Proceedings of the ACL-97 / EACL-97 Joint Conference, Madrid, July 1997. (The 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics)

Maximal Incrementality in Linear Categorial Deduction

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

Abstract

Recent work has seen the emergence of a common framework for parsing categorial grammar (CG) formalisms that fall within the 'type-logical' tradition (such as the Lambek calculus and related systems), whereby some method of linear logic theorem proving is used in combination with a system of labelling that ensures only deductions appropriate to the relevant grammatical logic are allowed. The approaches realising this framework, however, have not so far addressed the task of incremental parsing — a key issue in earlier work with 'flexible' categorial grammars. In this paper, the approach of (Hepple, 1996) is modified to yield a linear deduction system that does allow flexible deduction and hence incremental processing, but that hence also suffers the problem of 'spurious ambiguity'. This problem is avoided via normalisation.

1 Introduction

A key attraction of the class of formalisms known as 'flexible' categorial grammars is their compatibility with an incremental style of processing, in allowing sentences to be assigned analyses that are fully or primarily left-branching. Such analyses designate many initial substrings of a sentence as interpretable constituents, allowing its interpretation to be generated 'on-line' as it is presented. Incremental interpretation has been argued to provide for efficient language processing, by allowing early filtering of implausible readings.[1]

This paper is concerned with the parsing of categorial formalisms that fall within the 'type-logical' tradition, whose most familiar representative is the associative Lambek calculus (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 in one system.[2] Many of these proposals employ a 'labelled deductive system' methodology (Gabbay, 1996), whereby types in proofs are associated with *labels* which record proof information for use in ensuring correct inferencing.

A common framework is emerging for parsing type-logical formalisms, which exploits the labelled deduction idea. Approaches within this framework employ a theorem proving method that is appropriate for use with linear logic, and combine it with a labelling system that restricts admitted deductions to be those of a weaker system. Crucially, linear logic stands above *all* of the type-logical formalisms proposed in the hierarchy of substructural logics, and hence linear logic deduction methods can provide a common basis for parsing all of these systems. For example, Moortgat (1992) combines a linear proof net method with labelling to provide deduction for several categorial systems. Morrill (1995) shows how types of the associative Lambek calculus may be translated to labelled implicational linear types, with deduction implemented via a version of SLD resolution. Hepple (1996) introduces a linear deduction method, involving compilation to first order formulae, which can be combined with various labelling disciplines. These approaches, however, are not directed toward incremental processing.

In what follows, we show how the method of (Hepple, 1996) can be modified to allow processing which has a high degree of incrementality. These modifications, however, give a system which suffers

[1] Within the categorial field, the significance of incrementality has been emphasised most notably in the work of Steedman, e.g. (Steedman, 1989).

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

the problem of 'derivational equivalence', also called 'spurious ambiguity', i.e. allowing multiple proofs which assign the same reading for some combination, a fact which threatens processing efficiency. We show how this problem is solved via normalisation.

2 Implicational Linear Logic

Linear logic is an example of a "resource-sensitive" logic, requiring that each assumption ('resource') is used precisely once in any deduction. For the implicational fragment, the set of formulae $\mathcal{F}$ are defined by $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \circ\!\!- \mathcal{F}$ (with $\mathcal{A}$ a nonempty set of atomic types). A *natural deduction* formulation requires the *elimination* and *introduction* rules in (1), which correspond semantically to steps of functional application and abstraction, respectively.

(1) $$\frac{\mathrm{A} \circ\!\!- \mathrm{B} : a \qquad \mathrm{B} : b}{\mathrm{A} : (ab)} \circ\!\!-\mathrm{E} \qquad \frac{\begin{array}{c}[\mathrm{B} : v] \\ \mathrm{A} : a\end{array}}{\mathrm{A} \circ\!\!- \mathrm{B} : \lambda v.a} \circ\!\!-\mathrm{I}$$

The proof (2) (which omits lambda terms) illustrates that 'hypothetical reasoning' in proofs (i.e. the use of additional assumptions that are later discharged or cancelled, such as Z here) is driven by the presence of higher-order formulae (such as X∘–(Y∘–Z) here).

