Proceedings of the COLING-ACL'98 Joint Conference (The 17th International Conference on Computational Linguistics, and 36th Annual Meeting of the Association for Computational Linguistics). Montreal, August 1998

Memoisation for Glue Language Deduction and Categorial Parsing

Mark Hepple

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

Abstract

The multiplicative fragment of linear logic has found a number of applications in computational linguistics: in the "glue language" approach to LFG semantics, and in the formulation and parsing of various categorial grammars. These applications call for efficient deduction methods. Although a number of deduction methods for multiplicative linear logic are known, none of them are tabular methods, which bring a substantial efficiency gain by avoiding redundant computation (c.f. chart methods in CFG parsing): this paper presents such a method, and discusses its use in relation to the above applications.

1 Introduction

The multiplicative fragment of linear logic, which includes just the linear implication ($\multimap$) and multiplicative ($\otimes$) operators, has found a number of applications within linguistics and computational linguistics. Firstly, it can be used in combination with some system of labelling (after the 'labelled deduction' methodology of (Gabbay, 1996)) as a general method for formulating various categorial grammar systems. Linear deduction methods provide a common basis for parsing categorial systems formulated in this way. Secondly, the multiplicative fragment forms the core of the system used in work by Dalrymple and colleagues for handling the semantics of LFG derivations, providing a 'glue language' for assembling the meanings of sentences from those of words and phrases.

Although there are a number of deduction methods for multiplicative linear logic, there is a notable absence of tabular methods, which, like chart parsing for CFGs, avoid redundant computation. Hepple (1996) presents a compilation method which allows for tabular deduction for *implicational* linear logic (i.e. the fragment with only $\multimap$). This paper develops that method to cover the fragment that includes the multiplicative. The use of this method for the applications mentioned above is discussed.

2 Multiplicative Linear Logic

Linear logic is a 'resource-sensitive' logic: in any deduction, each assumption ('resource') is used precisely once. The formulae of the multiplicative fragment of (intuitionistic) linear logic are defined by $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \multimap \mathcal{F} \mid \mathcal{F} \otimes \mathcal{F}$ ($\mathcal{A}$ a nonempty set of atomic types). The following rules provide a *natural deduction* formulation:

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

$$\frac{\begin{array}{c}[\mathrm{B} : x],[\mathrm{C} : y] \qquad \mathrm{B}\otimes\mathrm{C} : b \\ \mathrm{A} : a\end{array}}{\mathrm{A} : \mathrm{E}^{\otimes}_{x,y}(b, a)}\otimes\mathrm{E} \qquad \frac{\mathrm{A} : a \qquad \mathrm{B} : b}{\mathrm{A}\otimes\mathrm{B} : (a \otimes b)}\otimes\mathrm{I}$$

The elimination (E) and introduction (I) rules for $\multimap$ correspond to steps of functional application and abstraction, respectively, as the term labelling reveals. The $\multimap$I rule discharges precisely one assumption (B) within the proof to which it applies. The $\otimes$I rule pairs together the premise terms, whereas $\otimes$E has a substitution like meaning.[1] Proofs that W$\multimap$(X$\multimap$Z), X$\multimap$Y, Y$\multimap$Z $\Rightarrow$ W and that X$\multimap$Y$\multimap$Z, Y$\otimes$Z $\Rightarrow$ X follow:

$$\cfrac{\mathrm{W}\multimap(\mathrm{X}\multimap\mathrm{Z}) : w \qquad \cfrac{\cfrac{\mathrm{X}\multimap\mathrm{Y} : x \qquad \cfrac{\mathrm{Y}\multimap\mathrm{Z} : y \qquad [\mathrm{Z} : z]}{\mathrm{Y} : (yz)}}{\mathrm{X} : (x(yz))}}{\mathrm{X}\multimap\mathrm{Z} : \lambda z.x(yz)}}{\mathrm{W} : w(\lambda z.x(yz))}$$

[1] The meaning is more obvious in the notation of (Benton *et al.*, 1992): **(let** b **be** $x \otimes y$ **in** a**)**.

