Filtering Axiom Links for Proof Nets

Richard Moot

Filtering Axiom Links for Proof Nets

Richard Moot
LaBRI (CNRS), INRIA Bordeaux SW, University of Bordeaux
Draft

September 3, 2008

Contents

ABSTRACT

An important problem for proving statements in multimodal categorial grammar is that even when using proof nets — which improve upon proof search in natural deduction and sequent calculus by identifying all equivalent proofs — the number of possible axiom links to consider is still enormous. We will propose several efficient strategies to reduce the number of axiom links and will evaluate the resulting combined strategy against a large number of statements in both multimodal and Lambek categorial grammars and find that we eliminate a very large number of the axiom links which do not correspond to any proof net.

Chapter 1

Introduction

The multimodal Lambek calculus (Moortgat, 1997) is a powerful and flexible grammar framework. Unfortunately, it has some sublogics — like the associative Lambek calculus L or the associative, commutative Lambek-Van Benthem calculus LP — which are known to be NP complete (Pentus, 2006, Kanovich, 1991).

In this article we look at proof nets for the multimodal Lambek calculus and investigate ways to reduce the number of axiom links we need to perform in order to decide whether a statement $\Gamma \vdash C$ is derivable or not.

We then evaluate the effect of these reductions on a large number of randomly generated Lambek calculus sequents as well as on a large number of sequents which have been extracted from a corpus and find that the combined filtering strategies filter our the large majority of the incorrect axiom links, that is, those axiom links which are not part of any proof net, showing that — in spite of the theoretical complexity — in practice we can parse Lambek grammar while making just a small percentage of incorrect axiom links.

The rest of this report is structured as follows.

Chapter 2 gives a short introduction to parsing statements of the multimodal Lambek calculus using proof nets. It discusses proof nets for $\mathbf{NL}\Diamond_{\mathcal{R}}$, which is the non-associative Lambek calculus **NL** extended with unary operators and a set of structural rules $\mathcal{R}$. Several instances of these structural rules as well as their applications are shown. The chapter concludes with an overview of some of the combinatorics of proof nets.

Chapter 3 discusses four strategies for reducing the total number of axioms links. The first two: acyclicity and connectedness (Section 3.1) and first-order approximation (Section 3.2) are fairly well-known and I touch upon them only briefly. The final two strategies are new. We show how to compute the relations between pairs of unary connectives as well as verify the possibility of binary and unary contractions in the presence of some frequently used sets of structural

rules using a context-free grammar in Section 3.3 and how to eliminate axiom links which cannot be part of a *total* linking using methods from constraint logic programming in Section 3.4.

Chapter 4 evaluates the combined algorithm against two sets of derivable sequents. The first set is a set of randomly generated derivable sequents in the Lambek calculus **L**. The second set is a set of multimodal sequents which has been automatically extracted from a corpus. In both cases, the algorithms proposed will filter out a very important number of underivable sequents.

The final chapter concludes and gives some possiblities for future research.

Chapter 2

Proof nets

We follow Moot and Puite (2002) for our presentation of proof nets in this section.

Constructing a proof structure for a statement in the multimodal Lambek calculus is done in three phases:

1. decompose the lexical formulas as well as the goal formula,
2. connect positive and negative atomic formulas,
3. decide whether the resulting proof structure is a proof net, using a correctness criterion.

We look at each of the three phases in turn.

2.1 Decomposition

Decomposing the lexical formulas is deterministic and linear in the number of connectives in the statement. Depending on whether a complex formula is a hypothesis (antecedent formula) or a conclusion (succedent formula), only one link can apply: a left link for a hypothesis and a right link for a conclusion. We can simply descend the formula tree until we reach the leaves, which are the atomic formulas.

The full set of links is shown in Table 2.1. There are two links for every connective: a left link for when it occurs as a hypothesis (portrayed at the bottom; antecedents formulas start their unfolding here) and a right link for when it occurs as a conclusion (portrayed at the top; the goal formula starts its unfolding here).

The left link for $B\backslash_i A$ is simply the modus ponens rule, saying it combines with a B on its left to form an A. The right link — for $B\backslash_i A$ as a conclusion

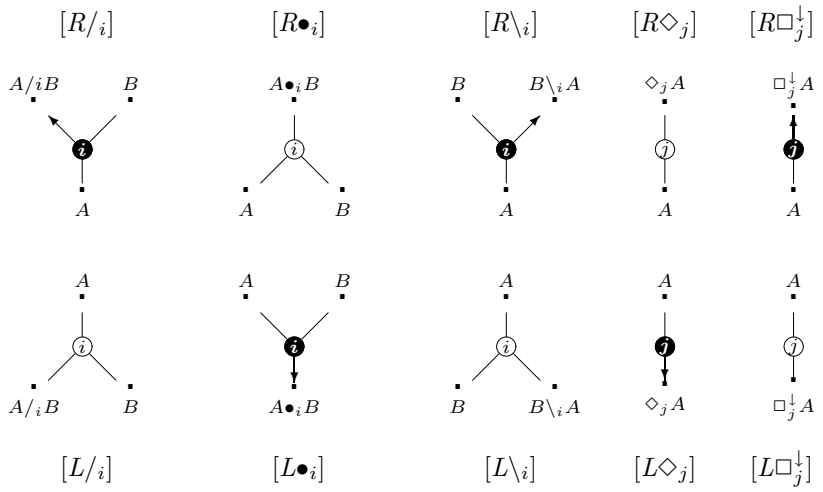

Table 2.1: Logical links for the multimodal Lambek calculus

— is its exact opposite: it allows us to use a B hypothesis to prove an A. Of course, we'll need to check that the B occurs as the leftmost sister of the A node and this is where the contractions will come in later.

Figure 2.1 shows an example unfolding of the sequent

$$np, (np\backslash s)/np, ((np\backslash s)/np)\backslash(np\backslash s) \vdash s$$

which would be a lexical lookup for a sentence like 'Robin wet himself'. The numbers on the atomic formulas are not a part of the proof structure, they serve only to help us refer to the different formulas later.

This is a slightly simplified version of the abstract proof structures of Moot and Puite (2002). We have suppressed as much information as possible: the information on the internal nodes, the lexical formula (both of them can be uniquely reconstructed from the unfolding) and the premiss/conclusion distinction of the individual nodes in Figure 2.1 as well (since we need it only when there is just a single antecedent formula).

2.2 Axioms

The second stage of constructing a proof net consists of identifying the axiomatic formulas: we select a positive and a negative atomic formula of the same type and identify their vertices, thereby connecting different parts of the graph. For this second step, there are potentially many solutions and the main part of this paper focusses on strategies for reducing these possibilities as much as possible.

Figure 2.1 shows the possibilities for the axioms links by portraying them in a grid: every row represents the axiom link possibilities for a negative formula,

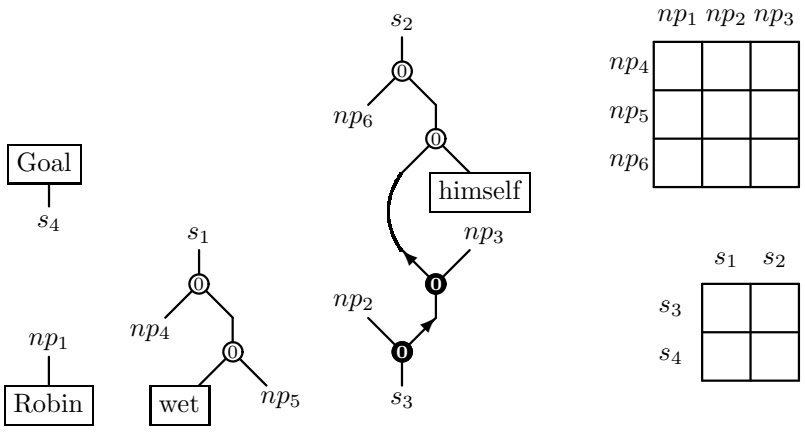

Figure 2.1: Formula unfolding and axiom link possibilities for $np, (np\backslash s)/np, ((np\backslash s)/np)\backslash(np\backslash s) \vdash s$

whereas every column represents the axiom link possibilities for a positive formula. A full axiom linking corresponds to putting exactly one mark in every row and column. In graph theory, this is usually called a *perfect matching*.