(2)
$$\dfrac{\mathrm{X} \circ\!\!- (\mathrm{Y} \circ\!\!- \mathrm{Z}) \qquad \dfrac{\mathrm{Y} \circ\!\!- \mathrm{W} \qquad \dfrac{\mathrm{W} \circ\!\!- \mathrm{Z} \qquad [\mathrm{Z}]}{\mathrm{W}}}{\mathrm{Y}}}{\mathrm{X}}$$

Various type-logical categorial formalisms (or strictly their implicational fragments) differ from the above system only in imposing further restrictions on resource usage. For example, the associative Lambek calculus imposes a *linear order* over formulae, in which context, implication divides into two cases, (usually written \ and /) depending on whether the argument type appears to the left or right of the functor. Then, formulae may combine only if they are adjacent and in the appropriate left-right order. The non-associative Lambek calculus (Lambek, 1961) sets the further requirement that types combine under some fixed initial bracketting. Such weaker systems can be implemented by combining implicational linear logic with a labelling system whose labels are structured objects that record relevant resource information, i.e. of sequencing and/or bracketting, and then using this information in restricting permitted inferences to only those that satisfy the resource requirements of the weaker logic.

3 First-order Compilation

The first-order formulae are those with only atomic argument types (i.e. $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \circ\!\!- \mathcal{A}$). Hepple (1996) shows how deductions in implicational linear logic can be recast as deductions involving only *first-order* formulae.[3] The method involves compiling the original formulae to *indexed* first-order formulae, where a higher-order initial formula yields multiple compiled formulae, e.g. (omitting indices) X∘–(Y∘–Z) would yield X∘–Y *and* Z, i.e. with the subformula relevant to hypothetical reasoning (Z) effectively excised from the initial formulae, to be treated as a separate assumption, leaving a first-order residue. Indexing is used in ensuring general linear use of resources, but also notably to ensure *proper* use of excised subformulae, i.e. so that Z, in our example, must be used in deriving the argument of X∘–Y, and not elsewhere (otherwise invalid deductions would be derivable).

The approach is best explained by example. In proving X∘–(Y∘–Z), Y∘–W, W∘–Z ⇒ X, compilation of the premise formulae yields the indexed formulae that form the assumptions of (3), where formulae (i) and (iv) both derive from X∘–(Y∘–Z). (Note in (3) that the lambda terms of assumptions are written below their indexed types, simply to help the proof fit in the column.) Combination is allowed by the single inference rule (4).

(3)
$$\dfrac{\dfrac{\{i\} : \mathrm{X} \circ\!\!- (\mathrm{Y}{:}\{j\}) \quad \lambda t.x(\lambda z.t) \qquad \dfrac{\{k\} : \mathrm{Y} \circ\!\!- (\mathrm{W}{:}\emptyset) \quad \lambda u.yu \qquad \dfrac{\{l\} : \mathrm{W} \circ\!\!- (\mathrm{Z}{:}\emptyset) \quad \lambda v.wv \qquad \{j\} : \mathrm{Z} \quad z}{\{j,l\} : \mathrm{W} : wz}}{\{j,k,l\} : \mathrm{Y} : y(wz)}}{\{i,j,k,l\} : \mathrm{X} : x(\lambda z.y(wz))}$$
(i) (ii) (iii) (iv)

(4)
$$\frac{\phi : \mathrm{A} \circ\!\!- (\mathrm{B}{:}\alpha) : \lambda v.a \qquad \psi : \mathrm{B} : b}{\pi : \mathrm{A} : a[b /\!\!/ v]} \qquad \begin{array}{l}\pi = \phi \uplus \psi \\ \alpha \subset \psi\end{array}$$

Each assumption in (3) is associated with a set containing a single index, which serves as the unique

[3] The point of this manoeuvre (i.e. compiling to first-order formulae) is to create a deduction method which, like chart parsing for phrase-structure grammar, avoids the need to recompute intermediate results when searching exhaustively for all possible analyses, i.e. where any combination of types contributes to more than one overall analysis, it need only be computed once. The incremental system to be developed in this paper is similarly compatible with a 'chart-like' processing approach, although this issue will not be further addressed within this paper. For earlier work on chart-parsing type-logical formalisms, specifically the associative Lambek calculus, see König (1990), Hepple (1992), König (1994).

identifier for that assumption. The index sets of a derived formula identify precisely those assumptions from which it is derived. The rule (4) ensures appropriate indexation, i.e. via the condition $\pi = \phi \uplus \psi$, where $\uplus$ stands for *disjoint* union (ensuring linear usage). The common origin of assumptions (i) and (iv) (i.e. from X⟜(Y⟜Z)) is recorded by the fact that (i)'s argument is marked with (iv)'s index (j). The condition $\alpha \subset \psi$ of (4) ensures that (iv) must contribute to the derivation of (i)'s argument (which is needed to ensure correct inferencing). Finally, observe that the semantics of (4) is handled not by simple application, but rather by direct substitution for the variable of a lambda expression, employing a special variant of substitution, notated $\_[\_/\!/\_]$ (e.g. $t[s/\!/v]$ to indicate substitution of s for v in t), which specifically does not act to avoid accidental binding. In the final inference of (3), this method allows the variable z to fall within the scope of an abstraction over z, and so become bound. Recall that introduction inferences of the original formulation are associated with abstraction steps. In this approach, these inferences are no longer required, their effects having been compiled into the semantics. See (Hepple, 1996) for more details, including a precise statement of the compilation procedure.

