Lambek Grammars, Tree Adjoining Grammars and Hyperedge Replacement Grammars

Richard Moot
LaBRI(CNRS), INRIA Bordeaux SO & Bordeaux University
351, cours de la Libération
33405 Talence, France
Richard.Moot@labri.fr

Abstract

Two recent extension of the non-associative Lambek calculus, the Lambek-Grishin calculus and the multimodal Lambek calculus, are shown to generate class of languages as tree adjoining grammars, using (tree generating) hyperedge replacement grammars as an intermediate step. As a consequence both extensions are mildly context-sensitive formalisms and benefit from polynomial parsing algorithms.

1 Introduction

Joshi et al., (1991) have shown that many independently proposed mildly context-sensitive grammar formalisms — combinatory categorial grammars, head grammars, linear indexed grammars and tree adjoining grammars (TAGs) — generate the same class of string languages.

For the Lambek calculus **L** (Lambek, 1958), Pentus (1995) has shown that **L** grammars generate only context-free languages. Two recent incarnations of Lambek grammars have sought to extend the generative capacity the Lambek calculus: the *multimodal* Lambek calculus $\mathbf{NL}\Diamond_{\mathcal{R}}$ (Moortgat, 1997) and the *Lambek-Grishin* calculus **LG** (Moortgat, 2007). Both of these systems use the non-associative Lambek calculus **NL** (Lambek, 1961), for which polynomial algorithms exist (de Groote, 1999; Capelletti, 2007), as their base, but add interaction principles to augment the descriptive power. While both systems have been shown to handle linguistic phenomena for which no satisfactory Lambek calculus analysis exists, little is known about the exact class of languages generated by either system or about the complexity cost of adding these interaction principles.

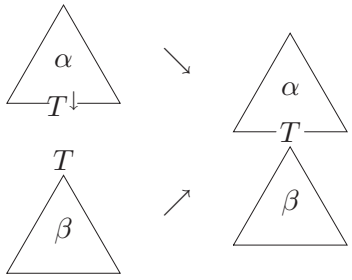

Figure 1: Substitution

In the current paper I shown that both $\mathbf{NL}\Diamond_{\mathcal{R}}$ and **LG** generate the same class of languages as TAGs, using hyperedge replacement grammars as an intermediate step.

2 Tree Adjoining Grammars and Hyperedge Replacement Grammars

Tree Adjoining Grammars (Joshi and Schabes, 1997) combine trees using the operations of *substitution* (shown in Figure 1) which replaces a nonterminal leaf $T^{\downarrow}$ by a tree with root T and *adjunction* (shown in Figure 2) which replaces and internal node A by a tree with root node A and foot node A^*.

Formally, TAGs are defined as follows.

Definition 1 *An TAG is a tuple* $\langle \Sigma, N_S, N_A, \mathcal{I}, \mathcal{A} \rangle$ *such that*

- Σ, N_S *and* N_A *and three disjoint alphabets of terminals, substitution nonterminals and adjunction nonterminals respectively, I will use upper case letters* $T, U, \ldots$ *and of course*

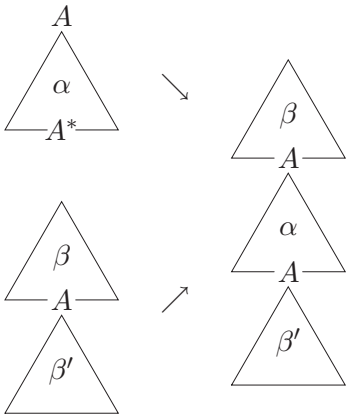

Figure 2: Adjunction

the distinguished start symbol S to stand for members of N_S whereas I will use upper case letters $A, B, \ldots$ for members of N_A.

- *$\mathcal{I}$ is a finite set of* initial trees,
- *$\mathcal{A}$ is a finite set of* auxiliary trees.

The trees in $\mathcal{I} \cup \mathcal{A}$ are called the elementary trees.

