A Compilation-Chart Method for Linear Categorial Deduction

Mark Hepple
Dept. of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
Sheffield S1 4DP, UK
hepple@dcs.shef.ac.uk

Abstract

Recent work in categorial grammar has seen proposals for a wide range of systems, differing in their 'resource sensitivity' and hence, implicitly, their underlying notion of 'linguistic structure'. A common framework for parsing such systems is emerging, whereby some method of linear logic theorem proving is used in combination with a system of labelling that ensures that only deductions appropriate to the relevant categorial formalism are allowed. This paper presents a deduction method for implicational linear logic that brings with it the benefit that chart parsing provides for CFG parsing, namely avoiding the need to recompute intermediate results when searching exhaustively for all possible analyses. The method involves compiling possibly higher-order linear formulae to indexed first-order formulae, over which deduction is made using just a single inference rule.

1 Introduction

This paper presents a method applicable to parsing a range of categorial grammar formalisms, in particular ones that fall within the 'type-logical' tradition, of which the (associative) Lambek calculus **L** is the most familiar representative (Lambek, 1958). Recent work has seen proposals for a range of such systems, differing in their resource sensitivity (and hence, implicitly, their underlying notion of 'linguistic structure'), in some cases combining differing resource sensitivities within a single system.[1] Some of these proposals employ a 'labelled deduction' methodology (Gabbay, 1994), whereby the types in a proof are associated with labels under a specified discipline, the labels recording proof information as a basis for ensuring correct inferencing.

Alongside such developments, various work has addressed the associated parsing problem.[2] Of particular interest here are systems that employ a theorem proving method that is (perhaps implicitly) appropriate for use with linear logic, and combine it with a labelling system that restricts admitted deductions to be those of some weaker logic. Moortgat (1992) shows how a linear proof net method may be combined with a range of labelling disciplines to provide deduction for a range of categorial systems. Morrill (1995) shows how **L** types may be translated to labelled implicational linear types, with deduction implemented via a version of SLD resolution. The crucial observation is that linear logic stands above *all* of the type-logical systems proposed as categorial formalisms in the hierarchy of substructural logics, and hence linear logic deduction methods can provide a common basis for parsing all of these systems.

The present work contributes to this project by providing a method of deduction for the implicational fragment of linear logic that, like chart parsing for PSG, avoids recomputation of results, i.e. where any combination of types contributes to more than one overall analysis, it need only be computed once. In what follows, I will first introduce deduction for implicational linear logic, and discuss its incompatibility with a chart-like deduction approach, before presenting a compilation method that converts formulae to a form for which a chart-like deduction method is possible. Finally, I will introduce the Morrill (1995) method for translating Lambek types to labelled linear types, as a basis for illustrating how the chart-compilation method can be used as a general framework for categorial deduction, via the use of such translations.

[1] See, for example, the formalisms developed in Moortgat & Morrill (1991), Moortgat & Oehrle (1994), Morrill (1994), Hepple (1995).

[2] Approaches include sequent proof normalisation methods for **L** (König, 1989; Hepple, 1990; Hendriks, 1992), chart parsing methods for **L** (König, 1990; Hepple, 1992), and proof net methods for a range of systems (Roorda, 1991; Moortgat, 1992).

2 Implicational Linear Logic

Linear logic is an example of a "resource-sensitive" logic, requiring that in any deduction, every assumption ('resource') is used precisely once. We consider only the implicational fragment of (intuitionistic) linear logic.[3] The set of formulae $\mathcal{F}$ arises by closing a (nonempty) set of atomic types $\mathcal{A}$ under the linear implication operator o– (i.e. $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F}$o–$\mathcal{F}$). Various alternative formulations are possible. We here use a natural deduction formulation, requiring the following rules (o–*elimination* and *introduction* respectively):

$$\frac{\text{A}\text{o–}\text{B} : a \qquad \text{B} : b}{\text{A} : (ab)}\ \text{o–E} \qquad\qquad \frac{\begin{array}{c}[\text{B} : v] \\ \text{A} : a\end{array}}{\text{A}\text{o–}\text{B} : \lambda v.a}\ \text{o–I}$$