4 Flexible Deduction

The approach just outlined is unsuited to incremental processing. Its single inference rule allows only a rigid style of combining formulae, where order of combination is completely determined by the argument order of functors. The formulae of (3), for example, must combine precisely as shown. It is not possible, say, to combine assumptions (i) and (ii) together first as part of a derivation. To overcome this limitation, we might generalise the combination rule to allow *composition* of functions, i.e. combinations akin to e.g. X⟜Y, Y⟜W ⇒ X⟜W. However, the treatment of indexation in the above system is one that does not readily adapt to flexible combination.

We will transform these indexed formulae to another form which better suits our needs, using the compilation procedure (5). This procedure returns a modified formula plus a set of equations that specify constraints on its indexation. For example, the assumptions (i-iv) of (3) yield the results (6) (ignoring semantic terms, which remain unchanged). Each atomic formula is partnered with an index set (or typically a variable over such), which corresponds to the full set of indices to be associated with the complete object of that category, e.g. in (i) we have (X+ϕ), plus the equation $\phi = \{i\} \uplus \pi$ which tells us that X's index set ϕ includes the argument formula Y's index set π plus its own index i. The further constraint equation $\phi = \{i\} \uplus \pi$ indicates that the argument's index set should include j (c.f. the conditions for using the original indexed formula).

(5)
$\sigma(\phi : X : t) = \langle (X{+}\phi) : t, \emptyset \rangle$
where X atomic
$\sigma(\phi : X \multimap Y : t) = \langle Z : t, C \rangle$
where $\sigma_1(\phi, X \multimap Y) = \langle Z, C \rangle$
$\sigma_1(\phi, X) = \langle (X{+}\gamma), \{\gamma = \phi\} \rangle$
where X atomic, γ a fresh variable
$\sigma_1(\phi, X_1 \multimap (Y : \pi)) = \langle X_2 \multimap (Y{+}\gamma), C' \rangle$
where δ, γ fresh variables, $\delta := \phi \uplus \gamma$
$\sigma_1(\delta, X_1) = \langle X_2, C \rangle$
$C' = C \cup \{\pi \subset \gamma\}$
(unless $\pi = \emptyset$, when $C = C'$)

(6)

i.	*old formula*:		$\{i\}$: X⟜(Y:$\{j\}$)
	↦ *new formula*:		(X+ϕ)⟜(Y+π)
	constraints:		$\{\phi = \{i\} \uplus \pi,\ \{j\} \subset \pi\}$
ii.	*old formula*:		$\{k\}$: Y⟜(W:$\emptyset$)
	↦ *new formula*:		(Y+α)⟜(W+β)
	constraints:		$\{\alpha = \{k\} \uplus \beta\}$
iii.	*old formula*:		$\{l\}$: W⟜(Z:$\emptyset$)
	↦ *new formula*:		(W+γ)⟜(Z+δ)
	constraints:		$\{\gamma = \{l\} \uplus \delta\}$
iv.	*old formula*:		$\{j\}$: Z
	↦ *new formula*:		(Z+$\{j\}$)
	constraints:		$\emptyset$

(7)
$$\frac{\mathrm{A} \multimap \mathrm{B} : \lambda v.a \qquad \mathrm{B} : b}{\mathrm{A} : a[b/\!/v]}$$

The previous inference rule (4) modifies to (7), which is simpler since indexation constraints are now handled by the separate constraint equations. We leave implicit the fact that use of the rule involves *unification* of the index variables associated with the two occurrences of "B" (in the standard manner). The constraint equations for the result of the combination are simply the sum of those for the formulae combined (as affected by the unification step). For example, combination of the formulae from (iii) and (iv) of (6) requires unification of the index set expressions δ and $\{j\}$, yielding the result formula (W+γ) plus the single constraint equation $\gamma = \{l\} \uplus \{j\}$, which is obviously satisfiable (with $\gamma = \{j, l\}$). A combination is not allowed if it results in an unsatisfiable set of constraints. The modified approach so neatly moves indexation requirements off into the constraint equation domain that we shall henceforth drop all consideration of them, assuming them to be appropriately managed in the background.

