Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99). Maryland, June 1999, pp465-475.

An Earley-style Predictive Chart Parsing Method for Lambek Grammars

Mark Hepple

Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK [hepple@dcs.shef.ac.uk]

Abstract

We present a new chart parsing method for Lambek grammars, inspired by a method for D-Tree grammar parsing. The formulae of a Lambek sequent are firstly converted into rules of an indexed grammar formalism, which are used in an Earley-style predictive chart algorithm. The method is non-polynomial, but performs well for practical purposes — much better than previous chart methods for Lambek grammars.

1 Introduction

We present a new chart parsing method for Lambek grammars. The starting point for this work is the observation, in (Hepple, 1998), of certain similarities between categorial grammars and the D-Tree grammar (DTG) formalism of Rambow *et al.* (1995a). On this basis, we have explored adapting the DTG parsing approach of Rambow *et al.* (1995b) for use with the Lambek calculus. The resulting method is one in which the formulae of a Lambek sequent that is to be proven are first converted to produce rules of a formalism which combines ideas from the *multiset-valued linear indexed grammar* formalism of Rambow (1994), with the Lambek calculus span labelling scheme of Morrill (1995), and with the *first-order compilation* method for categorial parsing of Hepple (1996). The resulting 'grammar' is then parsed using an Earley-style predictive chart algorithm which is adapted from Rambow *et al.* (1995b).

2 The Lambek Calculus

We are concerned with the implicational (or 'product-free') fragment of the associative Lambek calculus (Lambek, 1958). A *natural deduction* formulation is provided by the following rules of *elimination* and *introduction*, which correspond to steps of functional application and abstraction, respectively (as the term labelling reveals). The rules are sensitive to the order of assumptions. In the [/I] (resp. [\I]) rule, $[B]$ indicates a discharged or withdrawn assumption, which is required to be the rightmost (resp. leftmost) of the proof.

$$\frac{A/B : a \quad B : b}{A : (ab)}/\mathrm{E} \qquad \frac{B : b \quad B\backslash A : a}{A : (ab)}\backslash\mathrm{E}$$

$$\frac{\begin{array}{c}\vdots\\ A : a\end{array}}{A/B : \lambda v.a}/\mathrm{I}\ [B : v] \qquad \frac{\begin{array}{c}\vdots\\ A : a\end{array}}{B\backslash A : \lambda v.a}\backslash\mathrm{I}\ [B : v]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{(which)}}{\text{rel/(s/np)}} \quad \dfrac{\dfrac{\dfrac{\text{(mary)}}{\text{np}} \quad \dfrac{\dfrac{\text{(ate)}}{\text{(np\s)/np}} \quad \text{[np]}}{\text{np\s}}/\mathrm{E}}{\text{s}}\backslash\mathrm{E}}{\text{s/np}}/\mathrm{I}}{\text{rel}}/\mathrm{E}$$

The above proof illustrates 'hypothetical reasoning', i.e. the presence of additional assumptions ('hypotheticals') in proofs that are subsequently discharged. It is because of this phenomenon that standard chart methods are inadequate for the Lambek calculus — hypotheticals don't belong at any position on the single ordering over lexical categories by which standard charts are organised.[1] The previous chart methods for the Lambek calculus deal with this problem in different ways. The method of König (1990, 1994) places hypotheticals on separate 'minicharts' which can attach into other (mini)charts where combinations are

[1] In effect, hypotheticals belong on additional suborderings, which can connect into the main ordering of the chart at various positions, generating a branching, multi-dimensional ordering scheme.

possible. The method requires rather complicated book-keeping. The method of Hepple (1992) avoids this complicated book-keeping, and also rules out some useless subderivations allowed by König's method, but does so at the cost of computing a representation of all the possible category sequences that might be tested in an exhaustive sequent proof search. Neither of these methods exhibits performance that would be satisfactory for practical use.[2]