Since there are $n!$ different perfect matchings possible, the goal of this paper is to exclude the maximum number of these connections when they cannot be part of a proof net.

2.3 Contractions and Structural Rules

The final step is checking if the resulting graph is a proof net, by contracting all par links, drawn with the black center, as shown in Table 2.2. When all par links have been contracted, the result will be a tree — a *tensor tree* — with the lexical entries used as its leaves.

In case it is impossible to contract a par link, the proof structure we are dealing with is not a proof net and therefore, we know there is no proof of the corresponding sequent.

A typical multimodal categorial grammar has a set of *structural rules* $\mathcal{R}$ in addition to these contractions. Structural rules allow us the replace a substructure of an abstract proof structure which is a tensor tree t_1 by another tensor tree t_2. Since we operate in a fragment of multiplicative linear logic, we require that t_1 and t_2 share the same leaves. In other words, deletion and copying of subtrees isn't allowed, though we can change the order of the different subtrees. It is this reordering which permits us to push the languages generated by multimodal categorial grammars beyond the context-free languages.

Table 2.3 and 2.4 show the frequently used structural rules of mixed asso-

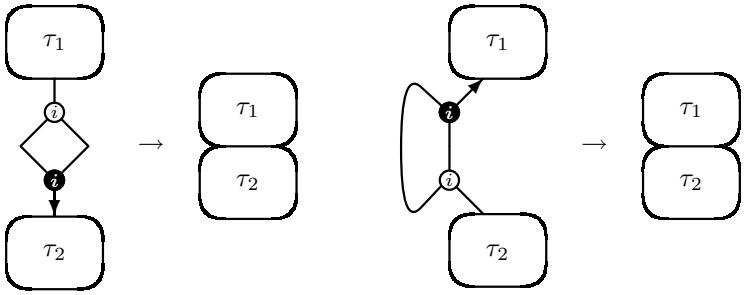

Table 2.2: Graph contractions

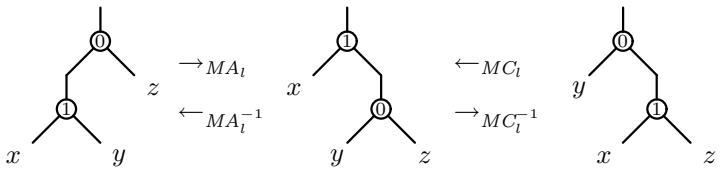

Table 2.3: Mixed associativity and commutativity — left branch

ciativity and mixed commutativity of modes 0 and 1. Observe that in Table 2.3 the leaf x always stays on the left branch of a mode 1 structure, whereas in Table 2.4 the leaf z always stays on the right branch of a mode 1 structure. Moortgat and Oehrle (1994) use the $[MA_l]$ and $[MC_l]$ rules to give an account of word order in Dutch embedded clauses.

A variation of these rules is shown in Table 2.5 and 2.6. In these cases, there is only a single unary mode 0 but a *unary* branch 0 indicates that an x leaf can move from one left branch to another (as in Table 2.5) or that a z leaf can move from one right branch to another (as in Table 2.6). Moortgat (1999) shows how the right branch extraction rules $[MA\Diamond_r]$ and $[MC\Diamond_r]$ of Table 2.6 give an account of English relativization, whereas the left branch extraction

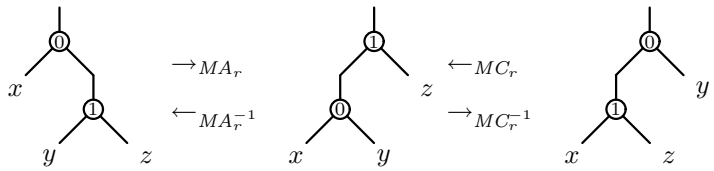

Table 2.4: Mixed associativity and commutativity — right branch

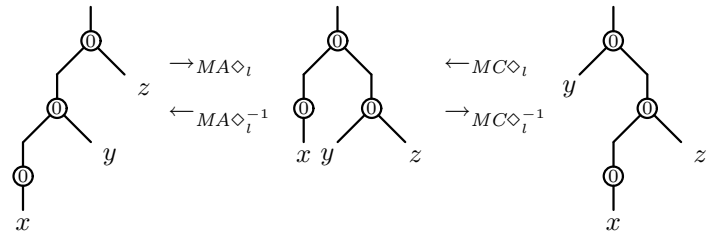

Table 2.5: Mixed associativity and commutativity — left branch with unary control

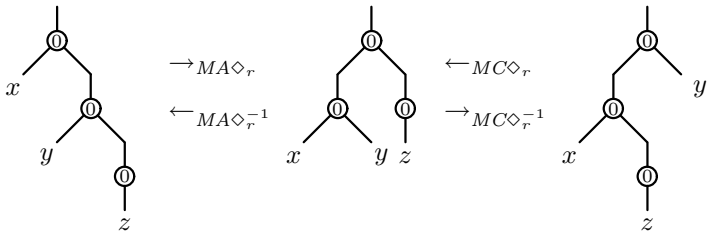

Table 2.6: Mixed associativity and commutativity — right branch with unary control

rules $[MA\diamond_l]$ and $[MC\diamond_l]$ of Table 2.5 can be used to give an account of Dutch relativization.

A final set of structural rules involves the communication between binary and unary modes.