Trees are subject to the following conditions:

- *the root nodes of all initial trees are members of N_S,*
- *the root nodes of all auxiliary trees are members of N_A,*
- *every auxiliary tree has exactly one leaf which is a member of N_A. This leaf is called the* foot node,
- *all other leaves of elementary trees are members of $N_S \cup \Sigma$.*

A TAG satisfying the additional condition that all elementary trees have exactly one terminal leaf is called a lexicalized *tree adjoining grammar (LTAG). This leaf is called the* lexical anchor,

In addition, a TAG is allowed to specify constraints on adjunction. Let $A \in N_A$ and let t be the set of auxiliary trees with root node A and foot node A^*. A node A in a elementary tree α is said to have selective adjunction *in case it specifies a subset $t' \subsetneq t$ of trees which are allowed to adjoin at this node. The special case where $t' = \emptyset$ is called* null adjunction. *Finally, a node can specify* obligatory adjunction *where an auxiliary tree has to be adjoined at the node.*

The only difference with the standard definition of tree adjoining grammars (Joshi and Schabes, 1997) is the use of separate alphabets for auxiliary and substitution nonterminals. In addition to making the substitution marker $T^{\downarrow}$ and the foot node marker A^* technically superfluous, this will make the different embedding results which follow slightly easier to prove.

Definition 2 *An LTAG' grammar G is an LTAG satisfying the following additional conditions.*

- *all internal nodes of elementary trees have exactly two daughters,*
- *every adjunction node either specifies the null adjunction or the obligatory adjunction constraint without any selectional restrictions,*
- *every adjunction node is on the path from the lexical anchor to the root of the tree.*

The definition of LTAG' is very close to the definition of *normal* or *spinal form* LTAGs used by Joshi et al. (1991) and by Vijay-Shanker and Weir (1994) to show correspondence between LTAGs and combinatory categorial grammars, so it should be no surprise it will serve as a way to shown inclusion of tree adjoining languages in multimodal and Lambek-Grishin languages. The only difference is that the adjunction nodes are required to be on the path from the root to the *lexical anchor* instead of the foot node.

Lemma 3 *For every LTAG grammar G there is a weakly equivalent LTAG' grammar G'.*

Proof (sketch) The proof is analogous to the proof of Vijay-Shanker and Weir (1994). □

A *hypergraph* is a set of *hyperedges*, portrayed as an edge label in a rectangular box, which can be incident to any number of vertices. These connections are portrayed by lines (called 'tentacles') labelled $0, \ldots, n$ (the selectors) for a hyperedge of arity $n + 1$. A *hyperedge replacement grammar* (Engelfriet, 1997; Drewes et al., 1997) replaces a hyperedge with a nonterminal symbol by a hypergraph.

The rank of a terminal or nonterminal symbol is the number of its tentacles. The rank k of a HR grammar is the maximum number of tentacles of a nonterminal symbol. We will be particularly interested in HR grammars of rank two (HR_2) even

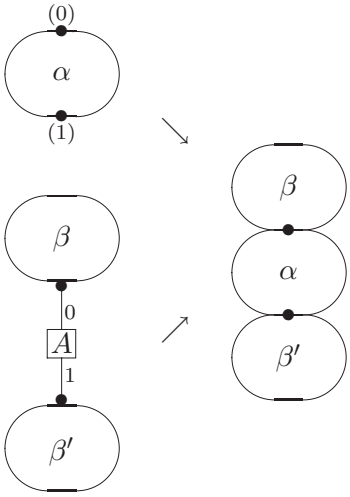

Figure 3: Hyperedge Replacement

though we do use *terminal* nodes with multiple edges.