3 Some Preliminaries

3.1 First-order Compilation for Categorial Parsing

Hepple (1996) introduces a method of *first-order compilation* for implicational linear logic, to provide a basis for efficient theorem proving of various categorial formalisms. Implicational linear logic is similar to the Lambek calculus, except having only a single non-directional implication $\multimap$. The idea of first-order compilation is to eliminate the need for hypothetical reasoning by simplifying higher-order formulae (whose presence requires hypothetical reasoning) to first-order formulae. This involves excising the subformulae that correspond to hypotheticals, leaving a first-order residue. The excised subformulae are added as additional assumptions. For example, a higher-order formula $(Z \multimap Y) \multimap X$ simplifies to $Z + (Y \multimap X)$, allowing proof (a) to be replaced by (b):

(a)
$$\dfrac{\dfrac{\dfrac{[Z] \quad Z \multimap W}{W} \quad W \multimap Y}{Y}}{\dfrac{Z \multimap Y}{X}} \quad (Z \multimap Y) \multimap X$$

(b)
$$\dfrac{\dfrac{\dfrac{Z \quad Z \multimap W}{W} \quad W \multimap Y}{Y}}{X} \quad Y \multimap X$$

The method faces two key problems: avoiding invalid deduction and getting an appropriate semantics for the combination. To avoid invalid deduction, an indexing scheme is used to ensure that a hypothetical *must* be used to derive the argument of the residue functor from which was excised (e.g. Z must be used to derive the argument Y of $Y \multimap X$, a condition satisfied in proof (b). To get the same semantics with compilation as without, the semantic effects of the introduction rule are compiled into the terms of the formulae produced, e.g. $(Z \multimap Y) \multimap X : w$ gives $Z : z$ plus $Y \multimap X : \lambda u.w(\lambda z.u)$. Terms are combined, not using standard application/β-reduction, but rather an operation $\lambda x.g \ + \ h \ \Rightarrow \ g[h/\!/x]$ where a variant of substitution is used that allows 'accidental' variable capture. Thus when $Y \multimap X$ combines with its argument, whose derivation includes Z, the latter's variable becomes bound, e.g. $\lambda u.w(\lambda z.u) \ + \ x(yz) \ \Rightarrow \ w(\lambda z.x(yz))$

3.2 Multiset-valued Linear Indexed Grammar

Rambow (1994) introduces the *multiset-valued linear indexed grammar* formalism ({}-LIG). Indices are stored in an unordered multiset representation (c.f. the *stack* of conventional linear indexed grammar). The contents of the multiset at any mother node in a tree is *distributed* amongst its daughter nodes in a linear fashion, i.e each index is passed to precisely one daughter. Rules take the form $A_0[m_0] \rightarrow A_1[m_1] \ldots A_n[m_n]$. The multiset of indices m_0 are required to be present in, and are *removed from*, the multiset context of the mother node in a tree. For each daughter A_i, the indices m_i are *added into* whatever other indices are inherited to that daughter. Thus, a rule $A[] \rightarrow B[1]\ C[]$ (where [] indicates an empty multiset) can *license* the use of a rule $D[1] \rightarrow \alpha$ within the derivation of its daughter B[1], and so the indexing system allows the encoding of *dominance* relations.

4 A New Chart Parsing Method for Lambek Grammars

4.1 Lambek to SLMG Conversion

The first task of the parsing approach is to convert the antecedent formulae of the sequent to be proved into a collection of rules of a formalism I call Span Labelled Multiset Grammar (SLMG). For digestibility, I will present the conversion process in three stages. (I will assume

[2] Morrill (1996) provides a somewhat different *tabular* method for Lambek parsing within the proof net deduction framework, in an approach where proof net checking is made by unifying labels marked on literals. The approach tabulates *MGU*'s for the labels of contiguous subsegments of a proof net.