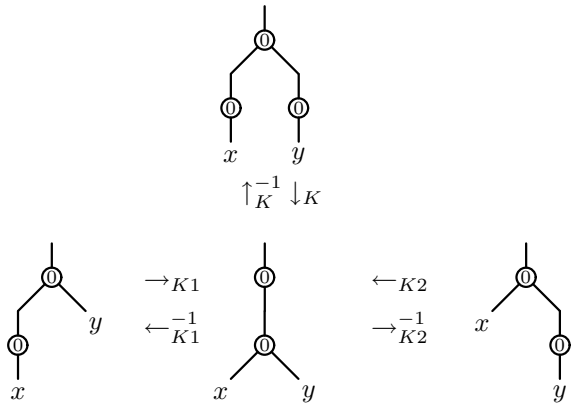

Table 2.7: Communication between unary and binary modes - K, K1 and K2

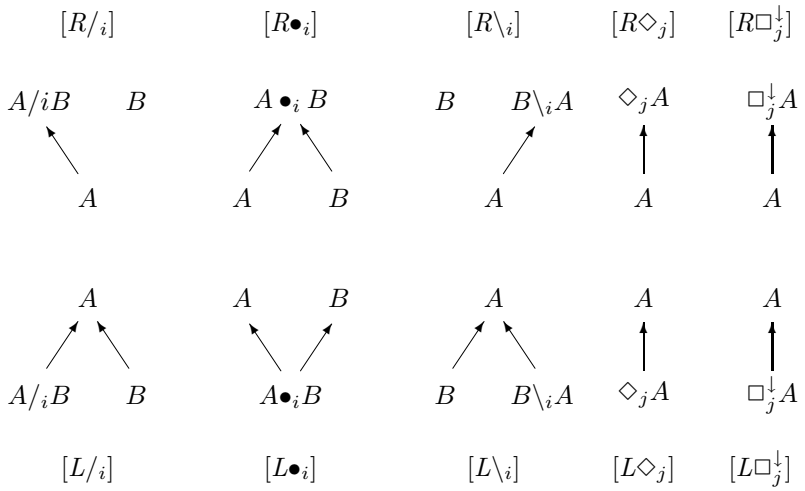

Table 2.8: Logical links for essential nets

2.4 Essential Nets

For the acyclicity and connectedness criterion discussed in Section 3.1 as well as for the context-free grammar criterion of Section 3.3, we are interested in an alternative correctness criterion proposed by Lamarche (1994). This criterion is based on a different way of decomposing a sequent, this time into a directed graph, with conditions on the paths performing the role of a correctness criterion. A net like this is called an *essential net*. The links for essential nets are shown in Table 2.8, though we follow de Groote (1999) in reversing the arrows of Lamarche (1994).

In addition — to keep the essential nets as close to the abstract proof structures of before — there are no axiom links and cut links and the main formula of a link can be its hypothesis as well as its conclusion.

As a consequence of this change of perspective, all edges point upwards.

Definition 2.1 *Given an essential net $\mathcal{E}$ its conclusion is called the* output *of the essential net and the hypotheses, as well as the B premisses of any $[R/_i]$ or $[R\backslash_i]$ link, are called its* inputs. *When we need to distinguish between these two types of inputs, we will call the hypotheses the* lexical inputs *and the B premisses of the $[R/_i]$ and $[R\backslash_i]$ the* auxiliary inputs.

Definition 2.2 *An essential net is* correct *iff the following properties hold.*

1. *it is acyclic,*
2. *every path from the B premiss of a $[R/_i]$ or $[R\backslash_i]$ link passes through the complex premiss of this link,*

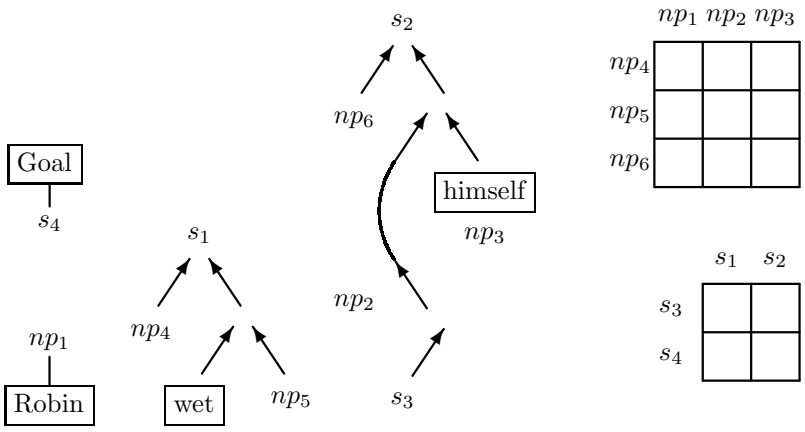

Figure 2.2: The essential net corresponding to Figure 2.1

3. every path from the inputs of the graph ends at the output of the graph.

Condition (1) reflects the acyclicity condition on proof nets, whereas conditions (2) and (3) reflect the connectedness condition. The formulation of 'every path' exists only to ensure correctness of the negative • link; in all other cases there is at most one path between two points in a correct essential net.

Theorem 2.3 (Lamarche (1994)) *A sequent $\Gamma \vdash C$ is provable in **MILL** iff its essential net is correct.*

To give an idea of why these three properties need to hold, given that we need to contract to a tensor tree with the hypotheses as its leaves and the conclusion as its root, we remark the following:

2.5 Basic Properties

In order to better analyze the properties of the algorithms we propose, we will first take a look at some basic properties of proof nets.

2.5.1 Axiom Links

Since we will be concerned with finding an axiom linking for an abstract proof structure $\mathcal{A}$ which will allow a contraction of $\mathcal{A}$ into a tensor tree, we first given some bounds on the number of proof structures we will have to consider. Given that the problem we are trying to solve is NP complete, even in the non-commutative case, it is not surprising that these bounds are quite high.

Proposition 2.4 *Let $\mathcal{P}$ be a proof net and f an atomic formula, then the number of positive occurrences of f is equal to the number of negative occurrences of f.*

This proposition follows immediately from the fact that every atomic formula has one occurrence as a negative hypothesis (either of the proof net or of a link) and one occurrence as a positive conclusion (again, either of the proof net or of a link).

Proposition 2.5 *Every formula unfolding $\mathcal{F}$ has $O(a!)$ possible identifications of atomic formulas which produce a proof structure, where a is the maximum number of positive and negative occurrences of an atomic formula in $\mathcal{F}$.*

If we have a positive atomic formulas, we have a possibilities for the first one, since all negative formulas may be selected, followed by $a-1$ for the second etc. giving us $a!$ possibilities.

Proposition 2.6 *Every formula unfolding $\mathcal{F}$ has $O(4^a)$ planar axiom linkings which produce a proof structure, where a is the maximum number of positive and negative occurrences of an atomic formula in $\mathcal{F}$.*

This follows from the fact that a planar axiom linking is simply a binary bracketing of the atomic formulas and the fact that there are C_{a-1} such bracketings, where C_k, the kth Catalan number, approaches $4^k/\sqrt{\pi}k^{3/2}$.

Proposition 2.7 *For every partial proof structure with a atomic formulas which are not yet identified with an atomic formula of opposite polarity there are $O(a^2)$ possible axiom links.*

Given that every positive atomic formula can be linked to every negative atomic formula of the same atomic type this gives us a^2 pairs.

2.5.2 Graph Size

Proposition 2.8 *For every proof structure $\mathcal{S}$ with h hypotheses, 1 conclusion, p binary par links and t binary tensor links, the following equation holds.*

$$p + h = t + 1 = a$$

Given Proposition 2.4, the number of positive and negative atomic formulas is both a. Suppose we want to construct a proof structure $\mathcal{S}$ with h negative conclusions and 1 positive conclusion from these atomic formulas. When we look at the links in Table 2.1 we see that all par links reduce the number of hypotheses by 1 and all tensor links reduce the number of conclusions by 1.

Proposition 2.9 *Every essential net $\mathcal{E}$ has $v = h+1+2(p+t) = O(a)$ vertices and $2t + p \leq e \leq 2(t + p) + a = O(a)$ edges.*

This follows immediately from inspection of the links: all h hypotheses and the single conclusion of the essential net start out as a single vertex and every link adds two new vertices. For the edges: the minimum number is obtained when we have no axiom links and all par links are positive links for $\backslash$ or $/$ which introduce one edge, the maximum number includes a axiom links and par links which are all negative links for $\bullet$.

Chapter 3

Filtering Axiom Links

Simple combinatorics shows that there are $n!$ possible axiom linkings for $2n$ atomic formulas (Proposition 2.5). In Figure 2.1, there are therefore 2 possibilities for s and 6 possibilities for np. This makes exhaustive search prohibitive for all but the most trivial statements.

However, there are several possibilities to rule out axiom links which can never contribute to a contractible proof structure. We discuss some known and some new strategies in the following sections.