Eliminations and introductions correspond to steps of functional application and abstraction, respectively, as the lambda term labelling reveals. The introduction rule discharges precisely one assumption (B) within the proof to which it applies (ensuring linear use of resources, i.e. that each resource is used precisely once). Consider the following proof that X o–Y, Y o–Z ⇒ X o–Z

$$\dfrac{\dfrac{\text{X}\text{o–}\text{Y} : x \qquad \dfrac{\text{Y}\text{o–}\text{Z} : y \qquad [\text{Z} : z]}{\text{Y} : (yz)}}{\text{X} : (x(yz))}}{\text{X}\text{o–}\text{Z} : \lambda z.(x(yz))}$$

Following Prawitz (1965), a *normal form* for proofs can be defined using just the following (meaning preserving) contraction rule (analogous to β-conversion). This observation is of note in that it restricts the form of proofs that we must consider in seeking to prove some possible theorem.

$$\dfrac{\dfrac{\begin{array}{c}[\text{B}] \\ \text{A}\end{array}}{\text{A}\text{o–}\text{B}} \qquad \text{B}}{\text{A}} \quad \triangleright \quad \begin{array}{c}\vdots \\ \text{B} \\ \text{A}\end{array}$$

The normal form proofs of this system have a straightforward structural characterisation, that their *main branch* (the unique path from an assumption to the proof's end-type that includes no minor premise of an elimination inference) consists of a sequence of (≥ 0) eliminations followed by a sequence of (≥ 0) introductions.

The differential status of the left and right hand side formulae in a sequent may be addressed in terms of *polarity*, with left formulae being deemed to have positive polarity, and the right formula to have negative polarity. Polarity applies also to subformulae, i.e. in a formula X o–Y with a given polarity p, the subformula X has the same polarity p, and Y has the opposite polarity. For example, a positively occuring higher-order type might have the following pattern of positive and negative subformulae: $(\ \text{X}^{+}$ o– $(\text{Y}^{-}$ o– $\text{Z}^{+}\ )^{-}\ )^{+}$ Consider the following proof involving this type:

$$\dfrac{\dfrac{\dfrac{\text{X}\text{o–}(\text{Y}\text{o–}\text{Z}) \qquad \dfrac{\text{Y}\text{o–}\text{W} \qquad \dfrac{\text{W}\text{o–}\text{Z} \qquad [\text{Z}]}{\text{W}}}{\text{Y}}}{\text{Y}\text{o–}\text{Z}}}{\text{X}}$$

Observe that the involvement of 'hypothetical reasoning' in this proof (i.e. the use of an additional assumption that is later discharged) is driven by the presence of the higher-order formula, and that the additional assumption in fact corresponds to the positive subformula occurrence Z within that higher-order formula. In the following proof that X o–(Y o–(Y o–Z)) ⇒ X o–Z, hypothetical reasoning again arises in relation to positive subformulae, i.e. the subformula Y o–Z of the higher-order formula $(\ \text{X}^{+}$ o– $(\ \text{Y}^{-}$ o– $(\ \text{Y}^{+}$ o– $\text{Z}^{-}\ )^{+}\ )^{-}\ )^{+}$, as well as the subformula Z of the (overall negative) goal formula $(\text{X}^{-}$ o– $\text{Z}^{+}\ )^{-}$.

$$\dfrac{\dfrac{\dfrac{\text{X}\text{o–}(\text{Y}\text{o–}(\text{Y}\text{o–}\text{Z})) \qquad \dfrac{[\text{Y}\text{o–}\text{Z}] \qquad [\text{Z}]}{\text{Y}}}{\text{Y}\text{o–}(\text{Y}\text{o–}\text{Z})}}{\text{X}}}{\text{X}\text{o–}\text{Z}}$$

More specifically, additional assumptions link to *maximal* positive subformulae, i.e. a subformula Y^{+} in a context of the form $(\text{X}^{-}$ o– $\text{Y}^{+})^{-}$, but not in $(\text{Y}^{+}$ o– $\text{Z}^{-})^{+}$. For an even more complex formula, e.g. $(\text{V}^{+}$ o–$(\text{W}^{-}$ o–$(\text{X}^{+}$ o–$(\text{Y}^{-}$ o– $\text{Z}^{+}\ )^{-}\ )^{+}\ )^{-}\ )^{+}$, we might find that a proof would involve not only an additional assumption corresponding to the positive subformula X o–(Y o–Z)), but that reasoning with that assumption would in turn involve a further additional assumption corresponding to its positive subformula Z.

