Lambek Grisin Proofnets and Hyperedge Replacement Grammars paper Logical Methods in Natural Language Processing

A.M.Leeuwenberg ; 3666328

July 17, 2012

Abstract

Lambek-Grishin proofnets (LG proofnets) and Hyperedge Replacement grammars (HR grammars) are both graph-grammars. In Moot (2008) he gives an overview of how these formalism relate to each other. Both formalisms are presented and there is made an attempt to show the connection between LG proofnets and HR_2 grammars.

Contents

1 Introduction

Lambek Grishin proofnets are a nice way to study and parse it's typelogical grammars. This paper attemps to show that there is an other way to represent these proofnets, namely by Hyperedge Replacement grammars. An introduction to Hyperedge Replacement grammars will be given. Also Lambek Grishin proofnets will be introduced. After the explanations of the formalisms it will be shown how Hyperedge Replacement grammars can model the Lambek Grishin proofnets and some of it's complications.

2 Hyperedge Replacement Grammars

Hyperedge replacement grammars are graph-grammars. The grammar uses a special type of graphs called hypergraphs. The operation that manipulates such graphs is called hyperedge replacement. This is some form of substitution with hypergraphs. Besides hyperedge replacement there is also a set of grammar rules that can be defined. Hyperedge replacement grammars are somewhat similar to TAGs. However TAGs are tree-grammars and HR grammars are graph-grammars.

2.1 Hypergraphs

Hypergraphs are graphs that have labeled edges that can connect multiple nodes. These edges are called hyperedges. Besides these hyperedges there is a differentiation between internal and external nodes. This difference will be important for hyperedge replacement later on. I will start by giving the definition of a hypergraph[1].

Definition Hypergraph: Let Γ be an alphabet of edge labels and let σ be an alphabet of selectors. A hypergraph over Γ and σ is a tuple $\langle V, E, lab, nod, ext \rangle$, where

- V is the finite set of vertices,
- E is the finite set of hyperedges disjoint with V,
- lab is the labeling function, from E to Γ, assigning an edge label to each hyperedge,
- nod is the incidence function that associates with each edge $e \in E$ a partial function $nod(e) : \sigma \rightarrow v$, that is, it selects a vertex for every selector σ of the edge.
- ext is the external function, a partial function from σ to V, that is, for every selector σ of the hypergraph it selects a vertex.

An example of a small hypergraph is shown in figure 1.

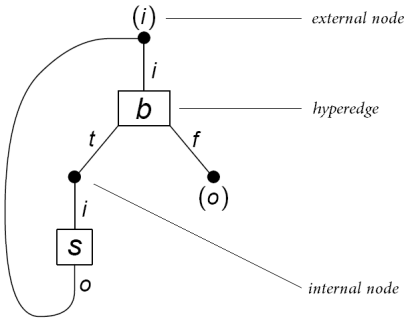

Figure 1: hypergraph[1]

Hypergraphs and their edges can be assigned a type. This is important for the hyperedge replacement. The type of a hypergraph H is the domain of the external function. The type of a hyperedge e is the domain of the incidence function using e as it's first argument. The types of hyperedges 'b' and 's' in figure 1 are $\{i, t, f\}$ and $\{i, o\}$.

2.2 Hyperedge Replacement

Hyperedge replacement is the basic operation on hypergraphs used in hyperedge replacement grammars. You can substitute a hyperedge e in hypergraph H by a hypergraph K if e and K have the same type. An example of a hyperegde replacement is given in figure 2. Hypergraph H has a hyperedge s of type $\{i, 0\}$. Hypergraph K also has type $\{i, 0\}$. This means hyperedge replacement can take place. In this case hyperedge s is 'replaced' by hypergraph K.