3.1 Acyclicity and Connectedness

Danos and Regnier (1989) introduce the acyclicity and connectedness criterion for proof nets of multiplicative linear logic. Give that the categorial logics we work with are all sublogics of multiplicative linear logic (in the sense that any derivable sequent in multimodal categorial grammar has a derivable image in multiplicative linear logic) we can use the fact that anything underivable in MLL is underivable in categorial grammar as well.

Moot (2004) shows how an adaptation of the Floyd-Warshall algorithm can be used to select from the total set of possible axiom links those that produce acyclic and connected proof structures.

The Floyd-Warshall algorithm computes the transitive closure of a graph by successively eliminating the intermediate vertices c from every path from a to b. Given a vertex c and the paths $a \rightarrow c \rightarrow b$ for all a and b we create a direct path $a \rightarrow b$ if it didn't exist before. That is to say, there is a path from a to b if either there is a path from a to c and from c to b or if there is a path from a to b which we already knew about (Figure 3.1 on the following page).

$$path(a, b) := path(a, b) \vee (path(a, c) \wedge path(c, b)) \tag{3.1}$$

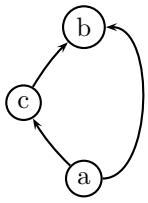

Figure 3.1: Eliminating node c from the path from a to b

After eliminating c, for every path in the original graph which passed through c there is now a shortcut which bypasses c. After we have created such shortcuts for all vertices in the graph it is clear that the resulting graph has an edge $a \to b$ iff there is a path from a to b in the original graph.

The acyclicity condition, condition (1) from Definition 2.2, is easily verified.

Verifying conditions (2) and (3) from Definition 2.2 is a bit harder. The question we want to ask about each link is: does this link contribute to a connected proof structure? Or, inversely, does excluding the other possibilities for the two atomic formulas we connect mean a connected proof structure is still possible.

To check the conditions we need to verify the following:

1. for every negative input of the net we verify there exists a path to the positive conclusion,

2. for every negative $\bullet$ link we verify that *both* paths leaving from it reach their destination,

3. for every positive / or \ link we check the existence of a path from its negative premiss to its positive conclusion continuing to the positive conclusion of the essential net.

Given that we are already computing the transitive closure of the graph for verifying acyclicity, we can exploit this by adding additional information to the matrix we use for the transitive closure. There are many ways of storing this extra information, the simplest being in the form of an ordered list of pairs. Given that, for a atomic formulas, each possible connection allows $(a-1)^2$ other connections (ie. it is agnostic about all possibilities not contradicting this one) but excludes $2(a-1)$ possibilities. It is therefore more economic to store the connections which are excluded. For example, the ordered set associated to the edge from 1 to 4 will be $\{1-5, 1-6, 2-4, 3-4\}$, meaning "there is an edge from 1 to 4 but not to anywhere else and the only edge arriving at 4 comes from 1".

Note that in the description of the Floyd-Warshall algorithm, we made use only of the logical 'and' and 'or' operators. For ordered sets, the corresponding

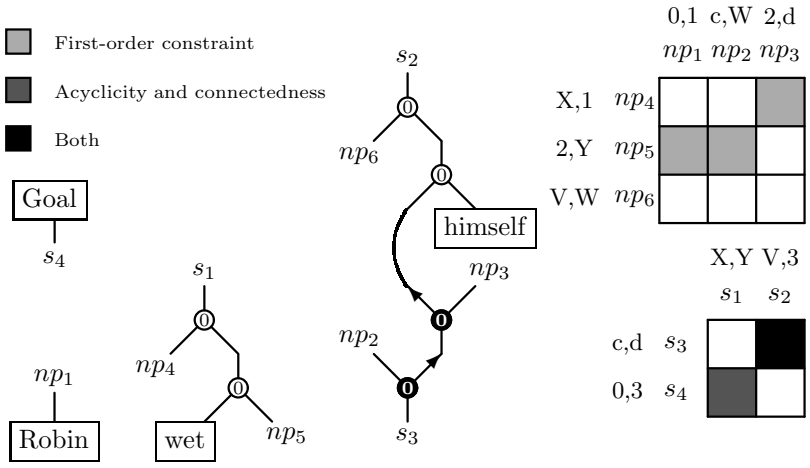

Figure 3.2: Links excluded by the acyclicity and connectedness condition

operations are set union and set intersection. For eliminating vertex c from a path from a to b, we first take the union of the ordered set representing the links which are not in a path from a to c with that representing the links not in a path from c to b (any vertex in either set couldn't be in a path from a via c to b). Then, we take the intersection of this set with the old set associated to the path from a to b.

$$\overline{path}(a,b) := \overline{path}(a,b) \cap (\overline{path}(a,c) \cup \overline{path}(c,b)) \tag{3.2}$$

Note that Equation 3.2 is simply Equation 3.1 with both sides negated, the negations moved inward and set union and intersection in the place of the logical 'or' and 'and' operators.

Given that we can implement the union and intersection operations in linear time with respect to the size of the input sets, the total complexity of our algorithm becomes $O(v^3 2(a-1)) = O(a^4)$.

Figure 3.2 shows, in dark gray and black, the links which are excluded when we use this condition. We remark that this leaves just one possibility for linking the s formulas.

3.2 First-order Approximation and Word Order

Even though the acyclicity and connectedness check is an effective test, it is based on the 'worst case' scenario of a fully associative and commutative logic. A second strategy for removing axiom links which cannot contribute to constructing a proof net is to use first-order approximation to take constraints on word order into account. This has been used at least since LLoré and Morrill (1995) (though in a slightly different context).

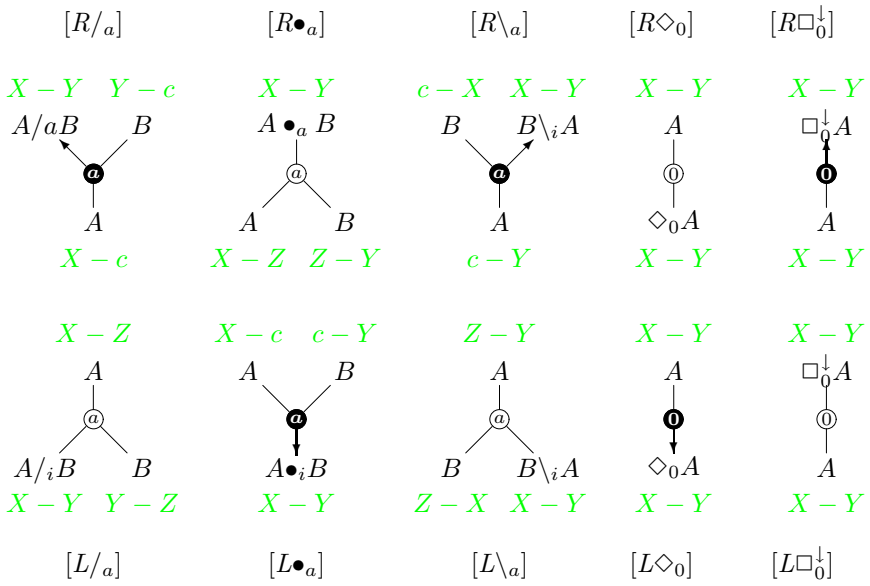

Table 3.1: First-order labelling for Lambek calculus formulas

Moot and Piazza (2001) propose an embedding of the Lambek calculus using first-order quantifiers. Each formula is assigned a *pair* of string positions which correspond to the left and right frontier of the string to which the formula corresponds. Lexical formulas are assigned successive string positions: $0-1$ for the first word, $1-2$ for the second up until $(n-1)-n$ for the last word of a sequence of n. The goal formula is assigned $0-n$. Table 3.1 shows the propagation of first-order variables to the other formulas. In every case a c corresponds to a fresh constant and a Z to a fresh variable.