$$\dfrac{\dfrac{\dfrac{\mathrm{X}\circ\!\!-\!\mathrm{Y}\circ\!\!-\!\mathrm{Z}:x \quad [\mathrm{Z}:z]}{\mathrm{X}\circ\!\!-\!\mathrm{Y}:(xz)} \quad [\mathrm{Y}:y]}{\mathrm{X}:(xzy)} \quad \mathrm{Y}\otimes\mathrm{Z}:w}{\mathrm{X}:\mathrm{E}^{\otimes}_{y,z}(w,(xzy))}$$

The differential status of the assumptions and goal of a deduction (i.e. between Γ and A in $\Gamma \Rightarrow A$) is addressed in terms of *polarity*: assumptions are deemed to have positive polarity, and goals negative polarity. Each Subformula also has a polarity, which is determined by the polarity of the immediately containing (sub)formula, according to the following schemata (where $\bar{p}$ is the opposite polarity to p):

(i) $(\ \mathrm{X}^{p} \circ\!\!-\ \mathrm{Y}^{\bar{p}}\ )^{p}$ (ii) $(\ \mathrm{X}^{p} \otimes \mathrm{Y}^{p}\ )^{p}$

For example, the leftmost assumption of the first proof above has the polarity pattern $(\ \mathrm{W}^{+} \circ\!\!-\ (\mathrm{X}^{-} \circ\!\!-\ \mathrm{Z}^{+}\ )^{-}\ )^{+}$. The proofs illustrate the phenomenon of 'hypothetical reasoning', where additional assumptions (called 'hypotheticals') are used, which are later discharged. The need for hypothetical reasoning in a proof is driven by the types of the assumptions and goal: the hypotheticals correspond to positive polarity subformulae of the assumptions/goal that occur in the following subformula contexts:

i) $(\mathrm{X}^{-} \circ\!\!-\ \mathrm{Y}^{+})^{-}$ (giving hypothetical Y)
ii) $(\mathrm{X}^{+} \otimes \mathrm{Y}^{+})^{+}$ (giving hypo's X and Y)

The subformula (X$\circ\!\!-$Z) of W$\circ\!\!-$(X$\circ\!\!-$Z) in the proof above is an instance of context (i), so a hypothetical Z results. Subformulae that are instances of patterns (i,ii) may nest within other such instances (e.g. in ((A$\otimes$B)$\otimes$C)$\circ\!\!-$D, both ((A$\otimes$B)$\otimes$C) and (A$\otimes$B) are instances of (ii)). In such cases, we can focus on the *maximal* pattern instances (i.e. not contained within any other), and then examine the hypotheticals produced for whether they in turn license hypothetical reasoning. This approach makes explicit the patterns of dependency amongst hypothetical elements.

3 First-order Compilation for Implicational Linear Logic

Hepple (1996) shows how deductions in *implicational* linear logic can be recast as deductions involving only *first-order* formulae, using only a single inference rule (a variant of $\circ\!\!-$E). The method involves compiling the original formulae to *indexed* first-order formulae, where a higher-order[2] initial formula yields multiple compiled formulae, e.g. (omitting indices) X$\circ\!\!-$(Y$\circ\!\!-$Z) would yield X$\circ\!\!-$Y *and* Z, i.e. with the subformula Z, relevant to hypothetical reasoning, being excised to be treated as a separate assumption, leaving a first-order residue.[3] Indexing is used to ensure 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$\circ\!\!-$Y, or otherwise invalid deductions would result). Simplifying X$\circ\!\!-$(Y$\circ\!\!-$Z) to X$\circ\!\!-$Y removes the need for an $\circ\!\!-$I inference, but the effect of such a step is not lost, since it is compiled into the semantics of the formula.

The approach is best explained by example. In proving X$\circ\!\!-$(Y$\circ\!\!-$Z), Y$\circ\!\!-$W, W$\circ\!\!-$Z $\Rightarrow$ X, the premise formulae compile to the indexed formulae (1–4) shown in the proof below. Each of these formulae (1–4) is associated with a set containing a single index, which serves as a unique identifier for that assumption.