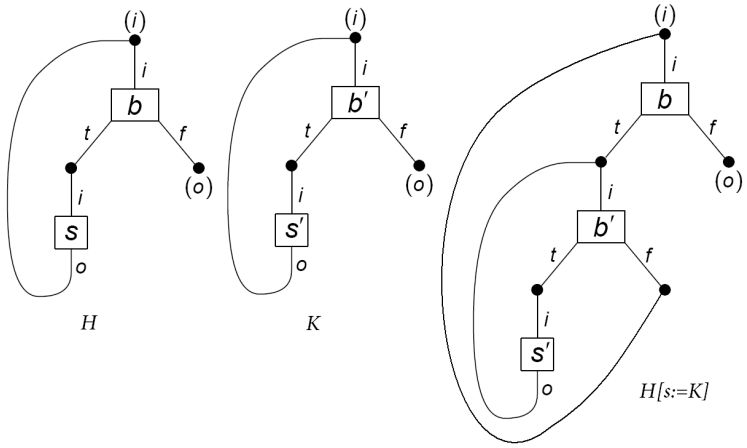

Figure 2: hyperedge replacement [1]

The formal definition of hyperedge replacement is the following.

Definition Hyperedge Replacement: Let H and K be two disjoint hypergraphs with the same set of edge labels Γ and the same set of selectors σ. Let s be an edge of H such that $\text{type}(s) = \text{type}(K)$. The hyperedge replacement of s by K, $H[s := K] = \langle V, E, lab, nod, ext \rangle$ is defined as follows.

- $V = V_H \cup V_K$
- $E = (E_H - s) \cup E_K$
- $lab = lab_H \cup lab_K$ restricted to the members of E.
- $nod = nod_H \cup nod_K$ restricted to the members of E.
- $ext = ext_H$
- For all $v \in \text{type}(s)$, $nod_H(s, v) = ext_K(v)$.

2.3 Hyperedge Replacement Grammar: $a^n b^n c^n d^n$

A hyperedge replacement grammar can be thought of as an initial hypergraph and a set of allowed hyperedge replacements. The set of hypergraphs that can be constructed by starting with the initial hypergraph and replacing it's hyperedges according to the grammar rules, in such way that there are no non-terminal edge labels left, is the graph-language the hyperedge replacement grammars describes. The formal definition of a hyperedge replacement grammar is the following.

Definition Hyperedge Replacement Grammar (HR grammar): A HR grammar is a tuple $G = \langle N, T, \sigma, P, S \rangle$ such that:

- N is the alphabet of nonterminal edge labels.
- T is the disjoint alphabet of terminal edge labels.
- σ is the alphabet of selectors.
- P is the finite set of productions.
- $S \in N$ is the start nonterminal symbol.

An example of a HR grammar is shown in figure 3. The graph-grammar corresponds to the string language $L = \{a^n b^n c^n d^n\}$. A derivation of the hypergraph corresponding to the string "$aabbccdd$" would be $S \to R \to R \to T$.

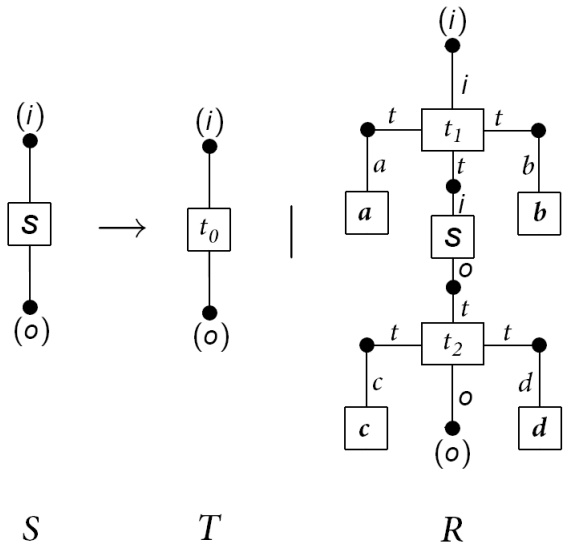

Figure 3: HR grammar for $L = \{a^n b^n c^n d^n\}$ [1]

HR grammars can be assigned a rank. This rank represents the maximum number of tentacles of a nonterminal symbol in the grammar. The rank of the HR grammar in figure 3 would be 2 since there is only one nonterminal symbol, namelijk S and S has two tentacles. The rank of a HR grammar relates to the potential expressivity of the grammar [1]. HR grammars with rank 2 (HR_2 grammars) are strongly equivalent with TAG [2].

