Exploiting Sequent Structure in Membership Algorithms for the Lambek Calculus

RYAN T. MCDONALD
Department of Computer and Information Science
University of Pennsylvania
ryantm@cis.upenn.edu

ABSTRACT. This paper will examine the open problem of whether or not a sequent is derivable in the Lambek Calculus (**L**) in polynomial-time. This will be done through an investigation of *Lambek Calculus Graphs* (LC-Graphs), which were introduced by Penn[7] to represent the well-formedness constraints of a sequent's derivation in **L**. Presented here is a simplified version of LC-Graphs and their integrity criteria. We also show that storing a small amount of structural information about a sequent during parsing can reduce the number of integrity criteria for LC-Graphs from four to two. To this effect, a polynomial-time membership algorithm is presented that recognizes all derivable sequents and falsely recognizes an identifiable class of underivable sequents.

1 Introduction

Substructural logics are a group of logics whose proof systems only use a subset of the structural rules of classical proof systems. The most well known substructural logics include *Relevance Logic*, which does not employ weakening, and Girard's[3] *Linear Logic*, which uses neither weakening nor contraction. The complexity of membership algorithms for these logics is well studied with results often proving their intractability [4, 5].

Also in the class of substructural logics is the Lambek Calculus (**L**), which has the following structural rules:

$$(/L)\ \frac{\Gamma'\vdash B \quad \Gamma,A,\Gamma''\vdash S}{\Gamma,A/B,\Gamma',\Gamma''\vdash S} \qquad (/R)\ \frac{\Gamma,B\vdash A}{\Gamma\vdash A/B}^{(*)} \qquad (\backslash L)\ \frac{\Gamma'\vdash B \quad \Gamma,A,\Gamma''\vdash S}{\Gamma,\Gamma',B\backslash A,\Gamma''\vdash S} \qquad (\backslash R)\ \frac{B,\Gamma\vdash A}{\Gamma\vdash B\backslash A}^{(*)}$$

$$(\bullet L)\ \frac{\Gamma,A,B,\Gamma'\vdash S}{\Gamma,A\bullet B,\Gamma'\vdash S} \qquad (\bullet R)\ \frac{\Gamma\vdash A \quad \Gamma'\vdash B}{\Gamma,\Gamma'\vdash A\bullet B}$$

* if Γ is non-empty.

These operators are known as left-implication (\), right-implication (/) and product (•), where A/B means looking for premise B on the right to imply A

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)
Chapter 18, Copyright © 2003, Ryan T. McDonald

and A\B means looking for premise A on the left to imply B. For simplicity this paper will focus on the product-free fragment of **L** and sequents with non-empty sequences of premises.

The Lambek Calculus is of interest to both computer scientists and linguists because it is a basis for a deductive system for *Categorial Grammars* (CG). A CG consists of a set of base categories (N, NP, S, etc.), a lexicon and a distinguished category, s. Given a grammar, a string of words, $w = (w_1, \ldots, w_n)$ and their corresponding categories, $C = (C_1, \ldots, C_n)$[1], the principle question is whether or not w is generated by the grammar. If we view each category C_i as a premise and the distinguished category, s, as the consequent, then this question is the same as asking "is the sequent $C_1\ C_2\ \ldots\ C_n\ \vdash\ s$, derivable in **L**?". It is, however, unknown whether or not a sequent can be derived in **L** in polynomial-time as a function of the input size.

There are many reasons to believe that sequent derivability in **L** is intractable, including results showing that membership in **LP** (**L** with permutation) and Semidirectional Lambek Calculus [2] is NP-complete. However, since both these logics either partially or completely allow for permutation there is still a reasonable basis for believing that the Lambek Calculus can achieve membership recognition in polynomial-time. This paper essentially focuses on finding such a polynomial solution, but it may be the case that observations made here lead to an NP-complete reduction. During this process we will exclusively work with Lambek Calculus Graphs (LC-Graphs)[7]. Section 2 describes the basic framework that will be used in this investigation as well as presents a simplified version of LC-Graphs and their correctness criteria. In section 3 a complete but unsound polynomial-time chart parsing algorithm is presented which satisfies all but two of the four correctness criteria of LC-Graphs.

