a realistic posibility that a Transformer is able to learn to compute the same input-output mapping as a typical CG.

The Computation Graph Method. In such a world where the same mapping would be computed by two kinds of systems, it is natural to ask whether the equivalent behaviour has something to do with a similar structure that is shared by both systems. Perhaps the kind of structure with most promising parallels is the high-level computation graph of both systems. A computation graph is a directed graph that shows the flow of information in a system that consists of several connected processing modules.

The Alignment Hypothesis. The current hypothesis is that both systems actually have corresponding computation steps that can be functionally aligned with each other.

Since the experiment facilitates the detection of analogies in design, it has potential value for further research: we may want to build robust systems where CG and statistical models complement one another, or hybrid systems that contain some computation steps from CG and some other steps from a neural encoder-decoder architecture.

3 Aligned Encoders

To argue for conceptual connections between Constraint Grammar and Transformer, we start from very general observations.

The encoder-decoder components. A Transformer is a composition of a stack of encoder networks and a stack of decoder networks: $enc \circ dec$. The layers of the *encoder networks* (*enc*) build an internal representation for the input string. Then, a multi-layer *decoder* network (*dec*) produces an output string based an internal representation.

Finding 1. *Both systems contain an* encoder component *that embeds the input sequence to a sequence of contextually disambiguated feature representations of tokens at each position.*

Proof. This is clearly true for Transformer. Constraint Grammar is an encoder that maps the sequences of ambiguous cohorts to sequences of (nearly) unambiguous cohorts that represent the contextual reading of each token in a sentence. □

The existence of a decoder component in both systems can also discussed. CG does not usually have a decoder component, but it has sometimes been extended with modules that can be seen as decoders. For example, Hurskainen (1999) and (Hurskainen and Tiedemann, 2017) describe CG-based systems where the disambiguated input string is processed further by "decoding" modules. These modules implement a mapping from the contextualised token representations to a representation of the corresponding target language tokens, and a mapping from the original word order to the target word order, etc. Since the internal structure of these modules is still somewhat different from the decoders of the Transformer architecture, their similarities cannot be demonstrated in the current work.

Information reduction. By design, the encoder stack of Transformer is a multi-layer neural network. The information content of the output is a subset of the information content that is available in the input. The rest of the input information is irrelevant for the output and is thrown out during the encoding process.

Finding 2. *The encoders of both systems are functions and thus reductionistic: the amount of information that is relevent for the output representation is not increasing inside the encoders.*

Proof. The output layer of a Transformer is a function of the input layer. A Constraint Grammar is generally known to be an iterated function that transforms the input sequence to a less ambiguous one. □

Feature vectors. Although the tokens are, in many ways, ambiguous in the beginning, Transformer assumes a finite lexicon of tokens. It embeds the tokens to a space of continuous representations.

Finding 3. *Both systems represent the input tokens as feature vectors.*

Proof. Yli-Jyrä (2011a) has demostrated how to embed the input cohorts of a CG system into finite vectors. These vectors contain the values of those hand-engineered features that are used in rule conditions and are thus relevant for the function defined by the CG grammar. Such vectors can be extended to a lossless representations from which the input cohorts in a finite lexicon can be decoded. Such a representation is comparable with the token embedding vectors used by Transformer. □

A finite number of layers. The encoder network in Transformer contains typically 6–64 similar layers (with different weights).[1]

Finding 4. *The number of layers in both encoder architectures is potentially finite.*

Proof. The finite bound for layers holds for Tranformer by definition. Yli-Jyrä (2017b) conjecture that, in practice, each CG can be viewed as a finite-visit Turing machine, which is known to be equivalent to a functional one-way finite-state automaton or transducer, see Yli-Jyrä (2017a). Such a machine model has a reading-writing head that does not cross any position in the sentence more than k times. According to the argument, this optimisation is made possible by the assumption that at most a finite amount of information needs to be communicated across each sequence position. Hulden (2011), Peltonen (2011) and Yli-Jyrä (2011a) present similar analyses for separate CG rules but they do not reach the conjecture that finite visits and bounded crossings per position would be sufficient for the correct function semantics of the whole CG parser where the rules are supposed to apply iteratively. Once the equivalence with a finite-state transducer is established, it is easy to see that a multi-layer composition of several finite-state transducers can be much more succinct way to compute the same function. Thus, both encoders have a finite number of layers in the case where the finite-visit conjecture holds. □

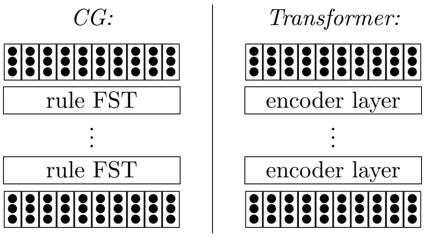

Figure 1: Alignment of the encoders

Given these four observations, we can draw a diagram (Figure 1) that aligns the computational structure of the encoder of Transformer with a CG system.