3 Lambek Grishin Graphical Calculus

Lambek Grishin calculus (LG calculus) is an extension of the nonassociative Lambek calculus (NL). The LG calculus is the symmetric version of the NL calculus. With this symmetry also comes a set of postulates that allow a controlled form of associativity and commutativity that applies to the Grishin connectives. This is somewhat similar to NL$\Diamond$ where the structural rules apply to the $\Diamond$ and $\Box$ operator. Like with NL and NL$\Diamond$ there is also a proofnet calculus of LG.

3.1 Proofnets for LG

The symmetry of the LG can be seen clearly in the proofnets. The Grishin connectives are a copy of their Lambek counterpart except for the fact that they allow reasoning in the opposite direction, by having multiple conclusions. The Lambek proofnet links and their Grishin counterparts are shown in figure 4.

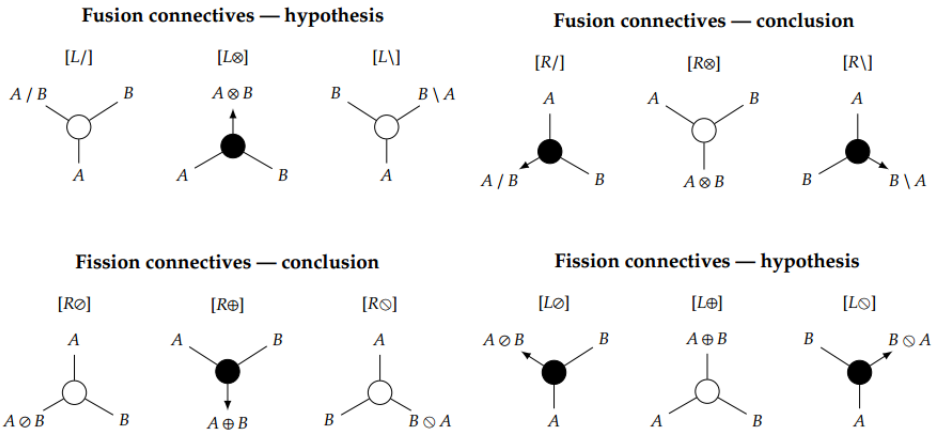

Figure 4: Proonet links for LG [3]

There are also slightly different contraction rules for the Grishin connectives (figure 5) to allow contraction to take place in the opposite direction.

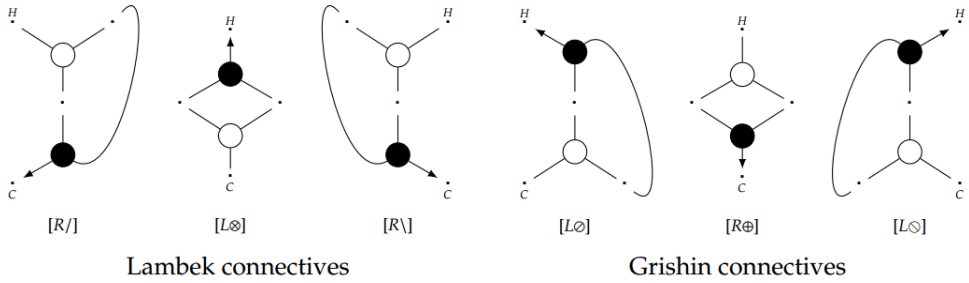

Figure 5: LG Contractions [3]

3.2 Structural Rules

These new links and contractions however do not deal with the postulates of LG that allow associativity and commutativity on the Grishin connectives. These postulates are represented in the form of structural manipulations that can be applied to the proof nets. In figure 6 the four types of manipulation are shown.