2 The Framework

Much of this and the next section have been drawn from Penn[7]. Both it and Roorda[8] have a more complete account. Below is only meant to provide enough background information to proceed.

Before the definition of LC-Graphs is presented we must first define two central constructs, **axiomatic formula** and **axiomatic linkage**.[2] Their definitions follow from how they are constructed from some given sequent. To illustrate, we will consider the sequent S:

$$(A/(A\backslash A))/A\ A\ A\backslash A\ A\backslash A\ \vdash\ A$$

[1]Here we assume a one-to-one correspondence of words and categories.

[2]Axiomatic formulae and linkages also serve as the basis for Lambek proof-nets [8], from which the correctness criteria of LC-Graphs is based [7].

2.1 Axiomatic Formulae

Polarize S so that each premise category becomes negatively polarized and the consequent becomes positively polarized. Then label each of these categories with a unique variable (see below).

For each polarized category, unfold it to obtain a sequence of axiomatic formulae using the following *lexical unfolding rules*:

$$\begin{array}{lll}
(A\backslash B)^{-}: t & \longrightarrow & A^{+}: u \; B^{-}: tu \\
(A\backslash B)^{+}: v & \longrightarrow & B^{+}: v' \; A^{-}: u \; [v := \lambda u.v'] \\
(A/B)^{-}: t & \longrightarrow & A^{-}: tu \; B^{+}: u \\
(A/B)^{+}: v & \longrightarrow & B^{-}: u \; A^{+}: v' \; [v := \lambda u.v']
\end{array}$$

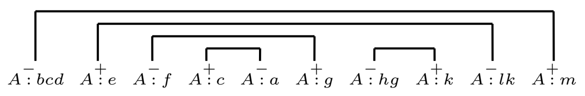
An *axiomatic formula* is an unfolded base category that was part of some base/complex category in either the sequent's premise set or it's conclusion. For example, the following sequent is labeled and unfolded to a produce a sequence of axiomatic formulae as follows:

$$\begin{array}{l}
(A/(A\backslash A))/A \;\; A \;\; A\backslash A \;\; A\backslash A \;\vdash\; A \longrightarrow \\
((A/(A\backslash A))/A)^{-}: b \; A^{-}: a \; (A\backslash A)^{-}: h \; (A\backslash A)^{-}: l \; A^{+}: m \;\longrightarrow \\
(A/(A\backslash A))^{-}: bc \; A^{+}: c \; A^{-}: a \; (A\backslash A)^{-}: h \; (A\backslash A)^{-}: l \; A^{+}: m \;\longrightarrow \\
A^{-}: bcd \; (A\backslash A)^{+}: d \; A^{+}: c \; A^{-}: a \; (A\backslash A)^{-}: h \; (A\backslash A)^{-}: l \; A^{+}: m \;\longrightarrow \\
A^{-}: bcd \; A^{+}: e \; A^{-}: f \; A^{+}: c \; A^{-}: a \; (A\backslash A)^{-}: h \; (A\backslash A)^{-}: l \; A^{+}: m \;\longrightarrow \\
A^{-}: bcd \; A^{+}: e \; A^{-}: f \; A^{+}: c \; A^{-}: a \; A^{+}: g \; A^{-}: hg \; (A\backslash A)^{-}: l \; A^{+}: m \;\longrightarrow \\
A^{-}: bcd \; A^{+}: e \; A^{-}: f \; A^{+}: c \; A^{-}: a \; A^{+}: g \; A^{-}: hg \; A^{+}: k \; A^{-}: lk \; A^{+}: m \qquad \text{where } d = \lambda f.e
\end{array}$$

Creating a sequence of axiomatic formulae from some sequent S is completely deterministic and takes a linear amount of time, with respect to the length of the sequent.