Method:

$$
\begin{array}{rcl}
(A{:}(i{-}j))^p & = & A{:}(i{-}j) \quad \text{where } A \text{ atomic} \\
(A/B{:}(h{-}i))^p & = & (A{:}(h{-}j))^p \ / \ (B{:}(i{-}j))^{\bar{p}} \\
(B\backslash A{:}(h{-}i))^p & = & (B{:}(j{-}h))^{\bar{p}} \ \backslash \ (A{:}(j{-}i))^p
\end{array}
\quad \Big\} \quad \text{where } j \text{ is a new variable/constant as } p \text{ is } +/-
$$

Example:

$$
\begin{array}{rcl}
(X/(Y/Z){:}(0{-}1))^+ & = & X{:}(0{-}h)/(Y{:}(1{-}\mathbf{k})/Z{:}(h{-}\mathbf{k})) \\
(W{:}(1{-}2))^+ & = & W{:}(1{-}2) \\
((W\backslash Y)/Z{:}(2{-}3))^+ & = & (W{:}(i{-}2)\backslash Y{:}(i{-}j))/Z{:}(3{-}j)
\end{array}
$$

Figure 1: Phase 1 of conversion (span labelling)

that in any sequent $\Gamma \Rightarrow A$ to be proved, the succedent A is atomic. Any sequent not in this form is easily converted to one, of equivalent theoremhood, which is.)

Firstly, directional types are labelled with span information using the labelling scheme of Morrill (1995) (which is justified in relation to relational algebraic models for the Lambek calculus (van Benthem, 1991)). An antecedent X_i in $X_1 \ldots X_n \Rightarrow X_0$ has basic span $(h{-}i)$ where $h = (i-1)$. The labelled formula is computed from $(X_i{:}(h{-}i))^+$ using the polar translation functions shown in Figure 1 (where $\bar{p}$ denotes the complementary polarity to p).[3] As an example, Figure 1 also shows the results of converting the antededents of $X/(Y/Z), W, (W\backslash Y)/Z \Rightarrow X$ (where $\mathbf{k}$ is a constant and i, j variables).[4]

The second stage of the conversion is adapted from the first-order compilation method of Hepple (1996), discussed earlier, modified to handle directional formulae and using a modified indexation scheme to record dependencies between residue formulae and excised hypotheticals (one where *both* the residue and hypothetical record the dependency). For this procedure, the 'atomic type plus span label' units that result from the previous stage are treated as atomic units. The procedure τ is defined by the cases shown in Figure 2 (although the method is perhaps best understood from the example also shown there). Its input is a pair $\langle T, t \rangle$, T a span labelled formula, t its associated term.[5]

This procedure simplifies higher-order formulae to first-order ones in the manner already discussed, and records dependencies between hypothetical and residue formulae using the indexing scheme. Assuming the antecedents of our example $X/(Y/Z), W, (W\backslash Y)/Z \Rightarrow X$, to have terms s_1, s_2, s_3 respectively, compilation yields results as in the example in Figure 2. The higher-order $X/(Y/Z)$ yields two output formulae: the main residue X/Y and the hypothetical Z, with the dependency between the two indicated by the common index 1 in the argument index set of the former and the principal index set of the latter. The empty sets elsewhere indicate the absence of such dependencies.

The final stage of the conversion process converts the results of the second phrase into SLMG productions. The method will be explained by example. For a functor such as $B\backslash(((A\backslash X)/D)/C)$, we can easily project the sequence of arguments it requires:

[3] The constants produced in the translation correspond to 'new' string positions, which make up the additional suborderings on which hypotheticals are located. The variables produced in the translation become instantiated to some string constant during an analysis, fixing the position at which an additional subordering becomes 'attached to' another (sub)ordering.

[4] The idea of implementing categorial grammar as a non-directional logic, but associating atomic types with string position pairs (i.e. spans) to handle word order, is used in Pareschi (1988), although in that approach all string positions instantiate to values on a single ordering (i.e. integers $0-n$ for a string of length n), which is not sufficient for Lambek calculus deductions.

[5] Note that the "+" of $(A + \Gamma)$ in $(\tau 0)$ simply pairs together the single compiled formula A with the set Γ of compiled formulae, where A is the main residue of the input formula and Γ its derived hypotheticals.