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

The formulae (5–7) arise under combination, allowed by the single rule below. The index sets of these formulae identify precisely the assumptions from which they are derived, with appropriate indexation being ensured by the condition $\pi = \phi \uplus \psi$ of the rule (where $\uplus$ stands for *disjoint* union, which enforces linear usage).

$$\dfrac{\langle \phi, \mathrm{A}\circ\!\!-(\mathrm{B}:\alpha), \lambda v.a\rangle \quad \langle \psi, \mathrm{B}, b\rangle}{\langle \pi, \mathrm{A}, a[b/\!\!/v]\rangle} \quad \begin{array}{l}\pi = \phi \uplus \psi \\ \alpha \subseteq \psi\end{array}$$

[2]The key division here is between higher-order formulae, which are are functors that seek at least one argument that bears a a functional type (e.g. W$\circ\!\!-$(X$\circ\!\!-$Z)), and *first-order* formulae, which seek no such argument.

[3]This 'excision' step has parallels to the 'emit' step used in the chart-parsing approaches for the associative Lambek calculus of (König, 1994) and (Hepple, 1992), although 'emit' differs in that there is no removal of the relevant subformula, i.e. the 'emitting formula' is not simplified, remaining higher-order.

Assumptions (1) and (4) both come from X⊸(Y⊸Z): note how (1)'s argument is marked with (4)'s index (j). The condition $\alpha \subseteq \psi$ of the rule ensures that (4) must contribute to the derivation of (1)'s argument. Finally, observe that the rule's semantics involves not simple application, but rather direct substitution for the variable of a lambda expression, employing a special variant of substitution, notated $\_[\_/\!\!/\_]$, which specifically does not act to avoid accidental binding. Hence, in the final inference of the proof, the variable z falls within the scope of an abstraction over z, becoming bound. The abstraction over z corresponds to an ⊸I step that is compiled into the semantics, so that an explicit inference is no longer required. It is the lack of the ⊸I rule that allows for non-redundant proof search, i.e. the single rule of the compiled system can be used to perform a systematic enumeration of type combinations in a manner such that any intermediate result need only be computed once, however many larger combinations it may contribute to (see Sec. 6 for algorithms).

4 First-order Compilation for Multiplicative Linear Logic

In extending the above approach to the multiplicative, we will address the ⊗I and ⊗E rules as separate problems. The need for an ⊗I use within a proof is driven by the type of either some assumption or the proof's overall goal, e.g. to build the argument of an assumption such as A⊸(B⊗C). For this specific example, we might try to avoid the need for an explicit ⊗I use by transforming the assumption to the form A⊸B⊸C (note that the two formulae are interderivable). This line of exploration, however, leads to incompleteness, since the manoeuvre results in proof structures that lack a node corresponding to the result of the ⊗I inference (which is present in the natural deduction proof), and this node may be needed as the locus of some other inference.[4] This problem can be overcome by the use of *goal atoms*, which are unique pseudo-type atoms, that are introduced into types by compilation (in the parlance of lisp, they are 'gensymmed' atoms). An assumption A⊸(B⊗C) would compile to A⊸$\mathcal{G}$ plus $\mathcal{G}$⊸B⊸C, where $\mathcal{G}$ is the unique goal atom (g1, perhaps). A proof using these types does contain a node corresponding to (what would be) the result of the ⊗ inference in the natural deduction proof, namely that bearing type $\mathcal{G}$, the result of combining $\mathcal{G}$⊸B⊸C with its arguments.