[3] It follows that the parsing method to be developed applies only to categorial systems having only implicational connectives. It is standard in categorial calculi to include also a 'product' operator, enabling matter like addition of substructures, e.g. **L** has a product (commonly notated as) •, with the Lambek implicationals / and \ being its left and right residuals. Although it is appealing from a logical point of view to include such operators, their use is not motivated in grammar.

3 A Compilation-Chart Method

Standard chart parsing for PSG has the advantage that a simple organising principle governs the storage of results and underpins search, namely *span* within a linear dimension, specified by limiting left and right points. A further crucial feature is that what we derive as an item for any span is purely a function of the results derived for substretches of that span, and ultimately of the lexical categories that it dominates (assuming a given grammar). Deduction in implicational linear logic lacks both of these features, although, as we shall see shortly, some notion of 'span' can be specified. The crucial problem for developing a chart-like method is the fact that, in combining any two elements A,B $\Rightarrow$ C, there is an *infinite* number of possible results C we could derive, and that what we in fact *should* derive depends not just on the formulae themselves, but upon other formulae that might combine with that result. More particularly, the reasoning needed to derive C is liable to involve hypothetical elements whose involvement is driven by the presence of some higher-order type elsewhere.

First-Order Linear Deduction

Let us begin by avoiding this latter problem by considering the fragment involving only first-order formulae, i.e. those defined by $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \circ\!\!- \mathcal{A}$, and furthermore allow only atomic goals (i.e. so A is atomic in any $\Gamma \Rightarrow$ A). Consequently, the [$\circ$–I] rule is not required, and hypothetical reasoning excluded. In combining types using just the remaining elimination rule, we must still ensure linear use of resources, i.e. that no resource may be used more than once in any deduction, and that in any *overall* deduction, every resource has been used. These requirements can be enforced using an indexation method, whereby each initial formula in our database is marked with a unique index (or strictly a singleton set containing that index), and where a formula that results from a combination is marked with the union of the index sets of the two formulae combined.[4] We can ensure that no initial assumption contributes more than once to any deduction by requiring that wherever two formulae are combined, their index sets must be *disjoint*. Thus, we require the following modified [$\circ$–E] rule (where ϕ, ψ, π are index sets, and $\uplus$ denotes union of sets that are required to be disjoint):

$$\frac{\phi : \mathrm{A} \circ\!\!- \mathrm{B} : a \qquad \psi : \mathrm{B} : b}{\pi : \mathrm{A} : (ab)} \quad \pi = \phi \uplus \psi$$

In proving $\Gamma \Rightarrow$ A, a successful overall analysis is recognised by the presence of a database formula A whose index set is the full set of indices assigned to the initial formulae in Γ. For example, to prove X$\circ$–X, X$\circ$–X, X$\circ$–Y, Y $\Rightarrow$ X, we might start with a database containing entries as follows (the numbering of entries is purely for exposition):

1. $i : \mathrm{X} \circ\!\!- \mathrm{X} : v$
2. $j : \mathrm{X} \circ\!\!- \mathrm{X} : w$
3. $k : \mathrm{X} \circ\!\!- \mathrm{Y} : x$
4. $l : \mathrm{Y} : y$

Use of the modified elimination rule gives additional formulae as follows:

5.	$\{k, l\} : \mathrm{X} : xy$	[3+4]
6.	$\{i, k, l\} : \mathrm{X} : v(xy)$	[1+5]
7.	$\{j, k, l\} : \mathrm{X} : w(xy)$	[2+5]
8.	$\{i, j, k, l\} : \mathrm{X} : v(w(xy))$	[1+7]
9.	$\{i, j, k, l\} : \mathrm{X} : w(v(xy))$	[2+6]

There are two successful analyses, numbered 8 and 9, which we recognise by the fact that they have the intended goal type (X), and are indexed with the full set of the indices assigned to the initial left hand side formulae. Note that the formula numbered 5 contributes to both of the sucessful overall analyses, without needing to be recomputed. Hence we can see that we have already gained the key benefit of a chart approach for PSG parsing, namely avoiding the need to recompute partial results. It can be seen that indexing in the above method plays a role similar to that of 'spans' within standard chart parsing.