Figure 3.2 shows how we can use these propagation rules to compute the first-order labels assigned to the atomic formulas of our example sequent and — in light grey and black — the axiom links which are excluded using the first order constraints for L, which is justified given that we have a sequent which is derivable even in the non-associative Lambek calculus. In the figure, the numbers $0, 1, \ldots$ correspond to constants referring to string positions, lower-case letters $c, d, \ldots$ to constants introduced by the par links and upper-case letters $X, Y, \ldots$ correspond to variables.

As long as we make sure that the structural rules permit a subset of the word order possibilities allowed by the first order variables, this strategy is correct. Given that the sequent is derivable in **NL**, we are justified in using the **L** first-order labelling.

In addition to embedding the Lambek calculus, Moot and Piazza (2001) show how several linguistic phenomena like quantifier scope ambiguities, *wh*

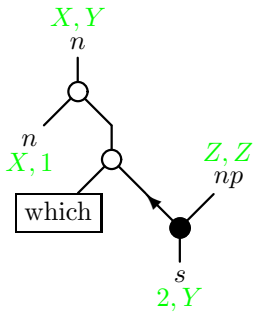

Figure 3.3: First order variables for 'which'

extraction and island constraints — for which there is no satisfactory treatment in the Lambek calculus — can be analysed using first-order quantifiers as well.

By assigning slightly more subtle first-order variables and constants, we can incorporate these analyses into our labels. An example for the treatment of extraction is shown in Figure 3.3. Here, we simply use the solution of Moot and Piazza (2001) for *wh* words. The first-order variables indicate that the word 'which', when it occurs between string positions 1 and 2, is looking for an n starting at some position X directly to its left and an s ending at position Y directly to its right. Inside this s we can use an np which can take up any position. The final result will then be an n between X (the leftmost position of the n argument) and Y (the rightmost position of the s argument). Note that we still need to add the appropriate structural rules to our grammar in order to derive medial extraction cases (see Moortgat, 1997, for a solution).

Given that we can check the first order constraints simply by unification of variables and constants, meaning $O(1)$ per cell in the axiom matrix, the total complexity of the first-order constraints is $O(n^2)$ for $2n$ atomic formulas.

3.3 Context-Free Grammars

3.3.1 The Unary Connectives

The unary connectives are a powerful addition to the multimodal Lambek calculus. They can be used to license structural rules but the relations between the logical formulas they induce can be used to encode linguistic features, as done, for example by Heylen (1999), or to restrict scope possibilities, as done, for example by Bernardi and Moot (2003) for the interaction between generalised quantifiers and negation in English, by Moot and Retoré (2006) for generalised object clitics in French and by Bernardi and Szabolsci (2007) for operators in Hungarian.

In certain cases, we use the unary modalities just for the derivability relations between the different types. Figure 3.4 summarizes the different relations between unary prefixes of up to four. Note that $\Diamond\Box\Diamond\Box A$ and $\Box\Diamond\Box\Diamond A$ are not

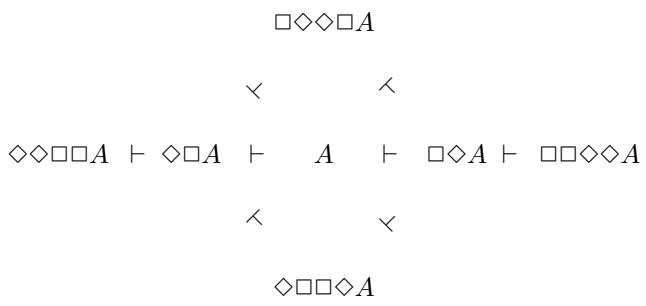

Figure 3.4: Derivability relations using the unary logical rules

displayed, given that they are equivalent to $\Diamond\Box A$ and $\Box\Diamond A$ respectively[1].

By extending the unary prefixes further we can generate an intricate hierarchy of formulas, but for many applications the seven formula recipes in Figure 3.4 suffice. However, even for the formulas we've shown here it's not directly clear which pairs of them are derivable.

Fortunately, there is a simply way to check the contractibility of a sequence of unary formulas. First, we convert this sequence to a string as follows (with ϵ being the empty string and . the concatenation operation)

Definition 3.1 *Let A and B be two formulas which are the identical up to their unary prefixes. $\sigma(A \vdash B)$, the* string corresponding to $A \vdash B$*, is defined as $\|A\|^-.\|B\|^+$, where the positive and negative formula are translated as follows.*

$$\begin{array}{llll} \|\Diamond B\|^+ = \|B\|^+.m & & \|\Diamond A\|^- = l.\|A\|^- \\ \|\Box B\|^+ = \|B\|^+.r & & \|\Box A\|^- = m.\|A\|^- \\ \|B\|^+ = \epsilon \textit{ otherwise} & & \|A\|^- = \epsilon \textit{ otherwise} \end{array}$$

The easiest way to see the correspondence between a sequence of unary modes and a string is by turning a page with a formula unfolding 90 degrees to the right and realizing that every arrow pointing left will produce an l, while every arrow pointing right will produce an r.

Now, given that we have produced a string corresponding to the two sequences of unary connectives, we can check contractibility of these unary modes using the following context free grammar.

$$\begin{array}{lll} \mathrm{S} \rightarrow \epsilon & & (1) \\ \quad \mid \mathrm{l\ S\ m\ S} & & (2) \\ \quad \mid \mathrm{m\ S\ r\ S} & & (3) \end{array}$$

Rule (1) corresponds to the fact that it is possible not to have any unary connectives in front of a formula at all. Rule (2) corresponds to the $\Diamond$ contraction and rule (3) to the $\Box$ contraction.

[1] Technically the graph of Figure 3.4 is the transitive reduction of the derivability relation where all equivalent formulas (which would correspond to cycles in the unreduced graph) have been replaced by their smallest element.

Proposition 3.2 *$A \vdash B$ contracts to a single vertex using the unary contractions iff $S \to \sigma(A \vdash B)$.*

Proof (sketch) $\Rightarrow$ Induction on the number of contractions c. If $c = 0$ we use rule (1). If $c > 0$ there we look at the point where the *first* link is contracted. In order for this contraction to be valid the links between the two contracted links need to contract to a single node and in order for the entire sequence to contract everything after the second link needs to contract to a single node as well. Induction hypothesis allows us to combine these smaller proof nets using either rule (2) or (3).

$\Leftarrow$ Induction on the length of the CFG derivation. □

To give an illustration of how to use the context free grammar, we show how the derivable sequent $\Diamond\Box\Box\Diamond A \vdash \Box\Diamond A$ translates to *lmmlmr*, which we can derive as shown below.

S $\to$ l S m S
$\to$ l m S
$\to$ l m m S r S
$\to$ l m m S r
$\to$ l m m l S m S r
$\to$ l m m l m S r
$\to$ l m m l m r

We can show the inverse statement $\Box\Diamond A \vdash \Diamond\Box\Box\Diamond A$ which would correspond to *mlmrrm* is underivable simply because we cannot match the final m: there is no r to its right and if we would match it to the single l we would need to derive *mrr*, the symbols in between, but this is impossible given that it has an odd number of symbols.

The context free grammar is easily extended to the multimodal case, simply by adding different symbols l_i, m_i and r_i to the grammar and adding two new grammar rules for each of the new symbols.

There are limitations to using this system, however. First of all, it requires us to remove all unary branches from the final tree, though we could overcome this limitation in the same way we do for the binary connectives in the next section, though this means adding extra rules and non-terminal symbols to our grammar.