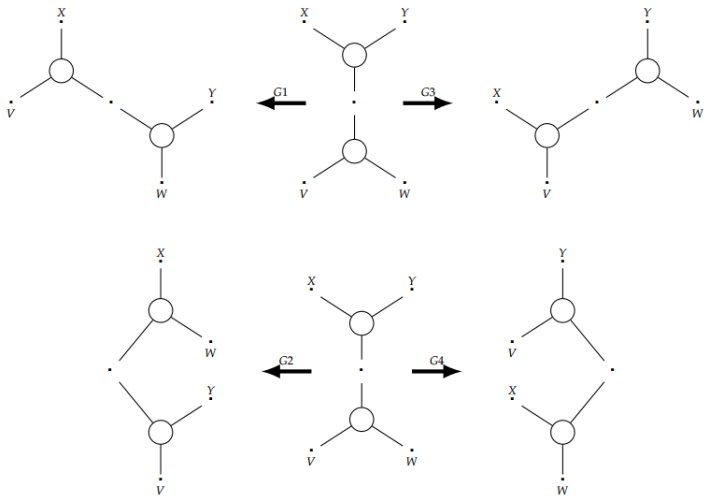

Figure 6: Structural rules for 'mixed associativity' and 'mixed commutativity' [3]

The $G1$ and $G3$ are allowing 'mixed associativity' and the $G2$ and $G4$ are allowing 'mixed commutativity'. The commutativity can easily be seen because the horizontal (and vertical) order of x and y, and v and w has changed. Which does not happen with the $G1$ and $G3$ rules.

3.3 LG Grammar: MIX

LG proofnets can be used as a graph-grammar. Take a grammar G that describes a language L. G would consist of types or proofnets that are assigned to the members of a string alphabet. This would be called the lexicon. If it is possible to derive a certain conclusion say S from the proofnets that correspond to a string of lexical items, the string corresponding to those lexical items is a member of language L. You can only derive an S by constructing a tensor tree with your lexical items as hypotheses and S as their conclusion. An example of an LG grammar that describes the MIX language ($MIX = \{w \in \{a,b,c\}^+ | w|_a = |w|_b = |w|_c\}$) would be the following.

$$a :: A \otimes \phi$$
$$b :: \phi \oslash (S \otimes (A \otimes (S \oslash C)))$$
$$c :: \phi \oslash C$$

Where ϕ is S/S unless it appears in the lexical item that is last in the to be derived string. An example derivation of the word "bca" would require the following types:

$$a :: A \otimes S$$
$$b :: (S/S) \oslash (S \otimes (A \otimes (S \oslash C)))$$
$$c :: (S/S) \oslash C$$

Their lexical axiom unfolding results in a set of proofnets that can be connected and then contracted to a tensor tree using the contractions rules and the Grishin interactions. The unfoldings and their connections are shown in figure 7 on the left. The graph on the right is when the connections are made. Note that the connections are crossed. From this you can see that the rules for 'mixed commutativity' are needed to derive a tensor tree.

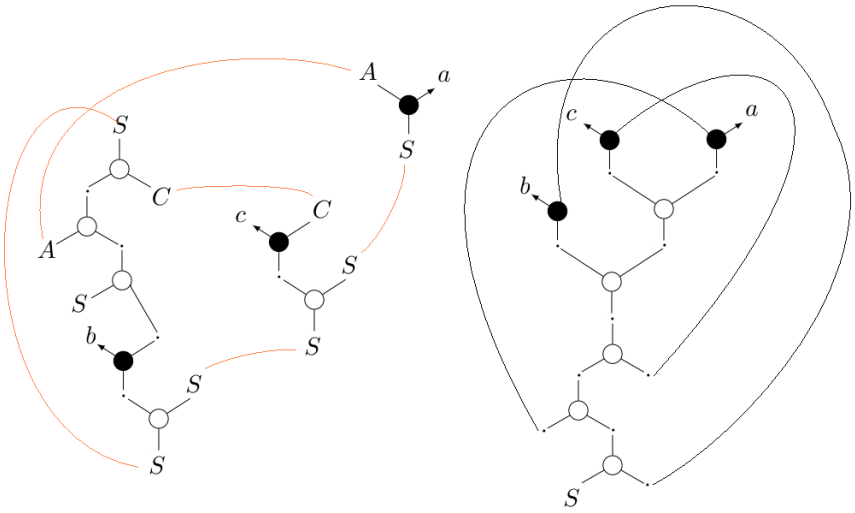

Figure 7: Lexical unfoldings and their contractions

The final tensor tree that you would want to derive is shown in figure 8. This final tree can be derived by using the following rules to the connected graph in figure 7: $G3 \rightarrow G2 \rightarrow L\oslash \rightarrow G4 \rightarrow L\otimes \rightarrow L\oslash$.