An adequate algorithm for use with the above approach is easily stated. Given a possible theorem $\mathrm{B}_1, \ldots, \mathrm{B}_n \Rightarrow$ A, the left hand side formulae are each assigned unique indices and semantic variables, and put on an agenda. Then, a loop is followed in which a formula is taken from the agenda and added to the database, and then the next formula is taken from the agenda and so on until the agenda is empty. Whenever a formula is added to the database, a check is made to see if it can combine with formulae already there, in which case new formulae are generated, which are added to the agenda. When the agenda is empty, a check is made for any successful overall analsyses, identified as described above. Note that since the result of a combination always bears an index set larger than either of its parent formulae, and since the maximal index set that any formula can carry includes all and only the indices assigned to the original left hand side formulae, the above process must terminate.

Higher-Order Linear Deduction

Let us turn now to the general case, where higher-order formulae are allowed. The method to be described involves compiling the initial formulae (which may be higher-order) to give a new, possibly larger, set of formulae which are all first order. We observed above how hypothetical reasoning in a proof is driven by the presence within higher-order formulae of positively occurring subformu-

[4] See Lloré & Morrill (1995) for a related use of indexing in ensuring linear use of resources.

lae. The compilation method involves identifying and excising such subformulae (thereby simplifying the containing formulae) and including them as additional assumptions. For example, this method will simplify the higher-order formula X⊸(Y⊸Z) to become X⊸Y, generating an additional assumption of Z. The two key challenges for such an approach are firstly ensuring that the additional assumptions are appropriately used (otherwise invalid reasoning will follow), and secondly ensuring that a proof term appropriate to the original type combination is returned.

Consider an attempt to prove the (invalid) type combination: X⊸Z⊸(Y⊸Z), Y ⇒ X. Compilation of the formula X⊸Z⊸(Y⊸Z) yields two formulae X⊸Z⊸Y and Z, so the initial query becomes X⊸Z⊸Y, Z, Y ⇒ X, which is provable. The problem arises due to inappropriate use of the additional formula Z, which should only be used to prove the argument Y (just as Z's role would be to contribute to proving the argument Y⊸Z in a standard proof involving the original formula X⊸Z⊸(Y⊸Z)). The solution to this problem relies upon the indexing method adopted above. The additional assumption generated in compiling a higher-order formula such as X⊸(Y⊸Z) will itself be marked with a unique index. By recording this index on the argument position from which the additional assumption was generated, we can enforce the requirement that the assumption contributes to the derivation of that argument. Note that a single argument position may give rise to more than one additional assumption, and so in fact an index *set* that should be recorded. For example, The (indexed) formula i:X⊸(Y⊸Z⊸W) will compile to give three indexed formulae:

$$i:\text{X}\multimap(\text{Y}:\{j,k\}) \qquad j:\text{Z} \qquad k:\text{W}$$

We require a modified elimination rule that will enforce appropriate usage:[5]

$$\frac{\phi:\text{A}\multimap(\text{B}:\alpha):a \qquad \psi:\text{B}:b}{\pi:\text{A}:(ab)} \qquad \begin{array}{l}\pi=\phi\uplus\psi\\ \alpha\subset\psi\end{array}$$

Note that the compilation process must also generate additional assumptions corresponding to the positive subformulae of the right hand side of a query, e.g. compilation of X⊸Y, Y⊸Z ⇒ X⊸Z simplifies the right hand side formula to atomic X, giving and additional assumption Z.

The second challenge we noted for such an approach is ensuring that a proof term (loosely, the 'semantic recipe' of the combination) appropriate to the original type combination is returned. Let us illustrate how this can be achieved with a simple example. Consider the following proof:

$$\dfrac{\dfrac{\dfrac{\text{X}\multimap(\text{Y}\multimap\text{Z}):x \qquad \text{Y}\multimap\text{W}:y \qquad \dfrac{\text{W}\multimap\text{Z}:w \qquad [\text{Z}:z]}{\text{W}:wz}}{\text{Y}:y(wz)}}{\text{Y}\multimap\text{Z}:\lambda z.y(wz)}}{\text{X}:x(\lambda z.y(wz))}$$

Deriving the argument Y⊸Z of the higher-order formula involves a final introduction step, which, semantically, corresponds to an abstraction step that binds the variable semantics of the additional assumption Z. The possibility arises that compilation might insert the abstraction into the semantics of the compiled formula, so that it later binds the variable of the additional formula. For example, compilation of X⊸(Y⊸Z) might yield X⊸Y with term $\lambda y.x(\lambda z.y)$ and Z with variable term z, so that combining the former with some formula derived from the latter (i.e. whose term included z) would cause the free occurrence of z to become bound, giving a result such as $x(\lambda z.f(z))$. In that case, we can see that although compilation has eliminated the need for an explicit introduction step in the proof, the step still occurs implicitly in the semantics.