Secondly, we cannot use this strategy if some of the structural rules for the unary modes we're interested in are incompatible with the formula to string translation. Examples would be any inclusion rules between unary modes (though adding grammar rules would again be an option here) or structural rules which move unary modes up or down the tree, like the K, K1 and K2 structural rules of Moortgat (1997), showing in Figure2.7, which would require changing the translation function. However, as we will explore in Secion 3.3.3, there are cases in which it is possible to incorporate the K1 and K2 postulates into the context-free rules for a combined unary-binary CFG.

Given that parsing a context free grammar is $O(n^3)$ and we would have to perform this calculation for all n^2 possible axiom links, the total $O(n^5)$ com-

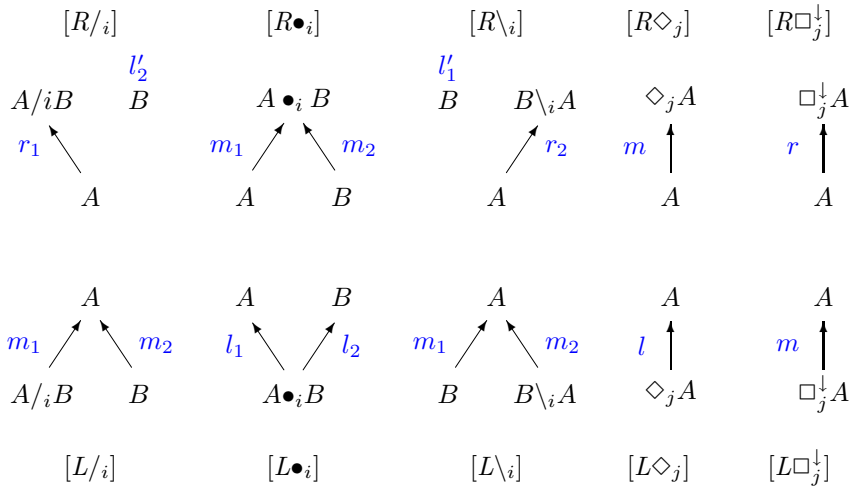

Table 3.2: Calculating the strings corresponding to paths in the essential nets

plexity is somewhat high. So for grammars which use the unary connectives extensively, it can be beneficial to pre-compute the relations between all sequences of unary connectives occurring in the grammar, after which we can do a simple table lookup to see if contraction is possible. This would reduce the total complexity to $O(n^2)$.

3.3.2 Adding the binary connectives

The strategy for the unary connectives extends easily to the binary case. This requires us to look at all paths in the graph. It is most easily seen using the essential nets of Section 2.4. Table 3.2 shows how to calculate the string corresponding to a path in an essential net. The treatment of the unary connectives is unchanged, the only additions are the symbols for the binary connectives. Remark that the paths starting at the B premisses of the $[R/_i]$ and $[R\backslash_i]$ links start with a symbol which does not correspond to an edge in the directed graph, but that otherwise every symbol corresponds to an edge.

Definition 3.3 *Let $\mathcal{E}$ be an essential net which is correct in the sense of Definition 2.2. The set of* paths *P of $\mathcal{E}$ is the smallest set satisfying.*

- *if i is a lexical input of $\mathcal{E}$ and p is a directed path from i to the output o then $p \in P$.*
- *if i is an auxiliary input of $\mathcal{E}$ and p is a directed path from i to the conclusion of the corresponding link then $p \in P$.*

The only subtlety in the definition of paths in essential nets is the case for the auxiliary inputs: we want this path to start at the B formula of the $[R/_i]$ and $[R\backslash_i]$ and end at the A formulas. Case 2 of Definition 2.2 guarantees there is a path from B to the complex formula. Given the orientation of the arrows, this path must pass through the A formula as well.

The number of different paths of an essential net $\mathcal{E}$ is $h + p$, where h is the number of hypotheses (lexical inputs) in the essential net and p the number of binary par links: the right rules for the implication each add a path because of Definition 3.3 and the left rule for product adds a path by splitting a single path into two.

Definition 3.4 *Let p be a path in a correct essential net $\mathcal{E}$ the* string $\sigma(p)$ corresponding to *p is obtained by traversing the path p from start to finish writing down each symbol corresponding the auxiliary input and each edge we pass.*

The grammar rules extend straightforwardly. Because for the binary modes we don't generally require the antecedent to be a single formula and new category V (for 'vertex' since strings of category V contract to a single fortex) has been introduced which corresponds to the category S in the previous section. The category S in the grammar below permits us to pass the m_1 and m_2 letters when necessary.

$\mathrm{S} \to \epsilon$	(0)	
$\mid$ V S	(1)	[*Contract*]
$\mid$ m_1 S	(2)	[*Left*]
$\mid$ m_2 S	(3)	[*Right*]
$\mathrm{V} \to \epsilon$	(4)	
$\mid$ l V m V	(5)	$[L\diamond]$
$\mid$ m V r V	(6)	$[R\square^{\downarrow}]$
$\mid$ l_1 V m_1 V	(7)	$[L\bullet]$
$\mid$ l_2 V m_2 V	(8)	$[L\bullet]$
$\mid$ m_1 V r_1 V	(9)	$[R/]$
$\mid$ l_2' V m_2 V	(10)	$[R/]$
$\mid$ l_1' V m_1 V	(11)	$[R\backslash]$
$\mid$ m_2 V r_2 V	(12)	$[R\backslash]$

Lemma 3.5 *$A \vdash B$ contracts to a single vertex using the contractions iff for every path p in the corresponding essential net $V \to \sigma(p)$.*

Lemma 3.6 *$\Gamma \vdash C$ contracts to a tensor tree using the contractions iff*

- *for every path p starting at a lexical hypothesis in the corresponding essential net $S \to \sigma(p)$.*
- *for every path p starting at an auxiliary hypothesis in the corresponding essential net $V \to \sigma(p)$.*

3.3.3 Adding structural rules

In some cases, we can go even further and include the structural rules in the context-free grammar.

If we calculate the path starting at x in the $K1$ rule on both the left hand side and the right hand side of the rule, we see it corresponds to replacing mm_1 by m_1m. This is a context-sensitive rule, however, and we would like to keep our filtering strategies polynomial. A typical use of the unary modes in combination with the K1 and K2 postulates is in the context of feature checking. In this case we know that the feature needs to be checked by a $[R\Box^\downarrow]$ rule later. If this is the case, we can — depending on which of the structural rules are available, K1, K2 or both — add one of the following rules to our context-free grammar.

$$\begin{array}{lll} \mathrm{S} \to m\ \mathrm{V}\ (m_1)^*\ \mathrm{V}\ r\ \mathrm{S} & (6a) & [R\Box^\downarrow + K1] \\ \mathrm{S} \to m\ \mathrm{V}\ (m_2)^*\ \mathrm{V}\ r\ \mathrm{S} & (6b) & [R\Box^\downarrow + K2] \\ \mathrm{S} \to m\ \mathrm{V}\ (m_1|m_2)^*\ \mathrm{V}\ r\ \mathrm{S} & (6c) & [R\Box^\downarrow + K1 + K2] \end{array}$$

Technically speaking, it is an abuse of notation to use the regular expression operators $*$ and $|$ in the context-free grammar. It would be more proper to add the regular language rules to the context-free grammar. I have chosen to use regular expression to give the most compact account possible of the generalised contractions as possible. For example, the rule $[R\Box^\downarrow + K1]$ says that in order to perform an $[R\Box^\downarrow]$ contraction, an arbitrary number of left braches can intervene between the unary tensor and the corresponding par.