Figure 3 shows an example of hyperedge replacement for a nonterminal A of rank two. It is attached to two vertices, represented by fat dots, with selectors 0 and 1 respectively. If $A \rightarrow \alpha$ is a rule in the HR grammar, we can replace the hyperedge A by first deleting it, then identifying the external node (0) of α with the node which was connected to tentacle 0 and external node (1) of α with the node which was connected to tentacle 1. The similarity with Figure 2 should be striking. Note however, that since the grammatical objects of hyperedge replacement grammars are *hypergraphs*, β and β' need not be disjoint. In fact, even the two tentacles are allowed to reach the same vertex. However, when we restrict the right hand sides of rules to be trees[1] then, as we will see in Lemma 5, hyperedge replacement and adjunction will correspond exactly.

There are several ways to represent trees in HR grammars, but the following will turn out to be convenient for our applications. A node with label A and n daughters[2] is represented as a hyperedge A with $n+1$ tentacles, with tentacle 0 pointing towards the parent node and tentacles $1, \ldots, n$ selecting its daughters from left to right.

Definition 4 *A hypergraph H is a (hyper-)*tree *iff every node in H is incident to two hyperedges, once by a selector 0 and once by a selector > 0, except the root node, which is incident to a single hyperedge by selector 0.*

Lemma 5 *HR_2 grammars generating trees and TAG grammars are strongly equivalent.*

Proof (sketch) From TAG to HR_2, we start with a TAG G and categorise the different adjunction nodes, introducing new symbols whenever two nodes labelled by the same symbol of N_A either select a different set of trees or differ with respect to obligatory adjunction to obtain a TAG G' which is equivalent to G up to a relabelling of the members of N_A.

Now let t be an initial tree with root node T in G', we transform it into a hypertree t' corresponding to Definition 4, with each adjunction point replaced by a unary branch with the nonterminal A corresponding to the adjunctions possible at the node and each leaf U marked for substitution replaced by a nonterminal leaf U and add rule $T \rightarrow t'$ to the HR grammar.

Each of the members of $A \in N_A$ in G' has a set of auxiliary trees t assigned to it as well as an indication of whether or not adjunction is obligatory. For each $\alpha \in t$ we add a rule $A \rightarrow \alpha$ to the grammar. In addition, if adjunction is not obligatory we add a rule $A \rightarrow \bullet$, eliminating the nonterminal hyperedge.

Now every adjunction corresponds to a hyperedge replacement as shown in Figure 3 and every substitution to the same figure, but with both the 1 tentacle and the β' subtree removed.

From HR_2 to TAG we use the fact that we generate a tree and that all nonterminals are of rank ≤ 2.

Suppose A is a nonterminal of rank two and $A \rightarrow \alpha$ is a rule in the HR_2 grammar G. In case α is a single node $\bullet$ we mark all adjunctions of a nonterminal A as optional in the grammar. If not, we add the auxiliary tree α' which we obtain from α by labelling the external node (0) by A and the external node (1) by A^* to the TAG.

Suppose T is a nonterminal of rank one and $T \rightarrow t$ is a rule in G. By Definition 4, in order for this rule to be productive the single external node has to be the root. We label the root by T and add the resulting tree as initial tree to the TAG. Again, it is easy to see that every hyperedge replacement of a nonterminal of rank 1 corresponds to a substitution and every hyperedge replacement of rank 2

[1] This is $TR(HR_{tr})$ from (Engelfriet and Maneth, 2000).

[2] We assume here that all occurrences of A have the same number of daughters, which we can accomplish by a simple renaming, if necessary.