We can now state a generalised composition rule as in (8). The inference is marked as $[m,n]$, where m is the argument position of the 'functor' (always the lefthand premise) that is involved in the combination, and n indicates the number of arguments inherited from the 'argument' (righthand premise). The notation "$\circ\!\!-\!\!Z_n ... \circ\!\!-\!\!Z_1$" indicates a sequence of n arguments, where n may be zero, e.g. the case [1,0] corresponds precisely to the rule (7). Rule (8) allows the non-applicative derivation (9) over the formulae from (6) (c.f. the earlier derivation (3)).

(8)
$$\dfrac{\begin{array}{cc} X\circ\!\!-\!\!Y_m ... \circ\!\!-\!\!Y_1 & Y_m\circ\!\!-\!\!Z_n ... \circ\!\!-\!\!Z_1 \\ \lambda y_1 ... y_m.a & \lambda z_1 ... z_n.b \end{array}}{X\circ\!\!-\!\!Z_n ... \circ\!\!-\!\!Z_1\circ\!\!-\!\!Y_{m-1} ... \circ\!\!-\!\!Y_1 \\ \lambda y_1 ... y_{m-1} z_1 ... z_n.a[b/\!\!/ y_m]}\,[m,n]$$

(9)
$$\dfrac{\dfrac{\dfrac{\begin{array}{cccc} \text{(i)} & \text{(ii)} & \text{(iii)} & \text{(iv)} \\ X\circ\!\!-\!\!Y & Y\circ\!\!-\!\!W & W\circ\!\!-\!\!Z & Z \\ \lambda t.x(\lambda z.t) & \lambda u.yu & \lambda v.wv & z \end{array}}{X\circ\!\!-\!\!W : \lambda u.x(\lambda z.yu)}\,[1,1]}{X\circ\!\!-\!\!Z : \lambda v.x(\lambda z.y(wv))}\,[1,1]}{X : x(\lambda z.y(wz))}\,[1,0]$$

5 Incremental Derivation

As noted earlier, the relevance of flexible CGs to incremental processing relates to their ability to assign highly left-branching analyses to sentences, so that many initial substrings are treated as interpretable constituents. Although we have adapted the (Hepple, 1996) approach to allow flexibility in deduction, the applicability of the notion 'left-branching' is not clear since it describes the form of structures built in proof systems where formulae are placed in a linear order, with combination dependent on adjacency. Linear deduction methods, on the other hand, work with *unordered* collections of formulae. Of course, the system of labelling that is in use — where the constraints of the 'real' grammatical logic reside — may well import word order information that limits combination possibilities, but in designing a general parsing method for linear categorial formalisms, these constraints must remain with the labelling system.

This is not to say that there is no order information available to be considered in distinguishing incremental and non-incremental analyses. In an incremental processing context, the words of a sentence are delivered to the parser one-by-one, in 'left-to-right' order. Given lexical look-up, there will then be an 'order of delivery' of lexical formulae to the parser. Consequently, we can characterise an incremental analysis as being one that at any stage includes the maximal amount of 'contentful' combination of the formulae (and hence also lexical meanings) so far delivered, within the limits of possible combination that the proof system allows. Note that we have not in these comments reintroduced an ordered proof system of the familiar kind by the back door. In particular, we do not require formulae to combine under any notion of 'adjacency', but simply 'as soon as possible'.

For example, if the order of arrival of the formulae in (9) were (i,iv)$\prec$(ii)$\prec$(iii) (recall that (i,iv) originate from the same initial formula, and so must arrive together), then the proof (9) would be an incremental analysis. However, if the order instead was (ii)$\prec$(iii)$\prec$(i,iv), then (9) would *not* be incremental, since at the stage when only (ii) and (iii) had arrived, they could combine (as part of an equivalent alternative analysis), but are not so combined in (9).

6 Derivational Equivalence, Dependency & Normalisation

It seems we have achieved our aim of a linear deduction method that allows incremental analysis quite easily, i.e. simply by generalising the combination rule as in (8), having modified indexed formulae using (5). However, without further work, this 'achievement' is of little value, because the resulting system will be very computationally expensive due to the problem of 'derivational equivalence' or 'spurious ambiguity', i.e. the existence of multiple distinct proofs which assign the same reading. For example, in addition to the proof (9), we have also the equivalent proof (10).