A similar strategy is possible for the left and right extraction structural rules ($[MA\Diamond_l]$ and $MC\Diamond_l]$ of Table 2.5, and $[MA\Diamond_r]$ and $MC\Diamond_r]$ of Table 2.6 respectively) provided they only occur in the context of positive $A/\Diamond\Box^\downarrow B$ or $\Diamond\Box^\downarrow B\backslash A$ occurrences. Rules (5a) and (5b) below show the two cases.

$$\begin{array}{lll} \mathrm{V} \to l'_1\ l\ m\ m_1\ \mathrm{V}\ (m_1|m_2)^*\ \mathrm{V} & (5a) & [R\backslash + L\Diamond + MA\Diamond_l + MC\Diamond_l] \\ \mathrm{V} \to l'_2\ l\ m\ m_2\ \mathrm{V}\ (m_1|m_2)^*\ \mathrm{V} & (5b) & [R/ + L\Diamond + MA\Diamond_r + MC\Diamond_r] \end{array}$$

A limitation of the use of context-free grammars is that it can only be used with structural rules where a single path in the abstract proof structure can be used to decide if the structural rules have been applied correctly. For example, it can't be used to check the correct application of the K postulate given that we'd have to verify *both* branches to see if there is a unary branch below it.

3.4 Régin's Algorithm

Even with all the previous constraints on axiom links in place, we sometimes fail to exclude some axiom links which cannot belong to a total matching. This is in part because an axiom link is regarded more or less in isolation, meaning that we don't exploit the fact the we need to find a *total* matching.

Régin (1994) proposes an algorithm for the slightly more general problem of finding solutions for 'all different' constraints in constraint logic programming. His algorithm separates the possible axiom links (in our case) into three cate-

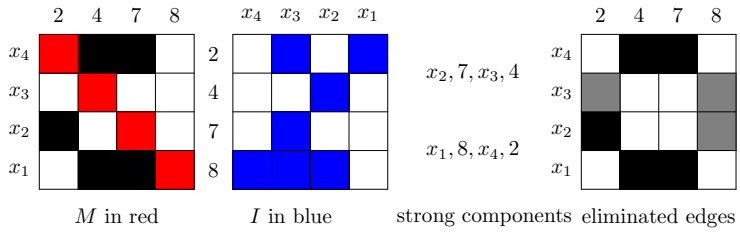

Figure 3.5: An illustration of Régin's algorithm

gories: those which must be a part of any linking, those who are only part of some linkings and those which do not belong to any linking.

Régin's algorithm works as follows:

1. find a total matching M of the graph, fail if none exists.

2. using the total matching M, compute the inverse I of the links outside of the matching $E - M$.

3. using edges which are alternatively in M and in I, compute the strong components.

4. when an edge E connectes two different strong components, do the following.

   - if $E \in M$, then E is part of every total linking and all alternatives for both vertices are deleted.
   - if $E \notin M$, then remove E from the graph, it cannot be part of any linking.

Régin's formulation includes an 'augmenting path' case to extend the matching, after which we repeat from step 2 with the new matching. However, we don't need this since we are guaranteed that if a matching exists it is necessarily total.

Figure 3.5 shows how Régin's algorithm operates. In the constraint problem, we have four variables x_1 to x_4 which can take four values 2, 4, 7 and 8. The squares marked in black on the left side of the picture are excluded because of the other constraints, whereas the red square are the total matching M computed in step 1 of the algorithm. In the middle, the inverse I of the edges outside of M — computed for step 2 of the algorithm — are shown in blue. Remark that these are just the white squares of the figure on the left mirrored at the diagonal. For step 3, we compute the strong components of the graph by alternating between red and blue edges. We have two strong components: $x_2, 7, x_3, 4$ and $x_1, 8, x_4, 2$. This means that the only values possible for x_2 and x_3 are 4 and 7, so we delete the edges between x_2 and 8, between x_3 and 2 and between x_3 and 8. The newly exlcuded edges are indicated in gray on the right of Figure 3.5.

Even though a smart axiom linking strategy (like always linking the atomic formula with the least possibilities first) would not benefit a lot from the reduction in this example, there are cases where it will exploit information like the absence of a total matching to fail directly.

The total complexity for Régin's algorithm is dominated by the cost of the total matching, which we can do at least in $O(n^{2.5})$ (Hopcroft and Karp, 1973). This makes the total cost of *all* filtering strategies $O(n^4)$ if the context-free grammars are precompiled and $O(n^5)$ if they are not.

Chapter 4

Evaluation

4.1 Random Sequents

In order to evaluate the combined filtering strategies, we have tested the axiom constraints on randomly generated derivable statements of the Lambek calculus. These statements have been generated using the inductive definition of Lambek calculus proof nets where all duplicate statements have been removed. To make the task as difficult as possible, only a single atomic formula a without unary prefixes has been used. Unfortunately, underivable statements don't have such an easy inductive characterization.

Figure 4.1 shows the amount of statements by the number of atomic formulas per sequent in the sample set, as well as the number of sequents by the number of connectives.

Out of 15.946 possible planar axiom links and 61.524 total possible axiom links, 2.546 correspond to different proofs. Therefore, there are 13.400 planar axioms links and 58.978 total axiom links which do not belong to any proof. Of these, the combined filtering algorithm excludes all but 279, for a total of 2.825 axiom links performed.

This means we eliminated 97.92% of the incorrect planar axiom links and 99.53% of the total number of incorrect axiom links. From the perspective of the *correct* axiom links, we perform only 10.96% more links than a 'perfect' linking strategy — which through some unknown method would only find links corresponding to proofs in an NP complete logic.

4.2 Extracted sequents

For the second evaluation we look at a subset of 5.454 sentences which have been extracted from the Spoken Dutch Corpus (see Moot, 2007, for details of the

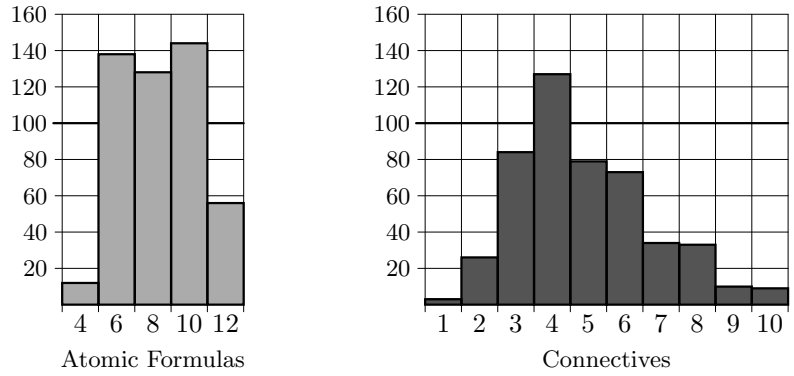

Figure 4.1: The distribution of the total number of atomic formulas and the total number of connectives in the randomly generated derivable statements