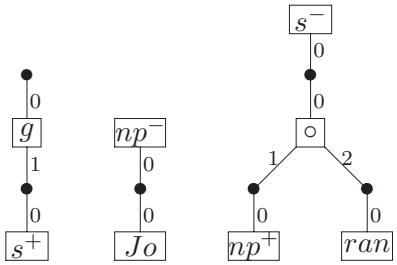

Figure 4: AB lexicon

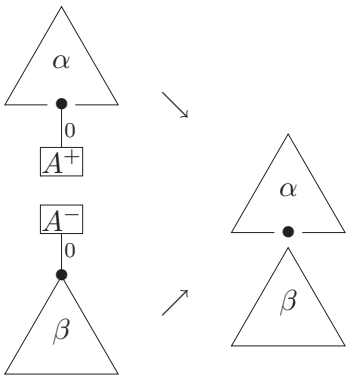

Figure 5: The axiom rule

corresponds to an adjunction in the generated TAG grammar. □

3 Lambek Grammars

Before talking about Lambek grammars, seen from a hypergraph perspective, it is useful to first show a lexicon for AB grammars. Figure 4 shows a trivial AB lexicon for the phrase 'Jo ran'. As usual in categorial grammars, we distinguish between positive and negative occurrences of atomic formulas.[3]

The tree on the left indicates that the goal formula g of this grammar is a positive s atomic formula. The lexical entry for 'ran' indicates it is looking for an np to its left to produce an s, whereas the entry for 'Jo' simply provides an np. Note that because of the extra unary hyperedge at the root nodes, the two lexical trees are not trees according to Definition 4. I will refer to them as *typed* trees.

The axiom rule (Figure 5) shows how positive and negative formulas of the same type cancel each-other out by identifying the nodes selected by the two unary hyperedges. The resemblance with the substitution operation in Figure 1 should be clear, though we do not require α and β to be disjoint: instead we require that an AB *derivation* — a set of applications of the axiom rule — ends in a (non-typed) tree where all leaves are labelled by terminal symbols.

With respect to Lambek grammars, we are primarily interested in two recent extensions of it, the multimodal Lambek calculus with two modes interacting by means of the mixed associativity and mixed commutativity structural rules (Moortgat, 1997) and the Lambek-Grishin calculus with the Grishin class IV interactions (Moortgat, 2007).

Extending the AB hypergraph calculus in this way involves adding new constructors and graph contractions which eliminate them. This moves the hypergraph calculus (or at least the intermediate structures in the derivations) further away from trees, but we will continue to require that the result of connecting and contracting the graph will be a (non-typed) tree. Contractions will correspond to the logical rules $[R/_i]$, $[L\bullet_i]$ and $[R\backslash_i]$ in the $\mathbf{NL}\Diamond_{\mathcal{R}}$ case and to the logical rules $[L\oslash]$, $[R\odot]$ and $[L\otimes]$ in the **LG** case. The contraction for $[L\bullet]$ is shown in the middle and on the right of Figure 6.

An additional constraint on the trees will be that in only contains mode 0 in the $\mathbf{NL}\Diamond_{\mathcal{R}}$ case and that it doesn't contain the 'inverse' Grishin structural connective ';' in the **LG** case. This has as a consequence that in any proof, the Grishin connectives and the connectives for mode 1 can only occur in pairs.

The calculus sketched here is just a hypergraph interpretation of the proof nets for the multimodal Lambek calculus of Moot and Puite (2002) and their extension to **LG** of Moot (2007), who show that it is sound and complete with respect to the sequent calculus.

As we have seen, it is trivial to model the substitution operation: in this respect substitution is modelled in a way which is equivalent up to notational choices to the work on partial proof trees (Joshi and Kulick, 1997).

Figure 7 shows how to model the adjunction operation, with the solution for $\mathbf{NL}\Diamond_{\mathcal{R}}$ on the left and

[3] Negative formulas correspond to resources we have and positive formulas correspond to resources we need. Think of A^+ as being similar to $A^{\downarrow}$.