This method can be used in combination with the existing compilation approach. For example, an initial assumption A⊸((B⊗C)⊸D) would yield a hypothetical D, leaving the residue A⊸(B⊗C), which would become A⊸$\mathcal{G}$ plus $\mathcal{G}$⊸B⊸C, as just discussed. This method of uniquely-generated 'goal atoms' can also be used in dealing with deductions having complex types for their intended overall result (which may license hypotheticals, by virtue of realising the polarity contexts discussed in section 2). Thus, we can replace an initial deduction $\Gamma \Rightarrow$ A with $\mathcal{G}$⊸A, $\Gamma \Rightarrow \mathcal{G}$, making the goal A part of the left hand side. The new premise $\mathcal{G}$⊸A can be compiled just like any other. Since the new goal formula $\mathcal{G}$ is atomic, it requires no compilation. For example, a goal type X⊸Y would become an extra premise $\mathcal{G}$⊸(X⊸Y), which would compile to formulae $\mathcal{G}$⊸X plus Y.

Turning next to ⊗E, the rule involves hypothetical reasoning, so compilation of a maximal positive polarity subformula B⊗C will add hypotheticals B,C. No further compilation of B⊗C itself is then required: whatever is needed for hypothetical reasoning with respect to the internal structure of its subformulae will arise elsewhere by compilation of the hypotheticals B,C. Assume that these latter hypotheticals have identifying indices i, j and semantic variables x, y respectively. A rule for ⊗E might combine B⊗C (with term t, say) with any other formula A (with term s, say) provided that the latter has a disjoint index set that includes i, j, to give a result that is also of type A, that is assigned semantics $\mathrm{E}^{\otimes}_{x,y}(t, s)$. To be able to construct this semantics, the rule would need to be able to access the identities of the variables x, y. The need to explicitly annotate this identity information might be avoided by 'raising' the semantics of the multiplicative formula at compilation time to be a function over the other term, e.g. t might be raised to $\lambda u.\mathrm{E}^{\otimes}_{x,y}(t, u)$. A usable inference rule might then take the follow-

[4] Specifically, the node must be present to allow for steps corresponding to ⊗E inferences. The expert reader should be able to convince themselves of this fact by considering an example such as X⊸((Y⊗U)⊸(Z⊗U)), Y⊸Z ⇒ X.

ing form (where the identifying indices of the hypotheticals have been marked on the product type):

$$\frac{\langle \phi, \mathrm{A}, s\rangle \quad \langle \psi, (\mathrm{B} \otimes \mathrm{C}) : \{i,j\}, \lambda u.t\rangle \quad \begin{array}{l} i,j \in \phi \\ \pi = \phi \uplus \psi \end{array}}{\langle \pi, \mathrm{A}, t[s/\!/u]\rangle}$$

Note that we can safely restrict the rule to require that the type A of the minor premise is *atomic*. This is possible since firstly, the first-order compilation context ensures that the arguments required by a functor to yield an atomic result are always present (with respect to completing a valid deduction), and secondly, the alternatives of combining a functor with a multiplicative under the rule either before or after supplying its arguments are equivalent.[5]

In fact, we do not need the rule above, as we can instead achieve the same effects using only the single ($\circ\!\!-$) inference rule that we already have, by allowing a very restricted use of type polymorphism. Thus, since the above rule's conclusion and minor premise are the same atomic type, we can in the compilation simply replace a formula X$\otimes$Y, with an implication $\mathcal{A}\circ\!\!-(\mathcal{A}{:}\{i,j\})$, where $\mathcal{A}$ is a variable over *atomic types* (and i,j the identifying indices of the two hypotheticals generated by compilation). The semantics provided for this functor is of the 'raised' kind discussed above. However, this approach to handling $\otimes$E inferences within the compiled system has an undesirable characteristic (which would also arise using the dedicated inference rule discussed above), which is that it will allow multiple derivations that assign equivalent proof terms for a given type combination. This is due to non-determinism for the stage at which a type such as $\mathcal{A}\circ\!\!-(\mathcal{A}{:}\{i,j\})$ participates in the proof. A proof might contain several nodes bearing atomic types which contain the required hypotheticals, and $\mathcal{A}\circ\!\!-(\mathcal{A}{:}\{i,j\})$ might combine in at any of these nodes, giving equivalent results.[6]