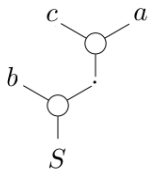

Figure 8: Final tensor tree

4 LG conversion to HR_2 grammar

$L^5 = \{a^n b^n c^n d^n e^n\}$ is not a TAL [4]. Also MIX as presented earlier is not a TAL [5]. These two languages can be expressed by an LG grammar [6]. There appears to be an equivalence between TAG and HR_2 grammars [2]. A certain subset of LG grammars can be converted to an HR_2 grammar. MIX and L^5 are clearly not in this subset due to transitivity. There are

some conditions that an LG grammar should satisfy in order to be converted to an HR_2 grammar [1].

4.1 HR_2 of LG proofnets

In Moot (2008) he shows how to build an HR_2 that models the LG proofnets [1]. I will here present this HR_2. The HR_2 proofnets for LG are build top down rather than bottom up. This means you start with an S from which you can only build tensor trees (which would normally be your goal). From these tensor trees there are grammar rules that allow you to do 'inverted' contractions. Also you can apply the Grishin rules. In the end you expect the lexical items to be constructed (as in figure 7). The lexical items you would start with when you did your normal LG proofnet derivation. This is how the HR_2 grammar for LG proofnets is build. In figure 8 you see the HR_2-grammar rules to build tensor trees. The nonterminal T constructs trees and the nonterminal V constructs vertices. The small indices in the edgelabels stand for the function of the hyperedge. In figure 8 you can also see the different types of 'connections' that a V can end up to be.

$00 \sim$ cut	$01 \sim$ flow down
$10 \sim$ flow up	$11 \sim$ axiom

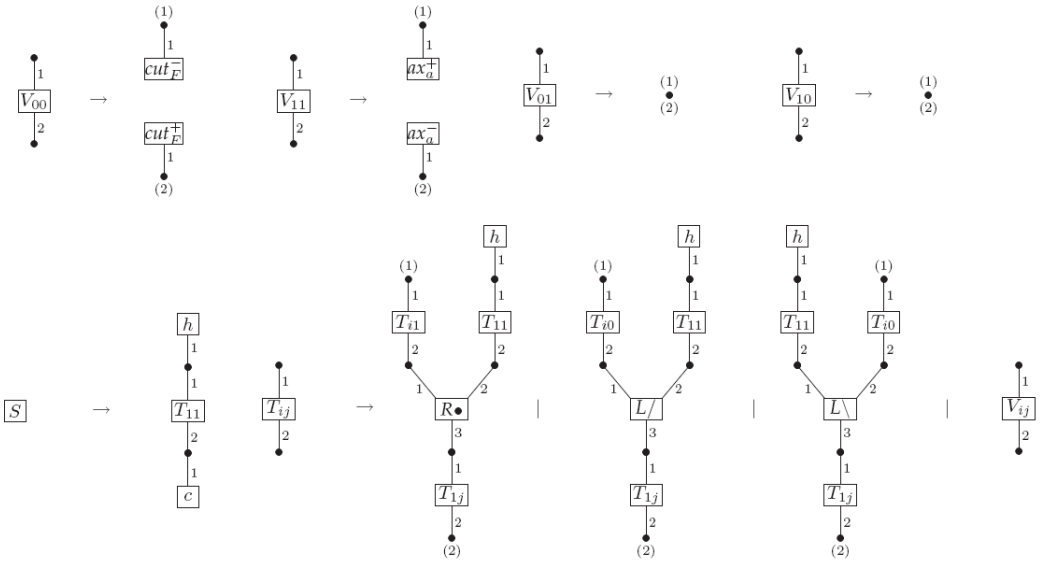

Figure 9: HR_2 for tensor trees [1]

For each tensor-par (or cotensor) pair that could possibly contract there is a rule that constructs the pair of links before they were contracted. One

example of such rules is given in figure 9. It concerns the rules for the par link $L\oslash$.