2.2 Axiomatic Linkages

To create an axiomatic linkage, match up pairs of positively and negatively polarized axiomatic formulae, X^{+} and X^{-}, i.e. with the same base category. Below is an example of a spanning linkage (spanning the whole sequent) for the above sequence of axiomatic formulae:



There can be many such linkages for a given sequence of axiomatic formulae. Any non-spanning linkage is defined as a sub-linkage and a linkage in which no two links cross is called a planar linkage (i.e. the above linkage is planar).

2.3 LC-Graphs

An LC-Graph for a spanning (or sub) linkage is a directed graph, $G = (V, E)$ such that V is the set of unique labels created at any point during lexical unfolding[3] and $(v, u) \in E$ iff either,

1. v labels a category that was positively unfolded and u labels the positive category that resulted from the unfolding.

2. There is a link between categories $X\overset{+}{:}v$ and $X\overset{-}{:}u$ in the linkage that G represents. Where u may be of the form $u = u_1u_2 \ldots u_n$, in which case $(v, u_1), (v, u_2), \ldots, (v, u_n) \in E$.

For example, the spanning linkage above has the following LC-Graph:

$$\begin{tikzcd}
 & b^- & & & \\
m^+ \arrow[ur] \arrow[r] \arrow[dr] & c^+ \arrow[r] & a^- & l^- & \\
 & \textcircled{d} \arrow[r] & e^+ \arrow[ur] \arrow[r] \arrow[dr] & k^+ \arrow[r] & h^- \\
 & & & g^+ \arrow[r] & f^-
\end{tikzcd}$$

Nodes labeled with $^+$ or $^-$ are referred to as **plus-nodes** and **minus-nodes** respectively. These nodes either label a positive axiomatic formula or a negative axiomatic formula after unfolding. We define a **lambda-node** as any node that labels a positive category which was unfolded during the creation of the axiomatic formulae. For instance, the node d, in the above graph, is a lambda-node since it labels the positive category (A\A) that was unfolded. We represent these nodes by enclosing them in circles. We may also define a **lambda-minus-daughter** as any node that labels a negative category which was the direct result of a positive category being unfolded. In the above graph, f is a lambda-minus-daughter. Similarly we can define a **lambda-plus-daughter** as any node that labels a positive category which was the direct result of another positive category being unfolded. The node e is a lambda-plus-daughter in the above graph. The concept of lambda-nodes and their daughters is central to a sequent's derivability in **L**.

A planar linkage and its corresponding LC-Graph, G, are **Integral** iff G satisfies:

- **I(1)** there is a unique node in G with in-degree 0 (a unique root), from which all other nodes are path-accessible,
- **I(2)** G is acyclic,
- **I(P)** for every lambda-node $v \in V$, there is a path from its plus-daughter, u, to its minus-daughter, w, and
- **I(CT)** for every lambda-node $v \in V$, there is a path in G, $v \to \ldots \to u \to x$, where x is a terminal node and u is not a lambda-plus-daughter (of any lambda-node).

Penn[7] showed that a sequent is valid if and only if there exists a planar spanning linkage whose corresponding LC-Graph is integral[4].

[3]V={a,b,c,d,e,f,g,h,k,l,m} for the above axiomatic formulae.

[4]The definition of an LC-Graph presented here is not exactly the same as that provided by [7]. However, it can be easily shown that, in terms of a sequent's derivability, they are equivalent.

2.4 Definitions

Definition 2.1 *An LC-Graph, G, is said to be* ***lambda-fragile*** *iff by adding edges from every lambda-node to their corresponding minus-daughters causes G to become connected (in the category of undirected graphs).*

Example:

$$\begin{tikzcd}
m^+ \arrow[rr] & & f^- & & & m^+ \arrow[dr] & & \\
\textcircled{d} \arrow[dr] & & & {} \arrow[r, "\text{add edge (d, f)}"] & {} & \textcircled{d} \arrow[r] \arrow[dr] & f^- & \\
 & e^+ \arrow[ul, bend left] \arrow[r] & b^- & & & & e^+ \arrow[ul, bend left] \arrow[r] & b^-
\end{tikzcd}$$

It should be noted that all connected LC-Graphs are already lambda-fragile.