The above ideas for handling the multiplicative are combined with the methods developed for the implicational fragment to give the compilation procedure (τ), stated in Figure 1. This takes a sequent $\Gamma \Rightarrow$ A as input (case $\tau 1$), where A is a type and each assumption in Γ takes the form *Type:Sem* (*Sem* minimally just some unique variable), and it returns a structure $\langle \mathcal{G}, \phi, \Delta\rangle$, where $\mathcal{G}$ is a goal atom, ϕ the set of all identifying indices, and Δ a set of indexed first order formulae (with associated semantics). Let Δ^* denote the result of closing Δ under the single inference rule. The sequent is proven *iff* $\langle \phi, \mathcal{G}, t\rangle \in \Delta^*$ for some term t, which is a complete proof term for the implicit deduction. The statement of the compilation procedure here is somewhat different to that given in (Hepple, 1996), which is based on polar translation functions. In the version here, the formula related cases address only positive formulae.[7]

As an example, consider the deduction X$\circ\!\!-$Y, Y$\otimes$Z $\Rightarrow$ X$\otimes$Z. Compilation returns the goal atom g0, the full index set $\{g,h,i,j,k,l\}$, plus the formulae show in (1–6) below.

1.	$\langle \{g\}, \mathrm{g0} \circ\!\!-(\mathrm{g1}{:}\{h\}), \lambda t.t\rangle$	
2.	$\langle \{h\}, \mathrm{g1} \circ\!\!-(\mathrm{X}{:}\emptyset)\circ\!\!-(\mathrm{Z}{:}\emptyset), \lambda v \lambda w.(w \otimes v)\rangle$	
3.	$\langle \{i\}, \mathrm{X} \circ\!\!-(\mathrm{Y}{:}\emptyset), \lambda x.(ax)\rangle$	
4.	$\langle \{j\}, \mathcal{A}\circ\!\!-(\mathcal{A}{:}\{k,l\}), \lambda u.\mathrm{E}^{\otimes}_{y,z}(b,u)\rangle$	
5.	$\langle \{k\}, \mathrm{Y}, y\rangle$	
6.	$\langle \{l\}, \mathrm{Z}, z\rangle$	
7.	$\langle \{i,k\}, \mathrm{X}, (ay)\rangle$	[3+5]
8.	$\langle \{h,l\}, \mathrm{g1} \circ\!\!-(\mathrm{X}{:}\emptyset), \lambda w.(w \otimes z)\rangle$	[2+6]
9.	$\langle \{h,i,k,l\}, \mathrm{g1}, ((ay) \otimes z)\rangle$	[7+8]
10.	$\langle \{h,i,j,k,l\}, \mathrm{g1}, \mathrm{E}^{\otimes}_{y,z}(b, ((ay) \otimes z))\rangle$	[4+9]
11.	$\langle \{g,h,i,j,k,l\}, \mathrm{g0}, \mathrm{E}^{\otimes}_{y,z}(b, ((ay) \otimes z))\rangle$	[1+11]
12.	$\langle \{g,h,i,k,l\}, \mathrm{g0}, ((ay) \otimes z)\rangle$	[1+9]
13.	$\langle \{g,h,i,j,k,l\}, \mathrm{g0}, \mathrm{E}^{\otimes}_{y,z}(b, ((ay) \otimes z))\rangle$	[4+12]

[5] This follows from the proof term equivalence $\mathrm{E}^{\otimes}_{x,y}(f,(ga)) = (\mathrm{E}^{\otimes}_{x,y}(f,g)\ a)$ where $x,y \in \mathsf{freevars}(g)$. The move of requiring the minor premise to be atomic effects a partial normalisation which involves not only the relative ordering of $\otimes$E and $\circ\!\!-$E steps, but also that between interdependent $\otimes$E steps (as might arise for an assumption such as ((A$\otimes$B)$\otimes$C)). It is straightforward to demonstrate that the restriction results in no loss of readings. See (Benton *et al.*, 1992) regarding term assignment and proof normalisation for linear logic.