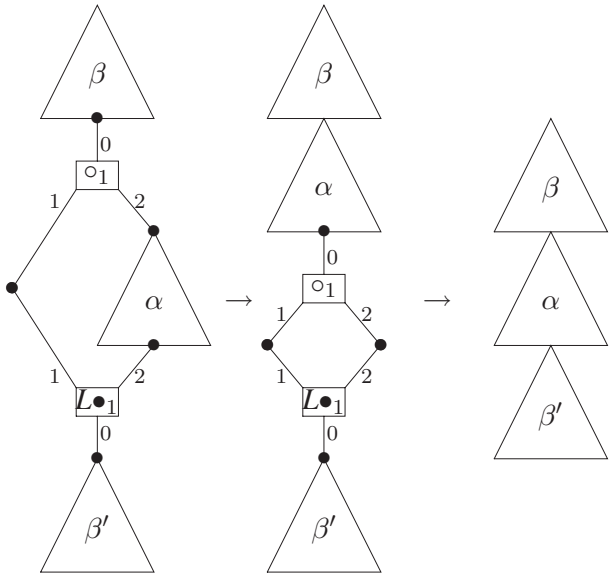

Figure 6: A contraction and structural rules for simulating adjunction

the solution for **LG** on the right.

Figure 6 shows how this allows us to adjoin a hypergraph corresponding to an auxiliary tree to this adjunction point. After two axiom rules, we will have the structure shown on the left. The mixed associativity and commutativity structural rules allow us to move the $\circ_1$ hyperedge down the tree until we have the structure shown in the middle. Finally, we apply the contraction, deleting the $\circ_1$ and $L\bullet_1$ edges and identifying the two nodes marked by selector 0 to obtain the structure on the right. Remark how these steps together perform an adjunction operation. For the sake of efficiency we will usually not apply the structural rules explicitly. Instead, we will use a generalised contraction, which moves directly from the left of the figure to the structure on the right.

Lemma 6 *If G is an LTAG' grammar, then there exists a strongly equivalent **NL**$\diamond_{\mathcal{R}}$ grammar G' and a strongly equivalent **LG** grammar G''.*

Proof (sketch) For each lexical tree t of G we construct a lexical tree t' in G' and a lexical tree t'' in G'', translating every adjunction point by the left hand side of Figure 7 for G' and by its right hand side for G''.

Now let d be a LTAG' derivation using grammar G. We translate this derivation into an **NL**$\diamond_{\mathcal{R}}$ derivation d' and an **LG** derivation d'' as follows:

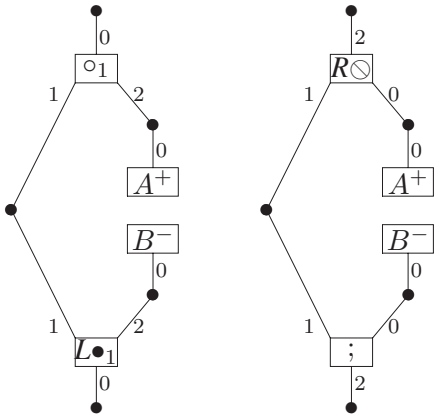

Figure 7: Lambek hypergraphs for adjunction

- Whenever we substitute a tree with root T for a leaf $T^{\downarrow}$ we perform the corresponding axiom connection in d' and d'', as shown in Figure 5.
- Whenever we adjoin a tree with root A and foot A^* we perform the A axiom for the root node, the B axiom for the foot node followed by the generalised contraction shown in Figure 6.

In order to show we generate *only* the LTAG' derivations we have to show that no other combination of axioms will produce a proof net. Given the separation of non-terminals into N_S and N_A as well as the contraction requirement this is trivial. □

To complete the proofs, we show that there is an HR grammar G' generating the hypergraphs corresponding to the proofs of an **LG** or **NL**$\diamond_{\mathcal{R}}$ grammar G, that is to say the grammar G' generates sequences of lexical graphs which, using axiom connections and generalised contractions contract to a tree.