Definition 2.2 *A* ***lexical-unfolding*** *is a contiguous sequence of axiomatic formulae that can be derived from some polarized category using the lexical-unfolding rules from §2.1.*

Example: The category $((A/(A\backslash A))/A)\overset{-}{:}b$ unfolds to the following lexical-unfolding,

$$(A/(A\backslash A))\overset{-}{:}bc\ A\overset{+}{:}c \Rightarrow A\overset{-}{:}bcd\ (A\backslash A)\overset{+}{:}d\ A\overset{+}{:}c \Rightarrow \mathbf{A\overset{-}{:}bcd\ A\overset{+}{:}e\ A\overset{-}{:}f\ A\overset{+}{:}c}$$

Observe that all the axioms of a lexical-unfolding will be contiguous, and all axioms belong to only one lexical-unfolding.

Definition 2.3 *An axiomatic formula is considered* ***left-peripheral*** *(right-peripheral) iff it exists at the left (right) endpoint of a lexical-unfolding.*

Example: $A\overset{+}{:}g$ and $A\overset{-}{:}bcd$ are left-peripheral and $A\overset{-}{:}gh$ and $A\overset{+}{:}c$ are right-peripheral in the following two lexical-unfoldings that result from unfolding the categories, $(A\backslash A)\overset{-}{:}h$ and $((A/(A\backslash A))/A)\overset{-}{:}b$

$$A\overset{+}{:}g\ A\overset{-}{:}gh\ A\overset{-}{:}bcd\ A\overset{+}{:}e\ A\overset{-}{:}f\ A\overset{+}{:}c$$

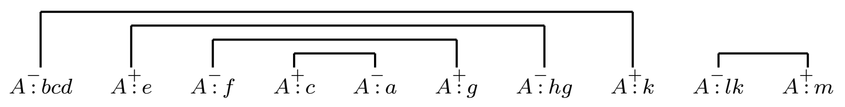
Definition 2.4 *A linkage L (sub or spanning) is said to* ***span*** *an axiomatic formula A iff A is completely enclosed by the two axioms that the L connects.*

Example: The linkage L spans $A\overset{+}{:}e$, $A\overset{-}{:}f$, $A\overset{+}{:}c$, $A\overset{-}{:}a$, $A\overset{+}{:}g$ and $A\overset{-}{:}hg$.

$$\overbrace{A\overset{-}{:}bcd \quad A\overset{+}{:}e \quad A\overset{-}{:}f \quad A\overset{+}{:}c \quad A\overset{-}{:}a \quad A\overset{+}{:}g \quad A\overset{-}{:}hg \quad A\overset{+}{:}k}^{L} \quad A\overset{-}{:}lk \quad A\overset{+}{:}m$$

Definition 2.5 *An axiomatic formula X in a linkage L is considered* ***exposed*** *iff in L, there is no linkage that spans X.*

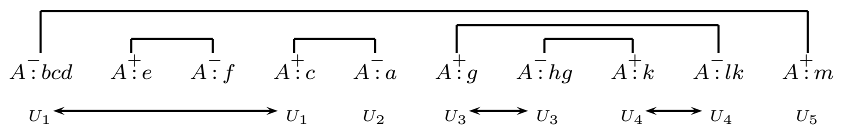
Example: In the following linkage $A\overset{-}{:}bcd$, $A\overset{+}{:}k$, $A\overset{-}{:}lk$ and $A\overset{+}{:}m$ are exposed:



We will further define $A\overset{-}{:}bcd$ and $A\overset{-}{:}lk$ as **left-exposed** and $A\overset{+}{:}k$ and $A\overset{+}{:}m$ as **right-exposed**.