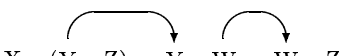
(10)
$$\dfrac{\begin{array}{cccc} \text{(i)} & \text{(ii)} & \text{(iii)} & \text{(iv)} \\ X\circ\!\!-\!\!Y & Y\circ\!\!-\!\!W & W\circ\!\!-\!\!Z & Z \\ \lambda t.x(\lambda z.t) & \lambda u.yu & \lambda v.wv & z \end{array}}{\dfrac{\dfrac{}{Y\circ\!\!-\!\!Z : \lambda v.y(wv)}\,[1,1]}{Y : y(wz)}\,[1,0]}{X : x(\lambda z.y(wz))}\,[1,0]$$

The solution to this problem involves specifying a *normal form* for deductions, and allowing that only normal form proofs are constructed.[4] Our route to specifying a normal form for proofs exploits a correspondence between proofs and *dependency structures*. Dependency grammar (DG) takes as fundamental

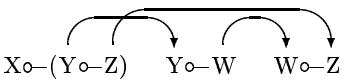
[4]This approach of 'normal form parsing' has been applied to the associative Lambek calculus in (König, 1989), (Hepple, 1990), (Hendriks, 1992), and to Combinatory Categorial Grammar in (Hepple & Morrill, 1989), (Eisner, 1996).

the notions of *head* and *dependent*. An analogy is often drawn between CG and DG based on equating categorial functors with heads, whereby the arguments sought by a functor are seen as its dependents. The two approaches have some obvious differences. Firstly, the argument requirements of a categorial functor are ordered. Secondly, arguments in CG are phrasal, whereas in DG dependencies are between words. However, to identify the dependency relations entailed by a proof, we may simply ignore argument ordering, and we can trace through the proof to identify those initial assumptions ('words') that are related as head and dependent by each combination of the proof. This simple idea unfortunately runs into complications, due to the presence of higher order functions. For example, in the proof (2), since the higher order functor's argument category (i.e. Y$\circ$–Z) has subformulae corresponding to components of *both* of the other two assumptions, Y$\circ$–W and W$\circ$–Z, it is not clear whether we should view the higher order functor as having a dependency relation only to the 'functionally dominant' assumption Y$\circ$–W, i.e. with dependencies as in (11a), or to *both* the assumptions Y$\circ$–W *and* W$\circ$–Z, i.e. with dependencies as perhaps in either (11b) or (11c). The compilation approach, however, lacks this problem, since we have only first order formulae, amongst which the dependencies are clear, e.g. as in (12).

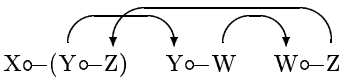
(11) (a)



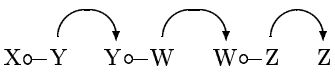
(b)



(c)



(12)



Some preliminaries. We assume that proof assumptions explicitly record 'order of delivery' information, marked by a natural number, and so take the form: $\frac{n}{X}[a]$

Further, we require the ordering to go beyond simple 'order of delivery' in relatively ordering first order assumptions that derive from the same original higher-order formula. (This move simply introduces some extra arbitrary bias as a basis for distinguishing proofs.) It is convenient to have a 'linear' notation for writing proofs. We will write $(n/X\ [a])$ for an assumption (such as that just shown), and $(X\ Y\ /\ Z\ [m,n])$ for a combination of subproofs X and Y to give result formula Z by inference $[m,n]$.

(13) $\mathsf{dep}((X\ Y\ /\ Z\ [m,n])) = \{\langle i,j,k\rangle\}$
where $\mathsf{gov}(m,X) = \langle i,k\rangle$, $\mathsf{fun}(Y) = j$

(14) $\mathsf{dep}*((n/X\ [a])) = \emptyset$

$\mathsf{dep}*((X\ Y\ /\ Z\ [m,n]))$
$= \{\delta\} \cup \mathsf{dep}*(X) \cup \mathsf{dep}*(Y)$
where $\delta = \mathsf{dep}((X\ Y\ /\ Z\ [m,n]))$

The procedure dep, defined in (13), identifies the dependency relation established by any combination, i.e. for any subproof $P = (X\ Y\ /\ Z\ [m,n])$, $\mathsf{dep}(P)$ returns a triple $\langle i,j,k\rangle$, where i,j identify the head and dependent assumptions for the combination, and k indicates the argument position of the head assumption that is involved (which has now been inherited to be argument m of the functor of the combination). The procedure dep$*$, defined in (14), returns the *set* of dependencies established within a subproof. Note that dep employs the procedures gov (which traces the relevant argument back to its source assumption — the head) and fun (which finds the functionally dominant assumption within the argument subproof — the dependent).