Of course, anyone familiar with lambda calculus will immediately spot the flaw in the preceding proposal, namely that the substitution process that is used in β-conversion is carefully stated to avoid such 'accidental binding' of variables (by renaming bound variables, wherever required). We will instead use a special variant of substition which specifically does not act to avoid accidental binding, notated _[_//_] (e.g. $t[s//v]$ to indicate substitution of s for v in t). Note that the assignment of term variables in the approach in general is such that other cases of 'accidental binding' (i.e. beyond those that we want) will not occur. Incorporating this idea, we arrive at the following (final) version of the elimination rule

$$\frac{\phi:\text{A}\multimap(\text{B}:\alpha):\lambda v.a \qquad \psi:\text{B}:b}{\pi:\text{A}:a[b//v]} \qquad \begin{array}{l}\pi=\phi\uplus\psi\\ \alpha\subset\psi\end{array}$$

Note that the form of the rule requires the implicational formula that it operates upon to be of a certain form, i.e. involving an abstraction ($\lambda v.a$). This requirement is met by all implicationals, (as a side effect of the compilation process.

A precise statement of the compilation procedure (τ) is given in Figure 1. This takes a sequent $\Gamma \Rightarrow$ A:x as input, where every left and right hand side formula is labelled with a unique variable, and returns a structure $\langle\Delta,(\phi:G:u)\rangle$, where Δ is a set of indexed first order formulae, ϕ is the full

[5] Note the requirement that α is a *proper* subset of ψ, which will have the consequence that other assumptions must also contribute to deriving the argument B. This will block a derivation of the linear logically valid X⊸(Y⊸Y) ⇒ X. However, this move accords with general categorial practice, where it is standard to require that each deduction rests on at least one assumption. The alternative regime is easily achieved, by making the condition $\alpha \subseteq \psi$.

$\tau(X_1 : x_1, \ldots, X_n : x_n \Rightarrow X_0 : x_0) = \langle \Delta, (\phi : G : u) \rangle$
where $i_0, \ldots, i_n$ fresh indices
$\mathsf{neg}(i_0 : X_0 : x_0) = (i_0 : G : u) \uplus \Gamma$
$\Delta = \Gamma \cup \mathsf{pos}(i_1 : X_1 : x_1)$
$\cup \ldots$
$\cup \mathsf{pos}(i_n : X_n : x_n)$
$\phi = \mathsf{indices}(\Delta)$.

$\mathsf{pos}(i : X : t) = (i : X : t)$ where X atomic.
$\mathsf{pos}(i : X_1 \circ\!\!- Y_1 : t) = (i : X_2 \circ\!\!- (Y_2 : \phi) : \lambda u.s)$
$\cup \Gamma \cup \Delta$
where $\mathsf{neg}(i : Y_1 : v) = (i : Y_2 : u) \uplus \Gamma$
(v a fresh variable)
$\mathsf{pos}(i : X_1 : (tv)) = (i : X_2 : s) \uplus \Delta$
$\phi = \mathsf{indices}(\Gamma)$.

$\mathsf{neg}(i : X : v) = (i : X : v)$ where X atomic.
$\mathsf{neg}(i : X_1 \circ\!\!- Y_1 : u) = (i : X_2 : w) \cup \Gamma \cup \Delta$
where $u := \lambda v.x$ (v, x fresh variables)
$\mathsf{neg}(i : X_1 : x) = (i : X_2 : w) \uplus \Gamma$
$\mathsf{pos}(j : Y_1 : v) = \Delta$ (j a fresh index).

Figure 1: The compilation procedure

set of indices, G is an atomic goal type, and u a variable. Let Δ^* denote the result of closing Δ under the elimination rule. The sequent is proven *iff* $(\phi : G : u) \in \Delta^*$ for some assignment of a value to u. Under that assignment, the original right hand side variable x will return a complete proof term for the implicit proof of the original sequent. Note that the proof terms so produced have a form which corresponds, under the Curry-Howard isomorphism, to normal form deductions (as defined earlier).