Definition 2.6 *A* ***closed-linkage*** *of some set of lexical-unfoldings, U, is a linkage that uses only and all the axioms in U. A sublinkage may contain a closed-linkage.*

Example: Consider the sequent below with the following lexical-unfoldings, U_1, U_2, U_3, U_4 and U_5. The linkage below contains two closed-linkages, $U' = \{U_1, U_2, U_5\}$ and $U'' = \{U_3, U_4\}$



2.5 Simplifying LC-Graphs

At this point we consider three new integrity criteria for LC-Graphs:

- **I(C)** G is connected
- **I(LF)** G is lambda-fragile
- **I(R)** G has a unique root node (node with in-degree of 0)[5]

Proposition 2.1 *If an LC-Graph satisfies I(R), then I(1), I(2) and I(C) are equivalent.*

Proof. I(C) $\Rightarrow$ I(2): Follows from the fact that all nodes in an LC-Graph have a maximum in-degree of 1. I(1) $\Rightarrow$ I(C): Follows directly from the definition of I(1). I(2) $\Rightarrow$ I(1): Given by [7] proposition 5.1. ■

Proposition 2.2 *I(P) and I(LF) imply I(C).*

Proof. Consider some lambda-node d, and it's corresponding plus and minus-daughters e and f. By I(P), there is a path from e to f. Furthermore, by the definition of an LC-Graph, there is a path from d to e. Therefore there is a path from d to f. So adding another edge (d, f) to the graph has no bearing on that graphs connectivity. Therefore the graph must already be connected. ■

[5]Note we don't require that the root node have a path to all other nodes as is the case with I(1)

Proposition 2.3 *A sequent S is derivable in L iff there exists a planar spanning linkage, whose corresponding LC-Graph satisfies I(R), I(LF), I(P) and I(CT).*

Proof. Consequence of propositions 2.1 and 2.2 ■

Where I(R) can be easily checked during lexical unfolding by ensuring the existence of exactly one consequent in the sequent, leaving only I(LF), I(P) and I(CT) needing to be enforced.

3 Enforcing Lambda-Fragility

Penn[7] displayed an algorithm that used a chart parser to incrementally create all the possible planar spanning linkages for a given sequent. Chart parsers were designed for CFGs, in which the use of non-terminals on RHSs is invariant over their specific LHS derivations. Therefore each edge in the chart could correspond to possibly many different sublinkages. Penn also showed how to store LC-Graphs on each edge so that each graph corresponds to some linkage represented by that edge. A sequent's derivability may then be determined by ensuring that at least one of the LC-Graphs on the spanning edge is integral. However, because an edge may represent an exponential number of linkages, it must then also store an exponential number of LC-Graphs.[6]

Here an extension to this algorithm is presented that enforces lambda-fragility. In other words it forces each LC-Graph associated with the spanning edge in the chart to be lambda-fragile.

Proposition 3.1 *All nodes $a_1, \ldots, a_n$ that represent labels in the same lexical unfolding will exist in the same connected component (in the undirected sense) for any spanning lambda-fragile LC-Graph.*

Proof. By induction on the number of unfoldings to create the lexical-unfolding (see lexical unfolding rules in §2.1). A single unfolding will either result in two axiomatic formula with a common label (negative unfolding) or two axiomatic formula that are the positive and negative daughters of some lambda node (positive unfolding). In the former case, the common label will ensure that all the labels are in the same connected component and in the latter case, the edges from the lambda-node to its plus and minus daughters will force all labels to be in the same component.

Inductively assume after n unfolding steps a lexical unfolding consits of m different unfolded categories (possibly not axiomatic). The $n + 1^{st}$ unfolding step must be on only one of these categories. By a similar analysis

[6]This approach is similar to the parsing algorithm presented by Morrill[6]. Where Penn stores possible LC-Graphs over spans, Morrill stores possible unifications over spans.

to the base case all the labels of the two new categories created will be in the same connected component. Hence by induction, all of the labels for the lexical-unfolding will be in the same component. ■

Proposition 3.2 *A spanning LC-Graph, G, for spanning linkage L is not lambda-fragile iff L contains a closed-linkage.*

Proof. Assume some LC-Graph G is not lambda-fragile. By proposition 3.1 all the labels in a single lexical-unfolding are in the same connected component. So if G is not lambda-fragile, then there are at least two connected components and each will consist of all and only a set of labels for some subset of lexical unfoldings. This component can only have been created by a closed-linkage (since links correspond to edges in LC-Graphs).