4 Aligned Sublayers

The alignment inside the corresponding layers of the encoders requires us to dig into the more detailed structure inside both CG and Transformer. There are at least four ways to approach a CG parser, but only the last one helps us to see the analogy between the two systems:

(i) Iterative Rule Application. Some CG parsers have a control mechanism that iterates over the positions of the sentence and over the disambiguation rules, trying to apply each rule at each position at a time. If non-monotonic rules are included, the parser becomes Turing complete (Kokke and Listenmaa, 2017; Yli-Jyrä, 2017b). Since Transformer has been specifically designed to have a bounded number of layers, the iterative rule application differs too much from it.

(ii) Constraint Programming. Another view on constraints treats the constraints conjuctively, requiring the output sequence to satisfy all the constraints. For some inputs, the system can be over-constrained, which would make the parser to fail, unless some constraints are relaxed (Listenmaa, 2019). Such constraint programming approach is quite different from the Transformer architecture whose encoders are position-wise, without any constraint relaxation.

(iii) Maximum Subgraph Parsing. A new approach to constraint relaxation is to re-

[1] This raises a question, could the encoder network have recurrent layers that share their weights. This would make the encoder even more similar to a CG.

formulate the constrained parsing as a maximum subgraph problem over encoded graphs. Yli-Jyrä and Gómez-Rodríguez (2017) consider complete dependency graphs with arc-factored weights and sketches an efficient parsing algorithm for maximun weighted noncrossing subgraphs that satisfy some hard constraints, such as acyclicity and connectivity. The weights are needed because the system is under-constrained. With a more general encoding (Yli-Jyrä, 2019), it is possible to remove the limitation of noncrossing parses from the parsing algorithm.

(iv) Separate Context Queries. In Yli-Jyrä (2011a), the context conditions of CG rules are represented by an efficient query-FSA (Figure 2) that can match thousands of complex conditions in parallel. The position-wise satisfaction of context conditions is encoded into strings using *position-wise flag diacritics* (Yli-Jyrä, 2011b), a generalisation of Liang's hyphenation algorithm (Liang, 1983). These diacritics of the query-FSA encode a positionwise vector that tells which contexts are present at each position.

Finding 5. *The two subnetworks of the encoder in Transformer can be compared with the computation steps used in Yli-Jyrä (2011a).*

Proof. In Transformer, each encoder network consists of two position-wise networks: (1) a self-attention network and (2) a feed-forward (FF) network. The first network queries, in parallel for every input position, an attention-weighted average vector that describes an aspect of its context in the sentence. After this, the FF network manipulates the vector that encodes the contents of each position based on the information gathered via self-attention.

The computation structure of Transformer corresponds to the fourth way to implement Constraint Grammar. The CG implementation of Yli-Jyrä (2011a) tests firsts the the context conditions of all disambiguation rules in all positions. After this, the systen chooses an input cohort, manipulates its feature representation (and the cohort), and updates the tests around the changed cohort. The system could also take some risk of suboptimal search and change multiple cohorts at the same time.

If sufficiently many positions are changed during one iteration, the CG parser can be implemented with a finite cascade of finite-state transductions. This is the general case where the system becomes quite Transformer-like. A special case of similar, cascaded application of parallel rules has been explored by Hulden (2011). □

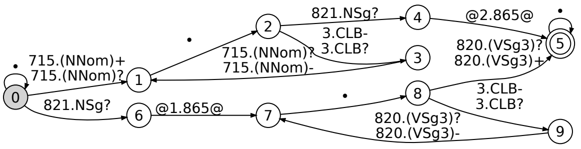

Figure 2: a small portion of a "self-attention" query network in CG

CG:	*Transformer:*
Feature-based cohort manipulations	FF network for position-wise manipulations
Recurrent "self-attention" query network	Non-recurrent self-attention network

Figure 3: Sublayers of the encoders

The alignment of the sublayers gives us a picture (Figure 3). Further analysis is needed to check whether the conditions of CG rules are simple enough to be simulated with the self-attention mechanism.