Method:

$(\tau 0)$ $\quad \tau(\langle \mathrm{T}, t\rangle) = A \cup \Gamma$ $\quad$ where $\tau(\langle \emptyset, \mathrm{T}, t\rangle) = A + \Gamma$

$(\tau 1a)$ $\quad \tau(\langle m, X/Y, t\rangle) = \tau(\langle m, X/(Y{:}\emptyset), t\rangle)$ $\quad$ where Y has no index set

$(\tau 1b)$ $\quad$ as for $(\tau 1a)$ modulo directionality of connective

$(\tau 2a)$ $\quad \tau(\langle m, X_1/(Y{:}m_1), t\rangle) = \langle m, X_2/(Y{:}m_1), \lambda v.s\rangle + \Gamma$
where Y atomic, $\tau(\langle m, X_1, (tv)\rangle) = \langle m, X_2, s\rangle + \Gamma$, v a fresh variable

$(\tau 2b)$ $\quad$ as for $(\tau 2a)$ modulo directionality of connective

$(\tau 3a)$ $\quad \tau(\langle m, X/((Y/Z){:}m_1), t\rangle) = A + (B \cup \Gamma \cup \Delta)$
where w, v fresh variables, i a fresh multiset index, $m_2 = i \cup m_1$
$\tau(\langle m, X/(Y{:}m_2), \lambda w.t(\lambda v.w)\rangle) = A + \Gamma, \quad \tau(\langle i, Z, v\rangle) = B + \Delta$

$(\tau 3b)$–$(\tau 3d)$ $\quad$ as for $(\tau 3a)$ modulo directionality of connectives

Example:

$$\tau(\langle X{:}(0{-}h)/(Y{:}(1{-}\mathbf{k})/Z{:}(h{-}\mathbf{k})), s_1\rangle) = \left\{ \begin{array}{c} \langle \emptyset, X{:}(0{-}h)/(Y{:}(1{-}\mathbf{k}){:}\{1\}), \lambda u.s_1(\lambda z.u)\rangle \\ \langle \{1\}, Z{:}(h{-}\mathbf{k})), z\rangle \end{array} \right\}$$

$$\tau(\langle W{:}(1{-}2), s_2\rangle) = \langle \emptyset, W{:}(1{-}2), s_2\rangle$$

$$\tau(\langle (W{:}(i{-}2)\backslash Y{:}(i{-}j))/Z{:}(3{-}j), s_3\rangle) = \langle \emptyset, ((W{:}(i{-}2){:}\emptyset)\backslash Y{:}(i{-}j))/(Z{:}(3{-}j){:}\emptyset), \lambda v \lambda w.(s_3\ v\ w)\rangle$$

Figure 2: Phase 2 of conversion (first-order compilation)

$A, B, B\backslash(((A\backslash X)/D)/C), C, D \Rightarrow X$. If the functor was the lexical category of a word w, it might be viewed as fulfilling a role akin to a PS rule such as $X \rightarrow A\ B\ w\ C\ D$. For the present approach, with explicit span labelling, there is no need to include a rhs element to mark the position of the functor (or word) itself, so the corresponding production would be more akin to $X \rightarrow A\ B\ C\ D$. For an atomic formula, the corresponding production will have an empty rhs, e.g. $A \rightarrow \langle\rangle$.[6]

The left and right hand side units of SLMG productions all take the form $A[m](i{-}j)$, where A is an atomic type, m is a set of indices (if m is empty, the unit may be written $A[\,](i{-}j)$), and $(i{-}j)$ a span label. For a formula $\langle m, T, t\rangle$ resulting after first-order compilation, the rhs elements of the corresponding production correspond to the arguments (if any) of T, whereas its lhs combines the result type (plus span) of T with the multiset m. For our running example $X/(Y/Z), W, (W\backslash Y)/Z \Rightarrow X$, the formulae resulting from the second phase (by first-order compilation) give rise to productions as shown in Figure 3. The associated semantic term for each rule is intended to be applied to the semantics if its daughters in their left-to-right order (which may require some reordering of the outermost lambdas c.f. the terms of the first-order formulae, e.g. as for the last rule).