Conversely assume that L contains a closed-linkage over some subset of lexical-unfoldings U and that G is lambda-fragile. By proposition 3.1 we know that all the labels of some unfolding in U will be in the same connected component. Also, since each link creates an edge in the LC-Graph, it must be the case that all the labels of all the axioms in U will be in the same connected component. And, since there are no other links to lexical-unfoldings outside of U, then it also must be the case that there are no edges from any node in the component of the labels of U to nodes outside of it. Hence, this component is disconnected from the rest of the graph and G cannot be lambda-fragile since adding an edge from a lambda-node to a minus-daughter can only connect two nodes in the same lexical-unfolding. ■

With proposition 3.2 in mind we would like each edge to store some kind of periphery information such that it forces the parser to never add an edge to the chart that can only be created using a sub-linkage that contains a closed-linkage.

To do this the chart parser will use the following six rules[7]:

1) $L \rightarrow B\ L$
2) $B \rightarrow X^-\ L\ X^+$, for every basic category X
3) $B \rightarrow X^+\ L\ X^-$, for every basic category X
4) $B \rightarrow X^-\ X^+$, for every basic category X
5) $B \rightarrow X^+\ X^-$, for every basic category X
6) $L \rightarrow B$

Using these six rules and by following the procedures outlined by Penn[7, sec. 7] it is possible to store LC-Graphs on each edge in the chart that correspond to the different sublinkages that that edge represents. However, as stated earlier, this may result in an exponential number of LC-Graphs. In order to keep the amount of information polynomial in size, we will drop

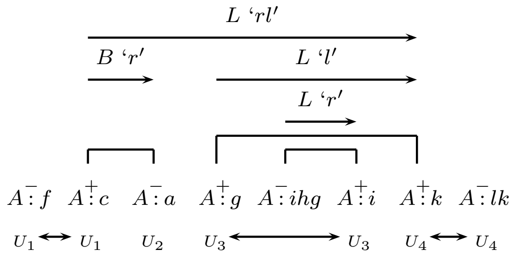
[7]These are the rules Penn[7] uses in his algorithm.

the idea of keeping LC-Graphs on the edges[8]. Instead we will store a small amount of information on each edge that will allow us to claim that if a spanning edge is created, then there exists at least one spanning linkage whose corresponding LC-Graph is lambda-fragile. The idea here is to take the first step towards storing just enough information so that one can answer the yes/no derivability question after parsing.

The information that will be stored on each edge are the following strings:

'l':	there is a left-peripheral axiom left-exposed in the sublinkage this edge represents.
'r':	there is a right-peripheral axiom right-exposed in the sublinkage this edge represents.
'rl':	a combination of the two cases above, with all right-peripheries existing to the left of all left-peripheries.
'nil':	there are no exposed peripheries in the sublinkage this edge represents.

Example: Consider the partial sequent with lexical-unfoldings, U_1, U_2, U_3 and U_4. Below is an example of a sub-linkage with the final edge having both left and right-peripheral exposed axioms, $A\overset{-}{:}a$ and $A\overset{+}{:}g$:



In order to use this new information some additional steps must be taken as the chart-parser uses each rule.

Rules 2 & 4: If axiomatic formula X^- is left-peripheral or axiomatic formula X^+ is right peripheral then B will store 'l' or 'r' accordingly. However, if X^- is left-peripheral and X^+ is right-peripheral then do not add B to the chart unless B will be a spanning edge. B stores nothing otherwise. There is never a case when B will store an 'rl'. Note: Rule 2 pays no attention to what L is storing.

Rules 3 & 5: are symmetric to rules 2 & 4.

Rule 6: L stores whatever B is storing.