[6] It is anticipated that this problem can be solved by using normalisation results as a basis for discarding partial analyses during processing, but further work is required in developing this idea.

[7] Note that the complexity of the compilation is linear in the 'size' of the initial deduction, as measured by a count of type atoms. For applications where the formulae that may participate are preset (e.g. they are drawn from a lexicon), formulae can be precompiled, although the results of precompilation would need to be parameterised with respect to the variables/indices appearing, with a sufficient supply of 'fresh' symbols being generated at time of lexical access, to ensure uniqueness.

$(\tau 1)$ $\tau(X_1{:}\,x_1, \ldots, X_n{:}\,x_n \Rightarrow X_0) = \langle \mathcal{G}, \phi, \Delta \rangle$
where $i_0, \ldots, i_n$ fresh indices; $\mathcal{G}$ a fresh goal atom; $\phi = \mathsf{indices}(\Delta)$
$\Delta = \tau(\langle i_0, \mathcal{G} \circ\!\!- X_0, \lambda y.y \rangle) \cup \tau(\langle i_1, X_1, x_1 \rangle) \cup \ldots \cup \tau(\langle i_n, X_n, x_n \rangle)$

$(\tau 2)$ $\tau(\langle \phi, X, s \rangle) = \langle \phi, X, s \rangle$ where X atomic

$(\tau 3)$ $\tau(\langle \phi, X \circ\!\!- Y, s \rangle) = \tau(\langle \phi, X \circ\!\!- (Y{:}\,\emptyset), s \rangle)$ where Y has no (inclusion) index set

$(\tau 4)$ $\tau(\langle \phi, X_1 \circ\!\!- (Y{:}\,\psi), s \rangle) = \langle \phi, X_2 \circ\!\!- (Y{:}\,\psi), \lambda x.t \rangle \cup \Gamma$
where Y is atomic; x a fresh variable; $\tau(\langle \phi, X_1, (s\,x) \rangle) = \langle \phi, X_2, t \rangle \uplus \Gamma$

$(\tau 5)$ $\tau(\langle \phi, X \circ\!\!- ((Y \circ\!\!- Z){:}\,\psi), s \rangle) = \tau(\langle \phi, X \circ\!\!- (Y{:}\,\pi), \lambda y.s(\lambda z.y) \rangle) \cup \tau(\langle i, Z, z \rangle)$
where i a fresh index; y, z fresh variables; $\pi = i \cup \psi$

$(\tau 6)$ $\tau(\langle \phi, X \circ\!\!- ((Y \otimes Z){:}\,\psi), s \rangle) = \tau(\langle \phi, X \circ\!\!- (\mathcal{G}{:}\,\pi), s \rangle) \cup \tau(\langle i, \mathcal{G} \circ\!\!- Y \circ\!\!- Z, \lambda z \lambda y.(y \otimes z) \rangle)$
where i a fresh index; $\mathcal{G}$ a fresh goal atom; y, z fresh variables; $\pi = i \cup \psi$

$(\tau 7)$ $\tau(\langle \phi, X \otimes Y, s \rangle) = \langle \phi, \mathcal{A} \circ\!\!- (\mathcal{A}{:}\,\{i, j\}), \lambda t.(\mathrm{E}^{\otimes}_{x,y}(s, t)) \rangle \cup \tau(\langle i, X, x \rangle) \cup \tau(\langle j, Y, y \rangle)$
where i, j fresh indices; x, y, t fresh variables; $\mathcal{A}$ a fresh variable over *atomic* types

Figure 1: The Compilation Procedure

The formulae (7–13) arise under combination. Formulae (11) and (13) correspond to successful overall analyses (i.e. have type g0, and are labelled with the full index set). The proof illustrates the possibility of multiple derivations assigning equivalent readings, i.e. (11) and (13) have identical proof terms, that arise by nondeterminism for involvement of formula (4).