(15) $\mathsf{gov}(i,(n/X\ [a])) = \langle n,i\rangle$

$\mathsf{gov}(i,(X\ Y\ /\ Z\ [m,n])) = \mathsf{gov}((i-m+1),Y)$
where $m \leq i < (m+n)$

$\mathsf{gov}(i,(X\ Y\ /\ Z\ [m,n])) = \mathsf{gov}(i,X)$
where $i < m$

$\mathsf{gov}(i,(X\ Y\ /\ Z\ [m,n])) = \mathsf{gov}((i-n+1),X)$
where $(m+n) \leq i$

(16) $\mathsf{fun}((n/X\ [a])) = n$

$\mathsf{fun}((X\ Y\ /\ Z\ [m,n])) = \mathsf{fun}(X)$

From earlier discussion, it should be clear that an 'incremental analysis' is one in which any dependency to be established is established as soon as possible in terms of the order of delivery of assumptions. The relation $<<$ of (17) orders dependencies in terms of which can be established earlier on, i.e. $\delta << \gamma$ if the later-arriving assumption of δ arrives before the later-arriving assumption of γ. Note however that δ, γ may have the *same* later arriving assumption (i.e. if this assumption is involved in more than one dependency). In this case, $<<$ arbitrarily gives precedence to the dependency whose two assumptions occur closer together in delivery order.

(17) $\delta << \gamma$ (*where* $\delta = \langle i,j,k\rangle, \gamma = \langle x,y,z\rangle$)

iff $(\mathtt{max}(i,j) < \mathtt{max}(x,y) \vee$
$(\mathtt{max}(i,j) = \mathtt{max}(x,y) \wedge$
$\mathtt{min}(i,j) > \mathtt{min}(x,y)))$

We can use $<<$ to define an *incremental normal form* for proofs, i.e. an incremental proof is one that is *well-ordered* with respect to $<<$ in the sense that every combination $(X\ Y\ /\ Z\ [m,n])$ within it establishes a dependency δ which follows under $<<$ every dependency δ' established within the subproofs X and Y it combines, i.e. $\delta' << \delta$ for each $\delta' \in \mathsf{dep}*(X) \cup \mathsf{dep}*(Y)$. This normal form is useful only if we can show that every proof has an equivalent normal form. For present purposes, we can take two proofs to be equivalent *iff* they establish identical sets of dependency relations.[5]

(18) $\mathsf{trace}(i,j,(i/X\ [a])) = j$

$\mathsf{trace}(i,j,(X\ Y\ /\ Z\ [m,n])) = (m+k-1)$
where $i \in \mathsf{assum}(Y)$
$\mathsf{trace}(i,j,Y) = k$

$\mathsf{trace}(i,j,(X\ Y\ /\ Z\ [m,n])) = k$
where $i \in \mathsf{assum}(X)$
$\mathsf{trace}(i,j,X) = k,\ \ k < m$

$\mathsf{trace}(i,j,(X\ Y\ /\ Z\ [m,n])) = (k+n-1)$
where $i \in \mathsf{assum}(X)$
$\mathsf{trace}(i,j,X) = k,\ \ k > m$

(19) $\mathsf{assum}((i/X\ [a])) = \{i\}$

$\mathsf{assum}((X\ Y\ /\ Z\ [m,n]))$
$= \mathsf{assum}(X) \cup \mathsf{assum}(Y)$

We can specify a method such that given a set of dependency relations $\mathcal{D}$ we can construct a corresponding proof. The process works with a set of subproofs $\mathcal{P}$, which are initially just the set of assumptions (i.e. each of the form $(n/F\ [a])$), and proceeds by combining pairs of subproofs together, until finally just a single proof remains. Each step involves selecting a dependency δ ($\delta = \langle i,j,k\rangle$) from $\mathcal{D}$ (setting $\mathcal{D} := \mathcal{D} - \{\delta\}$ for subsequent purposes), removing the subproofs P, Q from $\mathcal{P}$ which contain the assumptions i, j (respectively), combining P, Q (with P as functor) to give a new subproof R which is added to $\mathcal{P}$ (i.e. $\mathcal{P} := (\mathcal{P} - \{P,Q\}) \cup \{R\}$). It is important to get the right value for m in the combination $[m,n]$ used to combine P, Q, so that the correct argument of the assumption i (as now inherited to the end-type of P) is involved. This value is given by $m = \mathsf{trace}(i,k,P)$ (with trace as defined in (18)). The process of proof construction is nondeterministic, in the order of selection of dependencies for incorporation, and so a single set of dependences can yield multiple distinct, but equivalent, proofs (as we would expect).

