Higher-order Linear Logic Programming of Categorial Deduction

Glyn Morrill

Secció d'Intel·ligència Artificial
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona
morrill@lsi.upc.es

Abstract

We show how categorial deduction can be implemented in higher-order (linear) logic programming, thereby realising parsing as deduction for the associative and non-associative Lambek calculi. This provides a method of solution to the parsing problem of Lambek categorial grammar applicable to a variety of its extensions.

The present work deals with the parsing problem for Lambek calculus and its extensions as developed in, for example, Moortgat (1988), van Benthem (1991), Moortgat and Morrill (1991), Moortgat and Oehrle (1993), Morrill (1994b) and Hepple (1995). Some previous approaches to parsing Lambek grammar such as König (1989), Hepple (1990) and Hendriks (1993) have concentrated on the possibilities of sequent proof normalisation. In Roorda (1991), Moortgat (1992), Hendriks (1993) and Oehrle (1994) a strategy of unfolding and labelling for proof net construction is considered. We aim to show here how such unfolding allows compilation into programs executable by a version of SLD resolution, implementing categorial deduction in dynamic linear clauses. The linearity resides in the use exactly once per word token of each of the clauses compiled from lexical categorisations. By dynamic, it is meant that clauses may be higher-order (they are hereditary Harrop Horn clauses) so that clausal resolution involves insertion in, as well as retraction from, the resolution database; see Miller et al. (1991), and Hodas and Miller (1994).

It is shown how a range of calculi can be treated by dealing with the highest common factor of connectives as linear logical validity. The prosodic (i.e. sublinear) aspects of word order and hierarchical structure are encoded in labels, in effect the term structure of quantified linear logic. Compiling labels according to interpretations in groupoids provides a general method for calculi with various structural properties and also for multimodal hybrid formulations. Unification must be carried out according to the structural axioms but is limited to *one-way* matching, i.e. one term is always ground. Furthermore, for the particular case of associative Lambek calculus an additional perspective of binary relational interpretation allows an especially efficient coding in which the span of expressions is represented in such a way as to avoid the computation of unifiers under associativity, and this can also be exploited for non-associative calculus.

Higher-order linear logic programming has already been applied to natural language processing in, for example, Hodas (1992) and Hodas and Miller (1994), in work deriving from Pareschi (1989) and Pareschi and Miller (1990). What we show here is that such implementation can be realised systematically, indeed by a mechanical compilation, while grammars themselves are written in higher level categorial grammar formalism.

Automated deduction for Lambek calculi is of interest in its own right but solution of the parsing problem for categorial logic allowing significant linguistic coverage demands automated deduction for more than just individual calculi. There is a need for methods applying to whole classes of systems in ways which are principled and powerful enough to support the further generalisations that grammar development will demand. We aim to indicate here how higher-order logic programming can provide for such a need.

After reviewing the "standard" approach, via sequent proof normalisation, we outline the relevant features of (linear) logic programming and explain compilation and execution for associative and non-associative calculi in terms of groupoid and binary relational interpretations of categorial connectives. We go on to briefly mention multimodal calculi for the binary connectives.

The parsing problem is usually construed as the recovery of structural descriptions assigned to strings by a grammar. In practice the interest is in computing semantic forms implicit in the structural descriptions, which are themselves usually implicit in the history of a derivation recognising well-formedness of a string. This is true in particular of compositional categorial architec-

tures and we shall focus on algorithms for showing well-formedness. The further step to computing semantics is unproblematic.

For the non-associative Lambek calculus **NL** of Lambek (1961) we assume types freely generated from a set of primitive types by binary (infix) operators \, / and •. A sequent comprises a succedent type A and an antecedent configuration Γ which is a binary bracketed list of one or more types; we write $\Gamma \Rightarrow A$. The notation $\Gamma(\Delta)$ here refers to a configuration Γ with a distinguished subconfiguration Δ.

a. $A \Rightarrow A$ id $\dfrac{\Gamma \Rightarrow A \quad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}$Cut   (1)

b. $\dfrac{\Gamma \Rightarrow A \quad \Delta(B) \Rightarrow C}{\Delta([\Gamma, A\backslash B]) \Rightarrow C}$\L $\dfrac{[A, \Gamma] \Rightarrow B}{\Gamma \Rightarrow A\backslash B}$\R

c. $\dfrac{\Gamma \Rightarrow A \quad \Delta(B) \Rightarrow C}{\Delta([B/A, \Gamma]) \Rightarrow C}$/L $\dfrac{[\Gamma, A] \Rightarrow B}{\Gamma \Rightarrow B/A}$/R

d. $\dfrac{\Gamma([A, B]) \Rightarrow C}{\Gamma(A{\bullet}B) \Rightarrow C}${\bullet}L $\dfrac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A{\bullet}B}${\bullet}R

For the associative Lambek calculus **L** of Lambek (1958) the types are the same. A sequent comprises a succedent type A and an antecedent configuration Γ which is a list of one or more types; again we write $\Gamma \Rightarrow A$.