A simple example. Compilation of the sequent:
$\mathrm{X} \circ\!\!- (\mathrm{Y} \circ\!\!- \mathrm{Z}) : x$, $\mathrm{Y} \circ\!\!- \mathrm{W} : y$, $\mathrm{W} \circ\!\!- \mathrm{Z} : w \Rightarrow \mathrm{X} : v$
yields the goal specification $(\{i, j, k, l\} : \mathrm{X} : v)$ and formulae 1-4, with formulae 5-7 arising under combination. Formula 7 meets the goal specification, so the initial sequent is proven, with proof term $x(\lambda z.y(wz))$ returned.

1. $i : \mathrm{X} \circ\!\!- (\mathrm{Y} : \{j\}) : \lambda u.x(\lambda z.u)$
2. $j : \mathrm{Z} : z$
3. $k : \mathrm{Y} \circ\!\!- \mathrm{W} : \lambda u.yu$
4. $l : \mathrm{W} \circ\!\!- \mathrm{Z} : \lambda u.wu$
5. $\{j, l\} : \mathrm{W} : wz$ [2+4]
6. $\{j, k, l\} : \mathrm{Y} : y(wz)$ [3+5]
7. $\{i, j, k, l\} : \mathrm{X} : x(\lambda z.y(wz))$ [1+6]

The indexed first-order formulae generated by the compilation procedure can be processed using precisely the same algorithm as that described above for handling formulae of the first-order fragment, with precisely the same benefit, i.e. avoiding recomputation of partial results.

Some efficiency questions arise. Imagine a Prolog implementation of the method, with indexed formulae being stored as facts ('edges') in the Prolog database. An important overhead will arise when adding an agenda item to the database from locating those formula already there that the current formula can combine with, i.e. if we must separately access every formula already stored to evaluate if indexation requirements are satisfied, and combination possible. Note firstly that, since compiled formulae are all first-order, if we are adding an atomic formula we need only look to stored implicational formulae for possible combinations, and vice versa. This is easily achieved. The problem of evaluating indexation requirements can be eased by using a *bit-vector* encoding of index sets. The compilation process will return a full set I of the unique indices assigned to any formulae. If we impose an arbitrary order over the elements of this set, we can then encode the extension of any index set we encounter using an n-place bit-vector, where n is the cardinality of I, i.e. if some index set contains the ith element of (ordered) I, then the ith element of its bit-vector is 1, otherwise 0. It is useful to store fully specified bit-vectors with atomic formulae, specifying their index set. For implicational formula, however, it is useful to store a bit-vector encoding its *requirements* for an appropriately indexed argument, i.e. with 0s instantiated for the elements of the implicational's own index set (to enforce disjointness of index sets), and with 1s appearing for those indices that it requires have been involved in deriving the argument. Other positions will be filled with anonymous variables. The bit-vectors for an implicational and an atomic formula will match just in case they are permitted to combine, according to indexation requirements. (The one shortfall here is that the method allows the implicational to specify that certain indices are a subset of those of the argument, but not that they are a *proper* subset thereof.) By storing such vectors with formulae in the database, indexation requirements can be checked by the process of matching to the database, so that only appropriate entries are brought out for further examination.

4 Labelling and Lambek Calculus

As discussed in the introduction, the above method is proposed as a general method for parsing categorial systems, via a tranformation of formulae from the relevant system to linear formulae. Such translation should induce labelling that imports the constraints of the original weaker logic. In that case, although we employ a general method for implicational linear deduction, the results we derive will be all and only those that reflect validity of the weaker system. I will illustrate this idea by considering one of two such translation methods described by Morrill (1995). This method is based on a relational algebraic model for **L** (van Benthem, 1991), which interprets types as relations on some set V (intuitively, pointal

string positions), i.e. sets of ordered pairs from $V \times V$ (intuitively, strings identified by delimiting points):

$$D(A\backslash B) = \{\langle v_2, v_3\rangle \mid \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\}$$

$$D(B/A) = \{\langle v_1, v_2\rangle \mid \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\}$$

Morrill specifies polar translation functions, which convert Lambek types that are marked for position ('span') to labelled linear formulae. The translation functions are identity functions on atomic formulae, and for complex formulae are defined mutually as follows (where each superscript p stands for one of the functions, with $\bar{p}$ indicating the complementary function to p):

$$\frac{i-k : B^p \quad \multimap \quad i-j : A^{\bar{p}}}{j-k : A\backslash B^p}$$ where i is a new variable/constant as p is $+/-$

$$\frac{i-k : B^p \quad \multimap \quad j-k : A^{\bar{p}}}{i-j : B/A^p}$$ where k is a new variable/constant as p is $+/-$