To build normal form proofs, we only need to limit the order of selection of dependencies using $<<$, i.e. requiring that the *minimal* element under $<<$ is selected at each stage. Note that this ordering restriction makes the selection process deterministic, from which it follows that normal forms are *unique*. Putting the above methods together, we have a complete normal form method for proofs of the first-order linear deduction system, i.e. for any proof P, we can extract its dependency relations and use these to construct a unique, maximally incremental, alternative proof — the normal form of P.

7 Proof Reduction and Normalisation

The above normalisation approach is somewhat nonstandard. We shall next briefly sketch how normalisation could instead be handled via the standard method of *proof reduction*. This method involves defining a *contraction* relation ($\rhd_1$) between proofs, which is typically stated as a number of contraction rules of the form X $\rhd_1$ Y, where X is termed a *redex* and Y its *contractum*. Each rule allows that a proof containing a redex be transformed into one where that occurrence is replaced by its contractum. A proof is in normal form *iff* it contains no redexes. The contraction relation generates a *reduction* relation ($\rhd$) such that X *reduces to* Y (X $\rhd$ Y) *iff* Y is obtained from X by a finite series (possibly zero) of contractions. A term Y is a normal form of X *iff* Y is a normal form and X $\rhd$ Y.

We again require the ordering relation $<<$ defined in (17). A redex is any subproof whose final step is a combination of two well-ordered subproofs, which establishes a dependency that undermines well-orderedness. A contraction step modifies the proof to swap this final combination with the final one of an immediate subproof, so that the dependencies the two combinations establish are now appropriately ordered with respect to each other. The possibilities for reordering combination steps divide into four cases, which are shown in Figure 1. This re-

[5] This criterion turns out to be equivalent to one stated in terms of the lambda terms that proofs generate, i.e. two proofs will yield identical sets of dependency relations *iff* they yield proof terms that are $\beta\eta$-equivalent. This observation should not be surprising, since the set of 'dependency relations' returned for a proof is in essence just a rather unstructured summary of its functional relations.

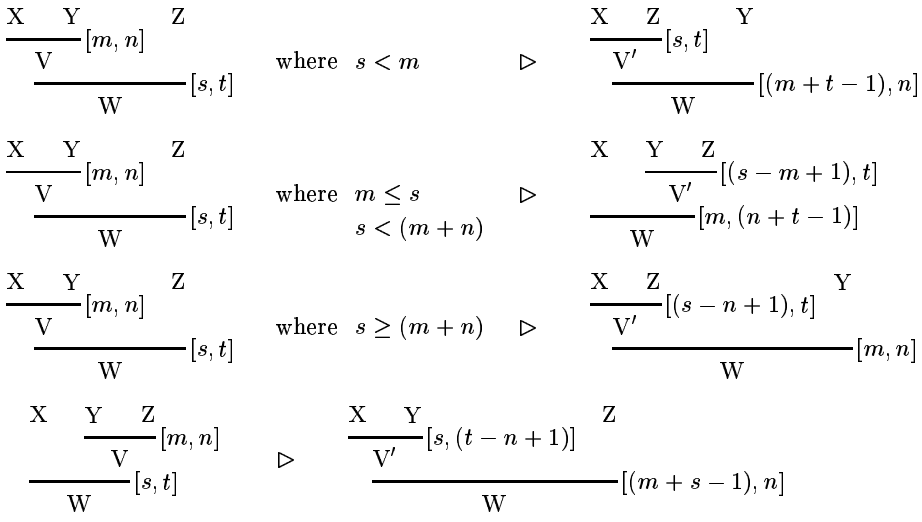


Figure 1: Local Reordering of Combination Steps: the four cases

duction system can be shown to exhibit the property (called *strong normalisation*) that every reduction is finite, from which it follows that every proof has a normal form.[6]

8 Normal form parsing

The technique of normal form parsing involves ensuring that only normal form proofs are constructed by the parser, avoiding the unnecessary work of building all the non-normal form proofs. At any stage, all subproofs so far constructed are in normal form, and the result of any combination is admitted only provided it is in normal form, otherwise it is discarded. The result of a combination is recognised as non-normal form if it establishes a dependency that is out of order with respect to that of the final combination of at least one of the two subproofs combined (which is an adequate criterion since the subproofs are well-ordered). The procedures defined above can be used to identify these dependencies.