A sequent $X_1 \ldots X_n \Rightarrow X_0$ is proven if we can build a SLMG tree with root $X_0[\,](0{-}n)$ in which the SLMG rules derived from the antecedents are each used precisely once, and which induces a consistent binding over span variables. For our running example, the required derivation, shown below, yields the correct interpretation $s_1(\lambda z.s_3\ z\ s_2)$. Note that 'linear resource use', i.e. that each rule must be used precisely

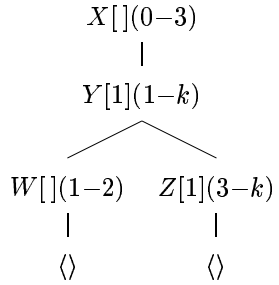
[6] Note that $\langle\rangle$ is used rather than ϵ to avoid the suggestion of the empty *string*, which it is not — matters to do with the 'string' are handled solely within the span labelling. This point is reinforced by observing that the 'string language' generated by a collection SLMG productions will consist only of (nonempty) sequences of $\langle\rangle$'s. The real import of a SLMG derivation is not its terminal yield, but rather the instantiation of span labels that it induces (for string matters), and its structure (for semantic matters).

Example:

$$\langle \emptyset, X{:}(0{-}h)/(Y{:}(1{-}\mathbf{k}){:}\{1\}), \lambda u.s_1(\lambda z.u)\rangle \Rightarrow X[](0{-}h) \rightarrow Y[1](1{-}\mathbf{k}) : \lambda u.s_1(\lambda z.u)$$
$$\langle \{1\}, Z{:}(h{-}\mathbf{k})), z\rangle \Rightarrow Z[1](h{-}\mathbf{k}) \rightarrow \langle\rangle : z$$
$$\langle \emptyset, W{:}(1{-}2), s_2\rangle \Rightarrow W[](1{-}2) \rightarrow \langle\rangle : s_2$$
$$\langle \emptyset, ((W{:}(i{-}2){:}\emptyset)\backslash Y{:}(i{-}j))/(Z{:}(3{-}j){:}\emptyset), \lambda v\lambda w.(s_3\, v\, w)\rangle$$
$$\Rightarrow Y[](i{-}j) \rightarrow W[](i{-}2)\ \ Z[](3{-}j) : \lambda w\lambda v.(s_3\, v\, w)$$

Figure 3: Phase 3 of conversion (converting to SLMG productions)

once, is enforced by the span labelling scheme and does not need to be separately stipulated. Thus, the span $(0{-}n)$ is marked on the root of the derivation. To bridge this span, the main residues of the antecedent formulae must all participate (since each 'consumes' a basic subspan of the main span) and they in turn require participation of their hypotheticals via the indexing scheme.



4.2 The Earley-style Parsing Method

The chart parsing method to be presented is derived from the Earley-style DTG parsing method of Rambow *et al.* (1995), and in some sense both simplifies and complicates their method. In effect, we abstract from their method a simpler one for Earley-style parsing of {}-LIG (which is a simpler formalism than the Linear Prioritized Multiset Grammar (LPMG) into which they compile DTG), and then extend this method to handle the span labelling of SLMG. A key differences of the new approach as compared to standard chart methods is that the usual external notion of span is dispensed with, and the combination of edges is instead regimented in terms of the explicit span labelling of categories in rules. The unification of span labels requires edges to carry explicit binding information for span variables. We use R to denote the set of rules derived from the sequent, and E the set of edges in the chart. The general form of edges is: $((m_1, m_2), \theta, r, (\mathcal{A} \rightarrow \Gamma \bullet \Delta))$ where $(\mathcal{A} \rightarrow \Gamma, \Delta) \in R$, θ is a substitution over span variables, r is a *restrictor set* identifying span variables whose values are required non-locally (explained below), and m_1, m_2 are multisets. In a {}-LIG or SLMG tree, there is no restriction on how the multiset indices associated with any non-terminal node can be distributed amongst its daughters. Rather than cashing out the possible distributions as alternative edges in the predictor step, we can instead, in effect, 'thread' the multiset through the daughters, i.e. passing the entire multiset down to the first daughter, and passing any that are not used there on to the next daughter, and so on. For an edge $((m_1, m_2), \theta, r, (\mathcal{A} \rightarrow \Gamma \bullet \Delta))$, m_1 corresponds to the multiset context at the time the ancestor edge with dotted rule $(\mathcal{A} \rightarrow \bullet\Gamma\Delta)$ was introduced, and m_2 is the current multiset for passing onto the daughters in Δ. We call m_1 the *initial* multiset and m_2 the current multiset.