Lemma 7 *If G is a Lambek Grammar, then there exists a strongly equivalent HR_2 grammar G'.*

Proof (sketch) Let G be a **NL**$\diamond_{\mathcal{R}}$ or an **LG** grammar. We generate a hyperedge replacement grammar H of rank two which generate all proof nets, that is to say all lexical graphs which by means of axiom connections and generalised contractions convert to a tree. Conceptually, we can think of this HR grammar as operating in three phases:

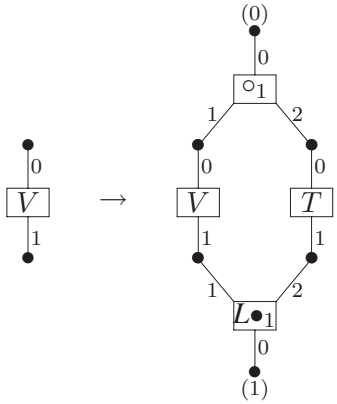

Figure 8: Lambek grammar as HR grammar — adjunction

1. Expand T (tree) nonterminals to generate *all* binary branching trees with V (vertex) nonterminals between all branches.

2. Expand V nonterminals to perform generalised expansions, that is to say inverse contractions, as shown in Figure 8.

3. Erase the V nonterminals which correspond to 'flow' formulas and disconnect the V nonterminals which correspond to axioms and as shown in Figure 9.

Now, given a sequence of lexical graphs, a total matching of the positive and negative axiomatic formulas and a sequence s of generalised contractions contracting this proof net to a tree we can generate an HR derivation d by induction on the length of s. The induction hypothesis is that during steps 1 and 2 we always have a hypergraph corresponding to the proof net $\mathcal{P}$ which has a V edge for every vertex in $\mathcal{P}$.

In case s is 0, there are no expansions and we already a binary tree with a V edge for every vertex in it. Since the T rules allow us to generate any binary branching tree, G can generate this tree as well. For every V edge in the tree which is the result of an axiom rule, we apply its inverse, shown in Figure 9 on the right, and we erase all other V edges as shown on the left of the same figure.

In case $s > 0$, we know by induction hypothesis that the hypergraph representation of the proof net we are constructing has a V edge for each of the vertices in the proof net. Because each subtree of the hypergraph has started as a T edge, this

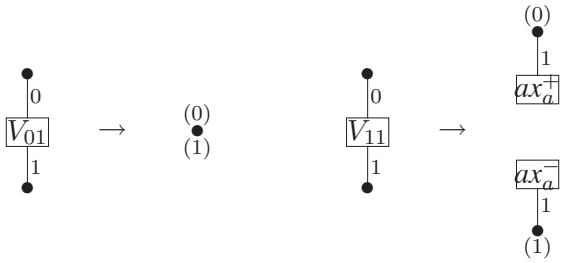

Figure 9: Lambek grammar as HR grammar — flow/axiom

is true of the α subtree as well. We rearrange the HR derivation in such a way that all expansions of the T edge into α occur at the end, then insert the expansion shown in Figure 8 just before this sequence. The result is a valid HR derivation of a hypergraph which contracts to the same tree as the proof net. □

Figure 10 summarises the different inclusions with their corresponding lemma's.

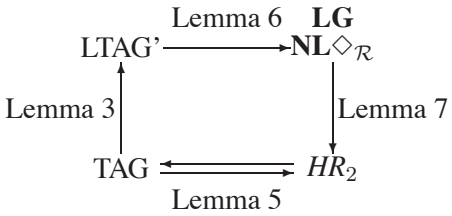

Figure 10: A summary of the previous lemma's

The following corollary follows immediately.

Corollary 8 ***NL***$\diamond_{\mathcal{R}}$ *and* ***LG*** *grammars generate mildly context-sensitive languages*.

4 Polynomial Parsing

The strong correspondence between Lambek grammars and hyperedge replacement grammars does not immediately give us polynomial parsing for Lambek grammars: as shown in (Drewes et al., 1997) for example, even hyperedge replacement grammars of rank 2 can generate NP complete graph languages, such as the Hamiltonian path problem. Lautemann (1990) presents a (very abstract) version of the well-known Cocke, Kasami and Younger algorithm for context-free string grammars (Hopcroft and Ullman, 1979) for hyperedge replacement grammars and presents two ways of obtaining polynomial complexity, the first of which will interest us here. It uses the

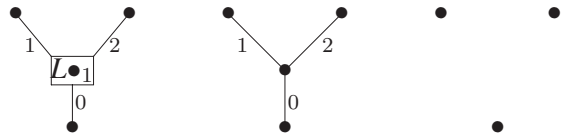

Figure 11: s-separability of the bipartite graph corresponding to a proof net

notion of s-separability, which corresponds to the maximum number of connected components in a graph when s vertices are deleted.