A sequent $B_1, \ldots, B_n \Rightarrow$ A is translated as: $0-1 : B_1^+, \ldots, (n-1)-n : B_n^+ \Rightarrow 0-n : A^-$
For example, X/(Y/Z), Y/W, W/Z ⇒ X translates to give the following linear formulae (where i, j, l are variables, and **k** a constant):

Database: $(0-i$:X)$\multimap((2-\mathbf{k}$:Y)$\multimap(i-\mathbf{k}$:Z))
$(1-j$:Y)$\multimap(2-j$:W)
$(2-l$:W)$\multimap(3-l$:Z)
Goal: $(0-3$:X)

Such linear formulae can be used with any linear deduction method, given the (trivial) additional task of unifying variables and constants in the string position labels. Note that for cases that are not **L** valid, but where the translation is linear logically valid, deduction will fail due to unification failure for string position labels. A minor complication arises for using this approach with the compilation-chart method described above. For example, the higher-order formula would compile to two indexed formulae:

a:$(0-i$:X)$\multimap(2-\mathbf{k}$:Y):$\lambda y.x(\lambda z.y)$
b:$(i-\mathbf{k}$:Z):z

Note that the string position variable i appears in both resulting formulae. For an overall deduction employing these two formulae to be correct, the binding of the two instances of i must be consistent. However, we cannot simply employ a global binding context since the chart method should be able to return alternative proofs of the same theorem, and such alternative proofs will typically induce distinct (but internally consistent) bindings over string position variables. Variable bindings must instead be handled locally, i.e. each formula in the database will carry with it a context indicating bindings that have been made in its derivation. Where two formula are combined, their contexts are merged (and must be consistent).

References

van Benthem, J. 1991. *Language in Action: Categories, Lamdas and Dynamic Logic.* Studies in Logic and the Foundations of Mathematics, vol 130, North-Holland, Amsterdam.

Gabbay, D.M. 1994. *Labelled deductive systems. Part I: Foundations.* Oxford University Press (to appear). First draft 1989, current draft May 1994.

Hendriks, H. 1992. 'Lambek Semantics: normalisation, spurious ambiguity, partial deduction and proof nets', *Proc. of Eighth Amsterdam Colloquium*, ILLI, University of Amsterdam.

Hepple, M. 1990. 'Normal form theorem proving for the Lambek calculus', *Proc. of COLING-90.*

Hepple, M. 1992. ' Chart Parsing Lambek Grammars: Modal Extensions and Incrementality', *Proc. of COLING-92.*

Hepple, M. 1995. 'Mixing Modes of Linguistic Description in Categorial Grammar', *Proceedings EACL-7*, Dublin.

König, E. 1989, 'Parsing as natural deduction', *Proc. of ACL-25.*

König, E. 1990, 'The complexity of parsing with extended categorial grammars', *Proc. of COLING-90.*

Lambek, J. 1958. The mathematics of sentence structure. *American Mathematical Monthly* **65**.

Lloré, F.X. & Morrill, G. 1995. 'Difference Lists and Difference Bags for Logic Programming of Categorial Deduction', *Proc. of SEPLN XI*, Duesto.

Moortgat, M. 1992. 'Labelled deductive systems for categorial theorem proving', *Proc. of Eighth Amsterdam Colloquium*, ILLI, University of Amsterdam.

Moortgat, M. & Oehrle, R. 1994. 'Adjacency, dependency and order'. *Proc. of Ninth Amsterdam Colloquium.*

Moortgat, M. & Morrill, G. 1991. 'Heads and Phrases: Type Calculus for Dependency and Constituency.' To appear: *Journal of Language, Logic and Information.*

Morrill, G. 1994. *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Publishers, Dordrecht.

Morrill, G. 1995. 'Higher-order Linear Logic Programming of Categorial Dedution', *Proc. of EACL-7*, Dublin.

Prawitz, D. 1965. *Natural Deduction: a Proof Theoretical Study*, Almqvist and Wiksell, Uppsala.

Roorda, D. 1991. *Resource Logics: Proof Theoretical Investigations.* Ph.D. Dissertation, Amsterdam.