5 Conclusion

In this paper, we have aligned the high-level computational structures of CG parser and a Transformer under their functional equivalence. This resulted in findings

1. on their encoder-decoder decomposition,
2. on their reductionistic nature,
3. on their vectorized token representations,
4. on their finite number of layers, and
5. on the two steps in each encoder layer.

We also noted that the alignment is not perfect. For example, it is probable that the cohort vectors and the word embedding vectors do not represent the lexical or morphological ambiguity in the same way. Understanding the significance of this difference would be crucial for the discussion about interpretability and invertibility of token representations. Nevertheless, the alignment suggests the possibility of hybrid parsing models that would combine these architectures and their complementary strengths in NLP.

References

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Comput. Linguist.*, 21(4):543–565.

Mans Hulden. 2011. Constraint grammar parsing with left and right sequential finite transducers. In *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, pages 39–47, Blois, France. Association for Computational Linguistics.

Arvi Hurskainen. 1999. SALAMA Swahili Language Manager. *Nordic Journal of African Studies*, 8(2):139–157.

Arvi Hurskainen and Jörg Tiedemann. 2017. Rule-based machine translation from english to finnish. In *Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017*, pages 323–329. Association for Computational Linguistics.

Pepijn Kokke and Inari Listenmaa. 2017. Exploring the expressivity of constraint grammar. In *Proceedings of the NoDaLiDa 2017 Workshop on Constraint Grammar - Methods, Tools, and Applications, 22 May 2017*, pages 15–22, Gothenburg, Sweden. Linkoping University Electronic Press.

Torbjörn Lager. 2001. Transformation-based learning of rules for constraint grammar tagging. In *NODALIDA*.

Franklin Mark Liang. 1983. *Word Hy-phen-a-tion by Com-pu-ter*. Ph.D. thesis, Stanford University, Department of Computer Science.

Inari Listenmaa. 2019. *Formal Methods for Testing Grammars*. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden.

Janne Peltonen. 2011. Rajoitekielioppien toteutuksesta äärellistilaisin menetelmin (On finite-state implementation of Constraint Grammars). Master's thesis, University of Helsinki, Department of Modern Languages, Helsinki.

Aarne Talman, Umut Sulubacak, Raúl Vázquez, Yves Scherrer, Sami Virpioja, Alessandro Raganato, Arvi Hurskainen, and Jörg Tiedemann. 2019. The university of Helsinki submissions to the WMT19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 412–423, Florence, Italy. Association for Computational Linguistics.

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 *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, USA. Curran Associates Inc.

Anssi Yli-Jyrä. 2011a. An efficient constraint grammar parser based on inward deterministic automata. In *Proceedings of the NODALIDA 2011 Workshop Constraint Grammar Applications*, volume 14 of *NEALT Proceedings Series*, pages 50–60.

Anssi Yli-Jyrä. 2011b. Explorations on position-wise flag diacritics in finite-state morphology. In *Proceedings of the 18th Nordic Conference of Computational Linguistics, NODALIDA 2011, May 11-13, 2011, Riga, Latvia*, pages 262–269. Northern European Association for Language Technology (NEALT) / Tartu University Library (Estonia) / ACL.

Anssi Yli-Jyrä. 2017a. Forgotten islands of regularity in phonology. In *K + K = 120: Papers dedicated to László Kálmán and András Kornai on the occasion of their 60th birthdays.*

Anssi Yli-Jyrä. 2017b. The power of Constraint Grammar revisited. In *Proceedings of the NoDaLiDa 2017 Workshop on Constraint Grammar - Methods, Tools, and Applications, 22 May 2017*, pages 23–31, Gothenburg, Sweden. Linkoping University Electronic Press.

Anssi Yli-Jyrä. 2019. Transition-based coding and formal language theory for ordered digraphs. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*, pages 118–131, Dresden, Germany. Association for Computational Linguistics.

Anssi Yli-Jyrä and Carlos Gómez-Rodríguez. 2017. Generic axiomatization of families of noncrossing graphs in dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1745–1755. Association for Computational Linguistics.