Theorem 9 (Lautemann (1990)) *If HR grammar G is of rank s and $s\text{-}sep_{L(G)} = O(1)$ then $L(G) \in$ LOGCFL*[4].

Corollary 10 *For any sentence length w, **NL**$\diamond_{\mathcal{R}}$ and **LG** grammars are in LOGCFL.*

Proof Lautemann discusses HR grammars generating graph languages and not hypergraph languages. However, this is not a real restriction given that we can interpret a hypergraph as a bipartite graph with the vertices corresponding to the vertices in the hypergraph as one partition and the hyperedges as the second partition. Figure 11 shows this interpretation of the leftmost graph in the middle of the figure. On the right, we can see how deleting the central vertex, corresponding to the hyperedge in the original graph on the left, results in (at most) three distinct components. The rank of the hyperedge replacement grammar for proof nets is two. Finally, for a sentence with w words, we start with w disjoint lexical graphs. Therefore, deleting two vertices from w disjoint graphs produces at most $w + 6$ disjoint graphs. □

It is slightly unsatisfactory that this complexity result uses the number of words in the sentence w as a constant, though it seems possible to eliminate the constant by using a slightly more specific algorithm.

Pentus (2006) has shown that the associative Lambek calculus **L** is NP complete. The reason deleting a vertex from an **NL**$\diamond_{\mathcal{R}}$ or **LG** hypergraph gives at most three different hypergraphs is because we work with a non-associative system. Although Moot and Puite (2002) show that associativity is easily accommodated in the proof net calculus, it results in a system without upper bound on the number of daughters which a node can have. Therefore, deleting a vertex from an **L** hypergraph can result in an unbounded number of connected components.

5 Discussion and Future Work

When looking at the proof of Lemma 5, it is clear that HR_1 grammars generate substitution only TAGs, whereas HR_2 grammars generate TAGs. The tree generating power of HR grammars increases with the maximum rank of the grammar. For example, it is easy to generate the non-TAG language $a^n b^n c^n d^n e^n f^n$ using nonterminals of rank 3. In general, Engelfriet and Maneth (2000) show that $TR(HR_{tr})$, the set of tree languages generated by hyperedge replacement grammar such that the right hand side of all rules is a (hyper-)tree is equal to CFT_{sp}, the context-free tree grammars which are simple in the parameters, ie. without copying or deletion of trees, which is a different way of stipulating the linear and non-erasing constraint on linear context-free rewrite systems (LCFRS) (Vijay-Shanker et al., 1987).

Weir (1992) shows that *string generating* hyperedge replacement grammars generate the same languages as LCFRS and multi-component tree adjoining grammars (MCTAGs). All this suggests a possible extension of the current results relating tree generating HR grammars of rank > 2 to MCTAGs and LCFRS.

With respect to **NL**$\diamond_{\mathcal{R}}$, it seems possible to extend the current results, increasing the number of modes to generate richer classes of languages, possibly the same classes of languages as those generated by LCFRS and MCTAGs. For **LG**, such extensions seem less evident. Indeed, an appealing property of **LG** is that we do not need different modes, but if we are willing to add different modes to **LG** then extensions of the classes of languages generated seem possible.

An interesting consequence of the translations proposed here is that they open the way for new parsing algorithms of Lambek grammars. In addition, compared to earlier work like that of Moortgat and Oehrle (1994), they give radically new ways of implementing phenomena like Dutch verb

[4]LOGCFL is the complexity class of of problems which are log-space reducible to the decision problems for context-free grammars. Vijay-Shanker et al., (1987) show that linear context-free rewrite systems and multicomponent TAGs are also in this complexity class.

clusters in $\mathbf{NL}\Diamond_{\mathcal{R}}$.