5 Computing Exclusion Constraints

The use of *inclusion constraints* (i.e. requirements that some formula must be used in deriving a given functor's argument) within the approach allows us to ensure that hypotheticals are appropriately used in any overall deduction and hence that deductions are valid. However, the approach allows that deduction can generate some intermediate results that cannot be part of an overall deduction. For example, compiling a formula X∘–(Y∘–(Z∘–W))∘–(V∘–W) gives the first-order residue X∘–Y∘–V, plus hypotheticals Z∘–W and W. A partial deduction in which the hypothetical Z∘–W is used in deriving the argument V of X∘–Y∘–V cannot be extended to a successful overall deduction, since its use again for the functor's second argument Y (as an inclusion constraint will require) would violate linear usage. For the same reason, a direct combination of the hypotheticals Z∘–W and W is likewise a deductive dead end.

This problem can be addressed via *exclusion constraints*, i.e. annotations to forbid stated formulae having been used in deriving a given functor's argument, as proposed in (Hepple, 1998). Thus, a functor might have the form X∘–(Y:{i}:{j}) to indicate that i must appear in its argument's index set, and that j *must not*. Such exclusions can be straightforwardly computed over the set of compiled formulae that derive from each initial assumption, using simple (set-theoretic) patterns of reasoning. For example, for the case above, since W must be used in deriving the argument V of the main residue formula, it can be excluded from the argument Y of that formula (which follows from the disjointness condition on the single inference rule). Given that the argument Y must include Z∘–W, but excludes W, we can infer that W cannot contribute to the argument of Z∘–W, giving an exclusion constraint that (amongst other things) blocks the direct combination of Z∘–W and W. See (Hepple, 1998) for more details (although a somewhat different version of the first-order formalism is used there).

6 Tabular Deduction

A simple algorithm for use with the above approach, which avoids much redundant computation, is as follows. Given a possible theorem to prove, the results of compilation (i.e. in-

dexed types plus semantics) are gathered on an agenda. Then, a loop is followed in which an item is taken from the agenda and added to the database (which is initially empty), and then the next item is taken from the agenda and so on until the agenda is empty. Whenever an entry is added to the database, a check is made to see if it can combine with any that are already there, in which case new agenda items are generated. When the agenda is empty, a check is made for any successful overall analyses. Since the result of a combination always bears an index set larger than either parent, and since the maximal index set is fixed at compilation time, the above process must terminate.

However, there is clearly more redundancy to be eliminated here. Where two items differ only in their semantics, their subsequent involvement in any further deductions will be precisely parallel, and so they can be collapsed together. For this purpose, the semantic component of database entries is replaced with a unique identifier, which serves as a 'hook' for semantic alternatives. Agenda items, on the other hand, instead record the way that the item was produced, which is either 'presupplied' (by compilation) or 'by combination', in which case the entries combined are recorded by their identifiers. When an agenda item is added to the database, a check is made for an entry with the same indexed type. If there is none, a new entry is created and a check made for possible combinations (giving rise to new agenda items). However, if an appropriate existing entry is found, a record is made for that entry of an additional way to produce it, but no check made for possible combinations. If at the end there is a successful overall analysis, its unique identifier, plus the records of what combined to produce what, can be used to enumerate directly the proof terms for successful analyses.

7 Application #1: Categorial Parsing

The associative Lambek calculus (Lambek, 1958) is perhaps the most familiar representative of the class of categorial formalisms that fall within the 'type-logical' tradition. 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.[8] 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 natural 'base logic' on which to construct such systems is the multiplicative fragment of linear logic, since (i) it stands above the various categorial systems in the hierarchy of substructural logics, and (ii) its operators correspond to precisely those appearing in any standard categorial logic. The key requirement for parsing categorial systems formulated in this way is some theorem proving method that is sufficient for the fragment of linear logic employed (although some additional work will be required for managing labels), and a number of different approaches have been used, e.g. proof nets (Moortgat, 1992), and SLD resolution (Morrill, 1995). Hepple (1996) introduces first-order compilation for implicational linear logic, and shows how that method can be used with labelling as a basis parsing implicational categorial systems. No further complications arise for combining the extended compilation approach described in this paper with labelling systems as a basis for efficient, non-redundant parsing of categorial formalisms in the core multiplicative fragment. See (Hepple, 1996) for a worked example.