The chart method employs the rules shown in Figure 4. We shall consider each in turn.

Initialisation:
The rule recorded on the edge in this chart rule is not a real one (i.e. $\notin R$), but serves to drive the parsing process via the prediction of edges for rules that can derive $X_0[](1{-}n)$. A successful proof of the sequent is shown if the completed chart contains an inactive edge for the special goal category, i.e. there is some edge $((\emptyset, \emptyset), \emptyset, \emptyset, (GOAL[](*{-}*) \rightarrow \Lambda\bullet)) \in E$

Prediction:
The current multiset of the predicting edge is passed onto the new edge as its initial multiset. The latter's current multiset (m_6) may differ from its initial one due either to the removal of an index to license the new rule's use (i.e. if

Initialisation:

if the initial sequent is $X_1 \ldots X_n \Rightarrow X_0$

then $((\emptyset,\emptyset),\emptyset,\emptyset,(GOAL[\,](*{-}*) \rightarrow \bullet X_0[\,](1{-}n))) \in E$

Prediction:

if $((m_1,m_2),\theta_1,r_1,(A[m_3](e{-}f) \rightarrow \Gamma \bullet B[m_4](g{-}h),\Delta)) \in E$

and $(B[m_5](i{-}j) \rightarrow \Lambda) \in R$

then $((m_2,m_6),\theta_2,r_2,(B[m_5](g{-}(h\theta)) \rightarrow \bullet(\Lambda\theta))) \in E$

where $\theta = \theta_1 + MGU((g{-}h),(i{-}j))$; $m_5 \subseteq m_2 \cup m_4$; $m_6 = (m_2 \cup m_4) - m_5$;
$r_2 = \mathsf{nlv}(m_2 \cup m_4)$; $\theta_2 = \theta/(r_2 \cup \mathsf{daug1nlv}(\Lambda))$

Completer:

if $((m_1,m_2),\theta_1,r_1,(A[m_3](f{-}g) \rightarrow \Gamma \bullet B[m_4](i{-}h),\Delta)) \in E$

and $((m_2,m_5),\theta_2,r_2,(B[m_6](i{-}j) \rightarrow \Lambda\bullet)) \in E$

then $((m_1,m_5),\theta_3,r_1,(A[m_3](f{-}(g\theta)) \rightarrow \Gamma, B[m_4](i{-}j) \bullet (\Delta\theta))) \in E$

where $\theta = \theta_1 + \theta_2 + MGU(h,j)$; $m_5 \subseteq m_2$; $m_6 \subseteq m_2 \cup m_4$;
$\theta_3 = \theta/(r_1 \cup \mathsf{daug1nlv}(\Delta))$

Figure 4: Chart rules

m_5 is non-empty), or to the addition of indices from the predicting edge's next rhs unit (i.e. if m_4 is non-empty). (Note the 'sloppy' use of set, rather than explicitly multiset, notation. The present approach is such that the same index should never appear in both of two unioned sets, so there is in practice little difference.)

The line $\theta = \theta_1 + MGU((g{-}h),(i{-}j))$ checks that the corresponding span labels unify, and that the resulting MGU can consistently augment the binding context of the predicting edge. This augmented binding is used to instantiate span variables in the new edge where possible. It is a characteristic of this parsing method, with top-down left-to-right traversal and associated propagation of span information, that the left span index of the next daughter sought by any active edge is guarenteed to be instantiated, i.e. g above is a constant.