6 Conclusions

$\mathbf{NL}\Diamond_{\mathcal{R}}$ and **LG** are mildly context-sensitive formalisms and therefore benefit from the pleasant properties this entails, such as polynomial parsability. TAGs and HR grammars, because of the simplicity of their basic operations, have played a central role in establishing this correspondence.

References

Capelletti, Matteo. 2007. *Parsing with Structure-Preserving Categorial Grammars*. Ph.D. thesis, Utrecht Institute of Linguistics OTS.

de Groote, Philippe. 1999. The non-associative Lambek calculus with product in polynomial time. In Murray, N. V., editor, *Automated Reasoning With Analytic Tableaux and Related Methods*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 128–139. Springer.

Drewes, Frank, Annegret Habel, and Hans-Joerg Kreowski. 1997. Hyperedge replacement graph grammars. In Rozenberg, Grzegorz, editor, *Handbook of Graph Grammars and Computing by Graph Transformations*, volume I, pages 95–162. World Scientific.

Engelfriet, Joost and Sebastian Maneth. 2000. Tree languages generated by context-free graph grammars. In *Theory and Applications of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 15–29. Springer.

Engelfriet, Joost. 1997. Context-free graph grammars. In Rosenberg, Grzegorz and Arto Salomaa, editors, *Handbook of Formal Languages 3: Beyond Words*, pages 125–213. Springer, New York.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Massachussets.

Joshi, Aravind and Seth Kulick. 1997. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 20(6):637–667.

Joshi, Aravind and Yves Schabes. 1997. Tree-adjoining grammars. In Rosenberg, Grzegorz and Arto Salomaa, editors, *Handbook of Formal Languages 3: Beyond Words*, pages 69–123. Springer, New York.

Joshi, Aravind, Vijay Shanker, and David Weir. 1991. The convergence of mildly context-sensitive grammar formalisms. In Sells, Peter, Stuart Shieber, and Thomas Wasow, editors, *Foundational Issues in Natural Language Processing*, pages 31–82. MIT Press, Cambridge, Massachusetts.

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

Lambek, Joachim. 1961. On the calculus of syntactic types. In Jacobson, R., editor, *Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics*, volume XII, pages 166–178. American Mathematical Society.

Lautemann, Clemens. 1990. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27(5):399–421.

Moortgat, Michael and Richard T. Oehrle. 1994. Adjacency, dependency and order. In *Proceedings 9th Amsterdam Colloquium*, pages 447–466.

Moortgat, Michael. 1997. Categorial type logics. In van Benthem, Johan and Alice ter Meulen, editors, *Handbook of Logic and Language*, chapter 2, pages 93–177. Elsevier/MIT Press.

Moortgat, Michael. 2007. Symmetries in natural language syntax and semantics: the Lambek-Grishin calculus. In *Proceedings of WoLLIC 2007*, volume 4567 of *LNCS*, pages 264–284. Springer.

Moot, Richard and Quintijn Puite. 2002. Proof nets for the multimodal Lambek calculus. *Studia Logica*, 71(3):415–442.

Moot, Richard. 2007. Proof nets for display logic. Technical report, CNRS and INRIA Futurs.

Pentus, Mati. 1995. Lambek grammars are context free. In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Montreal, Canada.

Pentus, Mati. 2006. Lambek calculus is NP-complete. *Theoretical Computer Science*, 357(1):186–201.

Vijay-Shanker, K. and David Weir. 1994. The equivalence of four extensions of context free grammars. *Mathematical Systems Theory*, 27(6):511–546.

Vijay-Shanker, K., David Weir, and Aravind Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, California. Association for Computational Linguistics.

Weir, David. 1992. Linear context-free rewriting systems and deterministic tree-walking transducers. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Morristown, New Jersey. Association for Computational Linguistics.