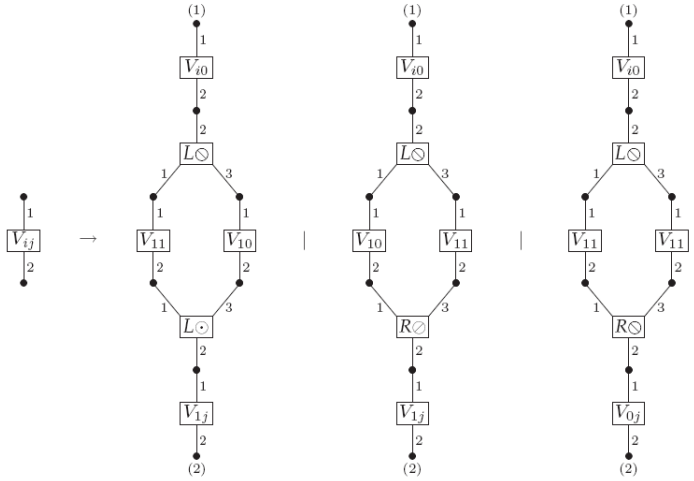

Figure 10: HR_2 prooftree rules for $L\oslash$ [1]

You can visually compare these rules to the normal LG proofnets in figure 4 and 5. You can then see that the $L\oslash$ link can only contract with it's tensor counterparts $L\oplus$, $R\oslash$ and $R\oslash$.

Now all that's left are the Grishin Interactions. Instead of manipulating a tree structurally, which happens in the nornal LG Grishin Interactions, the rules allow you to immediately construct the 'adjusted' tree from a T_{ij} edge. In figure 10 one set of the interactions is shown.

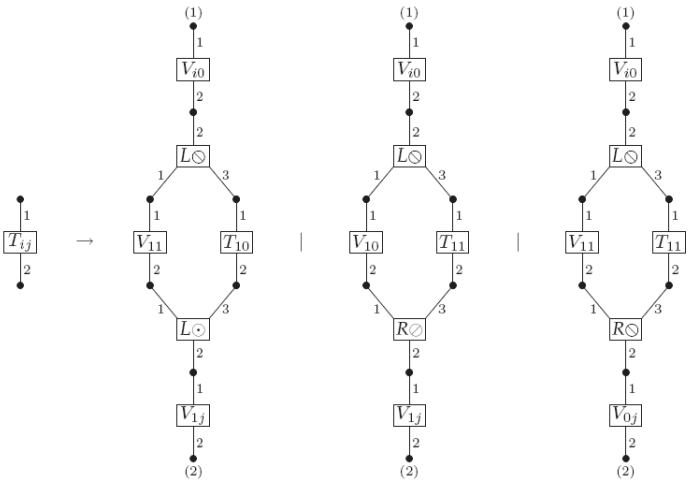

Figure 11: HR_2 Grishin interactions IV [1]

4.2 Limitations to HR_2

However not all LG proofnets can be derived by the HR_2 of LG. Since the proofnet in figure 7, which does derive a valid tensor tree, cannot be constructed by the HR_2. The crossed connections would have to be resolved by the Grishin rules, since they do in the normal LG proofnets, however they do not in the HR_2. As long as the connections are planair you can build the prooftree using the HR_2 grammar. It is intuitive that the HR_2 cannot build all proofnets as said before TAG and HR_2 are equivalent and LG and TAG are not.

5 Conclusions

Lambek Grishin proofnets can be represented by Hyperedge Replacement grammars of rank 2. For a subset of the LG grammars there is a representative HR_2 however not for every LG. For LG's that are not well nested, such as MIX, there is no HR_2.

References

[1] Richard Moot. Type-logical and hyperedge replacement grammars. juli 2008. Draft.

[2] Richard Moot. Lambek grammars, tree adjoining grammars and hyperedge replacement grammars. *Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms*, 2008.

[3] Michael Moortgat and Richard Moot. Proofs nets and the categorial flow of information. LIRa seminar, 2011.

[4] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer, first edition, 2010.

[5] Makoto Kanazawa and Sylvain Salvati. Mix is not a tree-adjoining language. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012.

[6] Matthijs Melissen. The generative capacity of the lambekgrishin calculus: A new lower bound. *Proceedings of the 14th Formal Grammar conference, Bordeaux*, 2009.