Commonly the variables appearing in SLMG rules have only local significance and so their substitutions do not need to be carried around with edges. For example, an active edge might require two daughters $B[\,](g{-}h)$ $C[\,](h{-}i)$. A substitution for h that comes from combining with an inactive edge for $B[\,](g{-}h)$ can be immediately applied to the next daughter $C[\,](h{-}i)$, and so does not need to be carried explicitly in the binding of the resulting edge. However, a situation where two occurrences of a variable appear in different rules may arise as a result of first-order compilation, which will sometimes (but not always) separate a variable occurrence in the hypothetical from another in the residue. For the rule set of our running example, we find an occurrence of h in both the first and second rule (corresponding to the main residue and hypothetical of the initial higher-order functor). The link between the two rules is also indicated by the indexing system. It turns out that for each index there is at most one variable that may appear in the two rules linked by the index. The identity of the 'non-local variables' that associate with each index can be straightforwardly computed off the SLMG grammar (or during the conversion process).

The function nlv returns the set of non-local variables that associate with a multiset of indices. The line $r_2 = \mathsf{nlv}(m_2 \cup m_4)$ computes the set of variables whose values may need to

be passed non-locally, i.e. from the predicting edge down to the predicted edge, or from an inactive edge that results from combination of this predicted edge up to the active edge that consumes it. This 'restrictor set' is used in reducing the substitution θ to cover only those variables whose values need to be stored with the edge. The only case where a substitution needs to be retained for variable that is not in the restrictor set arises regarding the next daughter it seeks. For example, an active edge might require two daughters $B[](g{-}h) \quad C[1](k{-}i)$, where the second's index links it to a hypothetical with span $(k{-}h)$. Here, a substitution for h from a combination for the first daughter cannot be immediately applied and so should be retained until a combination is made for the second daughter. The function call $\mathsf{daug1nlv}(\Lambda)$ returns the set of non-local variables associated with the multiset indices of the next daughter in Λ (or the empty set if Λ is empty). There may be at most one variable in this set that appears in the substitution θ. The line $\theta_2 = \theta/(r_2 \cup \mathsf{daug1nlv}(\Lambda))$ reduces the substitution to cover only the variables whose values need to be stored. Failing to restrict the substitution in this way undermines the compaction of derivations by the chart, i.e. so that we find edges in the chart corresponding to the same subderivation, but which are not recognised as such during parsing due to them recording incompatible substitutions.

Completer:
Recall from the prediction step that the predicted edge's current multiset may differ from its initial multiset due to the addition of indices from the predicting edge's next rhs unit (i.e. m_4 in the prediction rule). Any such added indices must be 'used up' within the subderivation of that rhs element which is realised by the combinations of the predicted edge. This requirement is checked by the condition $m_5 \subseteq m_2$.

The treatment of substitutions here is very much as for the prediction rule, except that both input edges contribute their own substitution. Note that for the inactive edge (as for all inactive edges), both components of the span $(i{-}j)$ will be instantiated, so we need only unify the right index of the two spans — the left indices can simply be checked for atomic identity. This observation is important to efficient implementation of the algorithm, for which most effort is in practice expended on the completer step. Active edges should be indexed (i.e. hashed) with respect to the (atomic) type and left span index of the next rhs element sought. For inactive edges, the type and left span index of the lhs element should be used. For the completer step when an active edge is added, we need only access inactive edges that are hashed on the same type/left span index to consider for combination, all others can be ignored, and *vice versa* for the addition of an inactive edge.

It is notable that the algorithm has no *scanning* rule, which is due to the fact that the positions of 'lexical items' or antecedent categories are encoded in the span labels of rules, and need no further attention. In the (Rambow *et al.*, 1995) algorithm, the scanning component also deals with epsilon productions. Here, rules with an empty rhs are dealt with by prediction, by allowing an edge added for a rule with an empty rhs to be treated as an inactive edge (i.e. we equate "$\langle\,\rangle\,\bullet$" and "$\bullet\,\langle\,\rangle$").