Rule 1: The following table outlines what L_{left} (the left-hand-side non-terminal) stores for all cases:

[8]The number of edges stored for chart parsing is known to be polynomial in size for a CFG.

B	$\mathbf{L}_{right}$	$\mathbf{L}_{left}$
‘nil’	‘nil’,‘r’,‘l’ or ‘rl’	‘nil’,‘r’,‘l’ or ‘rl’ respectively
‘l’	‘nil’,‘l’	‘l’
‘l’	‘r’,‘rl’	do not add to chart
‘r’	‘nil’,‘r’	‘r’
‘r’	‘l’,‘rl’	‘rl’
‘rl’	*	cannot happen (B never stores ‘rl’)

Observe that we do not add any edge to the chart that adjoins a left-exposed left-periphery to a right-exposed right-periphery. Also, we stipulate that if L_{left} is the spanning edge, then it is always added to the chart.

In order to make the number of edges polynomial in size we will union edge periphery information. If two edges of the same type (B or L) for the same span are created, E_1 and E_2, we will replace them with a third edge E_3 that stores the union of the periphery information stored in E_1 and E_2. For example, if E_1 is storing {‘l’,’rl’} and E_2 is storing {‘nil’,‘l’} then E_3 will store {‘l’,‘rl’,‘nil’}.

Edge union causes no change in the behaviour of parsing rules 2-6 due to the uniqueness of B-edges for a given span. Parsing rule 1 does need to be updated so that the resulting edge L_{left} stores all the possible periphery outcomes when adjoining edges B and L_{right}. This can easily be done in constant time by considering the bounded number of finite possibilities.

Proposition 3.3 *An edge E that is stored on the chart has a right-exposed right-peripheral axiom iff it is storing either ‘r’ or ‘rl’ (similarly for left-exposed left-peripheral axioms).*

Proof. Assume edge E has a right-exposed right-peripheral axiom. If the edge is a B-edge, then it can only have a right-exposed right-peripheral axiom at its rightmost axiom. By inspection of the augmented parsing rules 2-5 it is can be seen that E will store an ‘r’. Say that E is an L-edge. If it was created by rule 6, then its structure will be identical to some B-edge and must store an ‘r’. If it was created by rule 1, then it is the adjunction of some B-edge and some L-edge. Inductively we can assume that if one of these edges contain a right-exposed right-peripheral axiom then they will be storing ‘r’ or ‘rl’. Then, by inspecting the augmented parsing rule 1, it can be seen that E will store an ‘r’ or an ‘rl’ depending on the structure of the B and L-edge it is being created from.

Assume E is storing either an ‘r’ or an ‘rl’. If the edge is a B-edge, then by inspection of the augmented parsing rules 2-5 it can be seen that E will store an ‘r’ iff its rightmost axiom is a right-periphery (and hence right-exposed). Say that E is an L-edge. If it was created by rule 6, then its structure will be identical to some B-edge and so will store an ‘r’ only if its rightmost axiom is a right-periphery. If it was created by rule 1, then it is the adjunction of some B-edge and some L-edge. By inspection of the parsing rules, E will store an 'r' or an 'rl' iff at least one of the two adjoining edges are storing an 'r' or 'rl'. Inductively we can assume that since one of these edges is storing an 'r' or an 'rl' then it contains a right-exposed right-peripheral axiom and therefore so will E. ■

Proposition 3.4 *An edge E, for a sub-linkage SL is added to the chart iff it is not the case that SL contains a closed-linkage for some set of lexical-unfoldings U.*

Proof. Assume an edge E for a sub-linkage SL is added to the chart. By an examintion of the augmented parsing rules above, the only means by which an edge cannot be added to the chart is if it can only be created over a sub-linkage that has a left-exposed left-periphery axiom to the left of some right-exposed right-periphery axiom. This is precisely when a closed-linkage occurs and therefore SL does not contain a closed-linkage.

Conversely, assume that SL does not contain a closed-linkage. Again, there cannot be a left-exposed left-periphery axiom to left of a right-exposed right-periphery axiom (otherwise there would be a closed-linkage). So again, E will be added to the chart.