9 The Degree of Incrementality

Let us next consider the degree of incrementality that the above system allows, and the sense in which it might be considered maximal. Clearly, the system does not allow full 'word-by-word' incrementality, i.e. where the words that have been delivered at any stage in incremental processing are combined to give a single result formula, with combinations to incorporate each new lexical formula as it arrives.[7] For example, in incremental processing of *Today John sang*, the first two words might yield (after compilation) the first-order formulae so–s and np, which will not combine under the rule (8).[8]

Instead, the above system will allow precisely those combinations that establish functional relations that are marked out in lexical type structure (i.e. subcategorisation), which, given the parallelism of syntax and semantics, corresponds to allowing those combinations that establish semantically relevant functional relations amongst lexical meanings. Thus, we believe the above system to exhibit maximal incrementality in relation to allowing 'semantically contentful' combinations. In dependency terms, the system allows any set of initial formulae to combine to a single result *iff* they form a connected graph under the dependency relations that obtain amongst them.

Note that the extent of incrementality allowed by using 'generalised composition' in the compiled first-order system should not be equated with that which

[6]To prove strong normalisation, it is sufficient to give a metric which assigns to each proof a finite non-negative integer score, and under which every contraction reduces a proof's score by a non-zero amount. The following metric μ can be shown to suffice: (a) for $P = (n/X\ [a])$, $\mu(P) = 0$, (b) for $P = (X\ Y\ /\ Z\ [m,n])$, whose final step establishes a dependency δ, $\mu(P) = \mu(X) + \mu(Y) + D$, where D is the number of dependencies δ' such that $\delta << \delta'$, which are established in X and Y, i.e. $D = |\Delta|$ where $\Delta = \{\delta' \mid \delta' \in \mathsf{dep}*(X) \cup \mathsf{dep}*(Y) \ \wedge\ \delta << \delta'\}$.

[7]For an example of a system allowing word-by-word incrementality, see (Milward, 1995).

[8]Note that this is *not* to say that the system is unable to combine these two types, e.g. a combination so–s, np ⇒ so–(so–np) *is* derivable, with appropriate compilation. The point rather is that such a combination will typically not happen as a *component* in a proof of some other overall deduction.

would be allowed by such a rule in the original (non-compiled) system. We can illustrate this point using the following type combination, which is not an instance of even 'generalised' composition.

$$X\circ\!-(Y\circ\!-Z),\ \ Y\circ\!-W \Rightarrow X\circ\!-(W\circ\!-Z)$$

Compilation of the higher-order assumption would yield X∘–Y plus Z, of which the first formula can compose with the second assumption Y∘–W to give X∘–W, thereby achieving some semantically contentful combination of their associated meanings, which would not be allowed by composition over the original formulae.[9]

10 Conclusion

We have shown how the linear categorial deduction method of (Hepple, 1996) can be modified to allow incremental derivation, and specified an incremental normal form for proofs of the system. These results provide for an efficient incremental linear deduction method that can be used with various labelling disciplines as a basis for parsing a range of type-logical formalisms.

References

Jason Eisner 1996. 'Efficient Normal-Form Parsing for Combinatory Categorial Grammar.' *Proc. of ACL-34.*

Dov M. Gabbay. 1996. *Labelled deductive systems. Volume 1.* Oxford University Press.

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

Mark Hepple. 1990. 'Normal form theorem proving for the Lambek calculus'. *Proc. of COLING–90.*

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

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

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

[9]This combination corresponds to what in a directional system Wittenburg (1987) termed a 'predictive combinator', e.g. such as X/(Y/Z), Y/W ⇒ X/(W/Z). Indeed, the semantic result for the combination in the first-order system corresponds closely to that which would be produced under Wittenburg's rule.

Mark Hepple & Glyn Morrill. 1989. 'Parsing and derivational equivalence.' *Proc. of EACL–4.*

Esther König. 1989. 'Parsing as natural deduction'. *Proc. of ACL-27.*

Esther König. 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, pp1–19.

Joachim Lambek. 1958. 'The mathematics of sentence structure.' *American Mathematical Monthly*, **65**, pp154–170.

Joachim Lambek. 1961. 'On the calculus of syntactic types.' R. Jakobson (Ed), *Structure of Language and its Mathematical Aspects*, Proceedings of the Symposia in Applied Mathematics XII, American Mathematical Society.

David Milward. 1995. 'Incremental Interpretation of Categorial Grammar.' *Proceedings EACL–7*, Dublin.

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

Michael Moortgat & Richard T. Oehrle. 1994. 'Adjacency, dependency and order'. *Proc. of Ninth Amsterdam Colloquium.*

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

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

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

Mark J. Steedman. 1989. 'Grammar, interpretation and processing from the lexicon.' In Marslen-Wilson, W. (Ed), *Lexical Representation and Process*, MIT Press, Cambridge, MA.

Kent Wittenburg. 1987. 'Predictive Combinators: A method for efficient parsing of Combinatory Categorial Grammars.' *Proc. of ACL-25.*