If the completed chart indicates a successful analysis, it is straightforward to compute the proof terms of the corresponding natural deduction proofs, given a record of which edges were produced by combination of which other edges, or by prediction from which rule. Thus, the term for a predicted edge is simply that of the rule in R, whereas a term for an edge produced by a completer step is arrived at by combining a term of the active edge with one for the inactive edge (using the special substitution operation that allows 'accidental binding' of variables, as discussed earlier). Of course, a single edge may compact multiple alternative subproofs, and so return multiple terms. Note that the approach has no problem in handling multiple lexical assignments, they simply result in multiple rules generated off the same basic span of the chart.

5 Efficiency and Complexity

The method is shown to be non-polynomial by considering a simple class of examples of the form $X_1, \ldots X_{n-1}, a \Rightarrow a$, where each X_i is a/(a/(a\a)). Each such X_i gives a hypothetical whose dependency is encoded by a multiset index. Examination of the chart reveals spans for which there are multiple edges, differing in their 'initial' multiset (and other ways), there being

$$x_0/a/(x_1/(a/a)), x_1/(x_2/(a/a)), x_2/(a/a), a/a, a/a, a/a, a/a, a/a, a \Rightarrow x_0$$

Figure 5: Example for comparison of methods

one for edge for each subset of the indices deriving from the antecedents $X_1, \ldots X_{n-2}$, i.e. giving $2^{(n-2)}$ distinct edges. This non-polynomial number of edge results in non-polynomial time for the completer step, and in turn for the algorithm as a whole. Hence, this approach does not resolve the open question of the polynomial time parsability of the Lambek calculus. Informally, however, these observations are suggestive of a possible locus of difficulty in achieving such a result. Thus, the hope for polynomial time parsability of the Lambek calculus comes from it being an ordered 'list-like' system, rather than an unordered 'bag-like' system, but in the example just discussed, we observe 'bag-like' behaviour in a compact encoding (the multiset) of the dependencies of hypothetical reasoning.

We should note that the DTG parsing method of (Rambow *et al.*, 1995), from which the current approach is derived, *is* polynomial time. This follows from the fact that their compilation applies to a preset DTG, giving rise to a fixed maximal set of distinct indices in the LPMG that the compilation generates. This fixed set of indices gives rise to a very large, but polynomial, worst-case upper limit on the number of edges in a chart, which in turn yields a polynomial time result. A key difference for the present approach is that our task is to parse arbitrary initial sequents, and hence we do not have the fixed initial grammar that is the basis of the Rambow *et al.* complexity result.

For practical comparison to the previous Lambek chart methods, consider the highly ambiguous artificial example shown in Figure 5, (which has six readings). König (1994) reports that a Prolog implementation of her method, running on a major workstation produces 300 edges in 50 seconds. A Prolog implementation of the current method, on a current major work station, produces 75 edges in less than a tenth of a second. Of course, the increase in computing power over the years makes the times not strictly comparable, but still a substantial speed up is indicated. The difference in the number of edges suggests that the König method is suboptimal in its compaction of alternative derivations.

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.

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

Mark Hepple. 1996. 'A Compilation-Chart Method for Linear Categorial Deduction.' *Proc. COLING–96*, Copenhagen.

Hepple, M. 1998. 'On Some Similarities Between D-Tree Grammars and Type-Logical Grammars.' *Proc. Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks.*

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

Esther König. 1994. 'A Hypothetical Reasoning Algorithm for Linguistic Analysis.' *Journal of Logic and Computation*, Vol. 4, No 1.

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

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

Morrill, G. 1996. 'Memoisation for Categorial Proof Nets: Parallelism in Categorial Processing.' Research Report LSI-96-24-R, Universitat Politécnica de Catalunya.

Pareschi, R. 1988. 'A Definite Clause Version of Categorial Grammar.' *Proc. 26th ACL.*

Rambow, O. 1994. 'Multiset-valued linear index grammars.' *Proc. ACL'94.*

Rambow, O., Vijay-Shanker, K. & Weir, D. 1995a. 'D-Tree Grammars.' *Proc. ACL–95.*

Rambow, O., Vijay-Shanker, K. & Weir, D. 1995b. 'Parsing D-Tree Grammars.' *Proc. Int. Workshop on Parsing Technologies.*