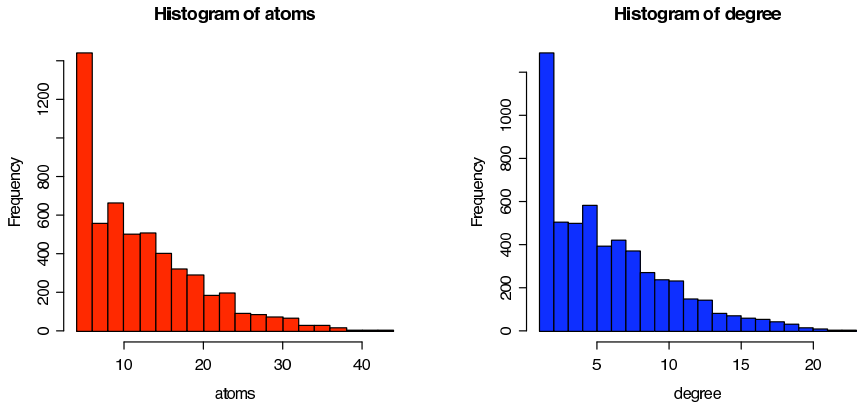

Figure 4.2: The distribution of the number of atomic formulas and the number of connectives of the extracted sequents

extraction procedure). Unlike the Lambek calculus sequents from before, there are discontinuous constituents as well as *wh* extractions present in this database. In addition, the sequents tend to have significantly more atomic formulas than the previous experiment: we have a total of 69.810 atomic formulas and 33.383 connectives (divided into 5.355 discontinuous connectives and 28.028 continuous connectives). Figure 4.2 shows the distribution. The median number of atoms is 12 whereas the mean is 12,80. The maximum number of atoms for a sequent in our set is 44. For the connectives, the median is 5, the mean 6,12 and the maximum number of connectives is 23.

For this experiment Grail performs a total of 136.070 axioms out of a total

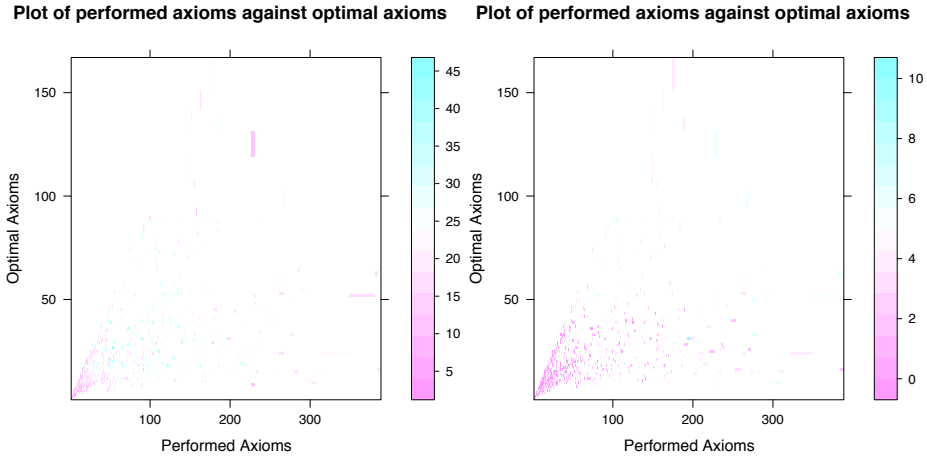

Figure 4.3: Evaluation of the performed versus the minimum number of axiom links

of over $7,9762 \cdot 10^{54}$.[1] Even though this is still more than double the optimal 61.487 axioms, it is a reduction from a truly astronomic number of potential axioms to something computationally feasible. The median number of axioms performed is 7, which is the median of the optimal number of axioms as well. The mean number of axioms performed is 24,95 (as opposed to 11,27 for the optimal number of axioms) and the maximum number of axioms performed is 388 (as opposed to 159 optimal). For a total of 3.388 sentences (62,12% of the total) the exact optimal number of axiom links is performed.

Figure 4.3 plots the total number of axiom links performed against the minimum number of axiom links required in order to obtain all semantic readings. The figure on the left uses colors to show degradation of performance as the number of atomic formulas increases, whereas the figure on the right shows the degradation of performance as the number of discontinuous connectives increases — deep purple indicating a low number of total axioms or discontinuous connectives and cyan indicating a very high number of them.

[1] To give an impression of how enormous this number is, this is a lot more than the number of atoms on Earth, which is $8,87 \cdot 10^{48}$ and a bit less than the volume of a small dwarf galaxy like NGC 1705, which has a volume of around $10^{55}\ m^3$

Chapter 5

Conclusions and Future Work

We have seen that in spite of the computational complexity of computing all axiom links for a given statement, a combination of constraints on the possible axiom links can reduce the total number of axiom links to just a bit over the optimal number of axiom links to be performed.

Some important questions remain unresolved. For example, are there conditions where the filtered axiom links correspond *exactly* to the axiom links which belong to a proof net? This would mean that after the filtering algorithm has done its job, the resulting axiom matrix would contain all and only those links which would be used for a proof net. In that case, the axiom possibilities would form a sort of shared representation of all proofs for a statement and moreover, it would be computed in $O(n^4)$ time. However, analyzing the incorrect axiom links of our experiment doesn't seem to give an easily identifiable handle on the subclasses of multimodal categorial grammar which would have this property.

The results on context-free grammars in Section 3.3 seem to provide an interesting possibility of obtaining polynomial parsing results. It has been observed by Schabes and Vijay-Shanker (1990) that every path in a derived tree for a tree adjoining grammar is described by a context-free grammar. This would open up the possibility to give an implementation of (fragments of) $\mathbf{NL}\Diamond_{\mathcal{R}}$ using Embedded Pushdown Automata.

Bibliography

Raffaella Bernardi and Richard Moot. Generalized quantifiers in declarative and interrogative sentences. *Logic Journal of the IGPL*, 11(4):419–434, 2003.

Raffaella Bernardi and Anna Szabolsci. Partially ordered categories: Optionality, scope and licensing. Technical report, Faculty of Computer Science, Free University of Bolzano and Department of Linguistics, New York University, 2007. Submitted to the Journal of Logic, Language and Computation.

Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

Philippe de Groote. An algebraic correctness criterion for intuitionistic multiplicative proof-nets. *Theoretical Computer Science*, 224(1–2):115–134, 1999.

Dirk Heylen. *Types and Sorts: Resource Logic for Feature Checking*. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University, 1999.

John Hopcroft and Richard Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

Max Kanovich. The multiplicative fragment of linear logic is NP-complete. Technical report, University of Amsterdam, 1991. ITLI Prepublication Series X-91-13.

François Lamarche. Proof nets for intuitionistic linear logic I: Essential nets. Technical report, Imperial College, 1994.

F. Xavier LLoré and Glyn Morrill. Difference lists and difference bags for logic programming of categorial deduction. In *Proceedings of SEPLN XI*, Deusto, 1995.

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

Michael Moortgat. Constants of grammatical reasoning. In Gosse Bouma, Erhard Hinrichs, Geert-Jan Kruijff, and Richard T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, pages 195–219. CSLI, Stanford, 1999.

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

Richard Moot. Automated extraction of type-logical supertags from the Spoken Dutch Corpus. In Srinivas Bangalore and Aravind Joshi, editors, *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach*. MIT Press, 2007. to appear.

Richard Moot. Graph algorithms for improving type-logical proof search. In *Proceedings Categorial Grammars 2004: an Efficient Tool for Natural Language Processing*. Elsevier, 2004.

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

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

Richard Moot and Christian Retoré. Les indices pronominaux du français dans les grammaires catégorielles. *Lingvisticae Investigationes*, 29(1):137–146, 2006.

Mati Pentus. Lambek calculus is NP-complete. *Theoretical Computer Science*, 357(1–3):186–201, 2006.

Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 362–367, Seattle, 1994. AAAI.

Yves Schabes and K. Vijay-Shanker. Deterministic left to right parsing of tree adjoining languages. In *Proceedings of the 28th Annual Meeting of the Association of Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, 1990. ACL.