Proposition 3.5 *A spanning edge is added to the chart iff at least one of its representative LC-Graphs is lambda-fragile.*

Proof. A consequence of 3.2 and 3.4. ■

After the parsing algorithm is run, it can be said whether or not there exists a lambda-fragile LC-Graph for some spanning linkage. Since all integral LC-Graphs are lambda-fragile, then this algorithm recognizes all valid sequents. Furthermore, if we stored LC-Graphs on chart edges in conjunction with periphery information, then all lambda-fragile graphs will be stored on the spanning edge and all non-lambda-fragile graphs discarded. Hence, the algorithm does not throw away any valid parses that may become useful in methods to efficiently enforce I(P) and I(CT) - if such methods exist.

We can also identify the precise class of invalid sequents that are being recognized by this algorithm - those that have a planar linkage/LC-Graph pair that satisfy I(R) and I(LF), but do not satisfy at least one of I(P) or I(CT). For example, the following underivable sequent falls into this class:

$$(A/(A\backslash(A\backslash A))) : a \ \vdash \ A : m \ \Rightarrow \ A \overset{-}{:} abc \ A \overset{+}{:} f \ A \overset{-}{:} g \ A \overset{-}{:} e \ A \overset{+}{:} b \ A \overset{+}{:} m$$

where $c = \lambda e.d$ and $d = \lambda g.f$

has a planar spanning linkage with the following LC-Graph satisfying I(R) and I(LF):

$$\begin{tikzcd}
 & a^- & & & \\
m^+ \arrow[ur] \arrow[r] \arrow[dr] & b^+ \arrow[r] & e^- & & \\
 & \boxed{c} \arrow[r] & \boxed{d} \arrow[r] & f^+ \arrow[r] & g^-
\end{tikzcd}$$

4 Discussion

By storing a small amount of bounded information on each edge in a chart parser, it was shown that we could force certain properties of the LC-Graphs that are associated with those edges - namely that they are all lambda-fragile. Of course the only problem is that not all lambda-fragile LC-Graphs are integral, which means the algorithm presented here is not sound.

Future studies should focus on finding a similar method that uses sequent structure to force all LC-Graphs on the spanning edge to satisfy I(P). If this can be done in conjunction with the above algorithm, then it would be likely that I(CT) could be satisfied as well, since both integrity criteria involve ensuring the existence of particular paths.

References

[1] Carpenter, Bob. *Type-Logical Semantics*. Cambridge, MA: The MIT Press (1997).

[2] Dörre, Jochen. *Parsing for semidirectional lambek calculus is NP-Complete*, Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics (1996), pp. 95-100.

[3] Girard, J.-Y. *Linear Logic*, Theoretical Computer Science **56** (1987), pp. 1-102.

[4] Kanovich, M. *The multiplicative fragment of linear logic is NP-complete*, Technical Report X-91-13, Institute for Language, Logic and Information (1991).

[5] Lincoln, P., Mitchell, A., Scedrov, A. and Shankar, N. *Decision problems for propositional linear logic*, In Proceedings 31st Annual IEEE Symposium on Foundations of Computer Science (1990).

[6] Morrill G. *Memoisation of categorial proof nets: parallelism in categorial processing*, Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Datlunya (1996).

[7] Penn, Gerald. *A Graph theoretic approach to sequent derivability in the lambek calculus*, Electronic Notes in Theoretical Computer Science **53** (2001).

[8] Roorda, Dirk. "Resource Logics: Proof Theoretic Investigations" Ph.D. Thesis, Universiteit van Amsterdam (1991).

[9] Urquhart, A. *The undecidability of entailment and relevant implication*, Journal of Symbolic Logic, **49** (1990), pp. 1059-1073.

Acknowledgments

The author would like to thank Gerald Penn for donating much of his valuable time to provide many useful discussions and insights on the presented work. This work was supported by a grant from the Natural Sciences and Engineering Reasearch Council of Canada.