8 Application #2: Glue Language Deduction

In a line of research beginning with Dalrymple *et al.* (1993), a fragment of linear logic is used as a 'glue language' for assembling sentence meanings for LFG analyses in a 'deductive' fashion (enabling, for example, an direct treatment of quantifier scoping, without need of additional mechanisms). Some sample expressions:

hates:
$\forall X, Y.(s \rightsquigarrow_t hates(X,Y)) \multimap ((f \rightsquigarrow_e X) \otimes (g \rightsquigarrow_e Y))$

everyone: $\forall H, S.(H \rightsquigarrow_t every(person, S)) \multimap (\forall x.(H \rightsquigarrow_t S(x)) \multimap (g \rightsquigarrow_e x))$

The operator $\rightsquigarrow$ serves to pair together a 'role' with a meaning expression (whose semantic type is shown by a subscript), where a 'role' is essentially a node in a LFG f-structure. For

[8] See, for example, the formalisms developed in (Moortgat *et al.*, 1994), (Morrill, 1994), (Hepple, 1995).

our purposes roles can be treated as if they were just atomic symbols. For theorem proving purposes, the universal quantifiers above can be deleted: the uppercase variables can be treated as Prolog-like variables, which become instantiated under matching during proof construction; the lowercase variables can be replaced by arbitrary constants. Such deletion leaves a residue that can be treated as just expressions of multiplicative linear logic, with role/meaning pairs serving as 'basic formulae'.[9]

An observation contrasting the categorial and glue language approaches is that in the categorial case, all that is required of a deduction is the proof term it returns, which (for 'linguistic derivations') provides a 'semantic recipe' for combining the lexical meanings of initial formulae directly. However, for the glue language case, given the way that meanings are folded into the logical expressions, the lexical terms themselves must participate in a proof for the semantics of a LFG derivation to be produced.

Here is one way that the first-order compilation approach might be used for glue language deduction (other ways are possible). Firstly, we can take each (quantifier-free) glue term, replace each role/meaning pair with just the role component, and associate the resulting formula with a unique semantic variable. The set of formulae so produced can then undergo the first-order compilation procedure. Crucially for compilation, although some of the role expressions in the formulae may be ('Prolog-like') variables, they correspond to *atomic* formulae (so there is no 'hidden structure' that compilation cannot address). A complication here is that occurrences of a single role variable may end up in different first-order formulae. In any overall deduction, the binding of these multiple variable instances must be consistent, but we cannot rely on a global binding context, since alternative proofs will typically induce distinct (but internally consistent) bindings. Hence, bindings must be handled locally (i.e. relative to each database formula) and combinations will involve merging of local binding contexts. Each proof term that tabular deduction returns corresponds to a natural deduction proof over the precompilation formulae. If we mechanically mirror this pattern of proof over the original glue terms (with meanings, but quantifier-free), a role/meaning pair that provides a reading of the original LFG derivation will result.

[9] See (Fry, 1997), who uses a proof net method for glue language deduction, for relevant discussion. This paper also provides examples of glue language uses that require a full deductive system for the multiplicative fragment.

References

Nick Benton, Gavin Bierman, Valeria de Paiva & Martin Hyland. 1992. 'Term Assignment for Intuitionistic Linear Logic.' Tech. Report 262, Cambridge University Computer Lab.

Mary Dalrymple, John Lamping & Vijay Saraswat. 1993. 'LFG semantics via constraints.' *Proc. EACL–6*, Utrecht.

John Fry 1997. 'Negative Polarity Licensing at the Syntax-Semantics Interface.' *Proc. ACL/EACL–97 Joint Conference*, Madrid.

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

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

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

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

Mark Hepple. 1998. 'Linear Deduction via First-order Compilation.' *Proc. First Workshop on Tabulation in Parsing & Deduction.*

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

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

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.*

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.