a. $A \Rightarrow A$ id $\dfrac{\Gamma \Rightarrow A \quad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}$Cut   (2)

b. $\dfrac{\Gamma \Rightarrow A \quad \Delta(B) \Rightarrow C}{\Delta(\Gamma, A\backslash B) \Rightarrow C}$\L $\dfrac{A, \Gamma \Rightarrow B}{\Gamma \Rightarrow A\backslash B}$\R

c. $\dfrac{\Gamma \Rightarrow A \quad \Delta(B) \Rightarrow C}{\Delta(B/A, \Gamma) \Rightarrow C}$/L $\dfrac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow B/A}$/R

d. $\dfrac{\Gamma(A, B) \Rightarrow C}{\Gamma(A{\bullet}B) \Rightarrow C}${\bullet}L $\dfrac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A{\bullet}B}${\bullet}R

Lambek showed Cut-elimination for both calculi, i.e. every theorem has a Cut-free proof. Of the remaining rules each instance of premises has exactly one connective occurrence less than the corresponding conclusion so Cut-elimination shows decidability through finite space Cut-free sequent proof search from conclusions to premises. Lifting is derivable in **NL** as follows:

$$\dfrac{\dfrac{A \Rightarrow A \quad B \Rightarrow B}{[A, A\backslash B] \Rightarrow B}\backslash L}{A \Rightarrow B/(A\backslash B)}/R \qquad (3)$$

It is also derivable in **L**; indeed all **NL** derivations are converted to **L** derivations by simply

erasing the brackets. But **L**-derivable composition depends essentially on associativity and is not **NL**-derivable:

$$\dfrac{\dfrac{A \Rightarrow A \quad \dfrac{B \Rightarrow B \quad C \Rightarrow C}{B, B\backslash C \Rightarrow C}\backslash L}{A, A\backslash B, B\backslash C \Rightarrow C}\backslash L}{A\backslash B, B\backslash C \Rightarrow A\backslash C}\backslash R \qquad (4)$$

Even amongst the Cut-free proofs however there is still semantic equivalence under the Curry-Howard rendering (van Benthem, 1983; see Morrill, 1994b) and in this respect redundancy in parsing as exhaustive proof search since distinct lines of inference converge on common subproblems. This derivational equivalence (or: "spurious ambiguity") betrays the permutability of certain rule applications. Thus two left rules may be permutable: N/CN, CN, N\S $\Rightarrow$ S can be proved by choosing to work on either connective first. And left and right rules are permutable: N/CN, CN $\Rightarrow$ S/(N\S)) may be proved by applying a left rule first, or a right rule, (and the latter step then further admits the two options of the first example). Such non-determinism is not significant semantically: the variants have the same readings; the non-determinism in partitioning by the binary left rules in **L** is semantically significant, but still a source of inefficiency in its backward chaining "generate-and-test" incarnation. Another source of derivational equivalence is that a complex id axiom instance such as N\S $\Rightarrow$ N\S can be proved either by a direct matching against the axiom scheme, or by two rule applications. This is easily solved by restricting id to atomic formulas. More problematic are the permutability of rule applications, the non-determinism of rules requiring splitting of configurations in **L**, and the need in **NL** to hypothesise configuration structure a priori (such hierarchical structure is not given by the input to the parsing problem). It seems that only the first of these difficulties can be overcome from a Gentzen sequent perspective.

The situation regarding equivalence and rule ordering is solved, at least for **L**−{•L}, by sequent proof normalisation (König, 1989; Hepple, 1990; Hendriks, 1993):

a. $\boxed{A} \Rightarrow A$ id* $\dfrac{\Gamma_1, \boxed{A}, \Gamma_2 \Rightarrow B}{\Gamma_1, A, \Gamma_2 \Rightarrow \boxed{B}}$P*   (5)

b. $\dfrac{\Gamma \Rightarrow \boxed{A} \quad \Delta(\boxed{B}) \Rightarrow C}{\Delta(\Gamma, \boxed{A\backslash B}) \Rightarrow C}$\L* $\dfrac{A, \Gamma \Rightarrow \boxed{B}}{\Gamma \Rightarrow \boxed{A\backslash B}}$\R

c. $\dfrac{\Gamma \Rightarrow \boxed{A} \quad \Delta(\boxed{B}) \Rightarrow C}{\Delta(\boxed{B/A}, \Gamma) \Rightarrow C}$/L* $\dfrac{\Gamma, A \Rightarrow \boxed{B}}{\Gamma \Rightarrow \boxed{B/A}}$/R

This involves firstly ordering right rules before left rules reading from endsequent to axiom leaves

(so left rules only apply to sequents with atomic succedents; this effects uniform proof; see Miller et al., 1991), and secondly further demanding successive unfolding of the same configuration type ("focusing"). In the *-ed rules the succedent is atomic. A necessary condition for success is that an antecedent type is only selected by P^* if it yields the succedent atom as its eventual range. Let us refer to (5) as $\boxed{L}$. $\boxed{L}$ is free of spurious ambiguity, and $\vdash_L \Gamma \Rightarrow A$ iff $\vdash_{\boxed{L}} \Gamma \Rightarrow \boxed{A}$. The focusing strategy breaks down for $\bullet L$: $(VP/PP)/N, N\bullet PP \Rightarrow VP$ requires switching between configuration types. It happens that left occurrences of product are not motivated in grammar, but more critically sequent proof normalisation leaves the non-determinism of partitioning, and offers no general method for multimodal extensions which may have complex and interacting structural properties. To eliminate the splitting problem we need some kind of representation of configurations such that the domain of functors need not be hypothesised and then checked, but rather discovered by constraint propagation. Such is the character of our treatment, whereby partitioning is explored by unification in the term structure of higher-order linear logic programming, to which we now turn. By way of orientation we review the (propositional) features of clausal programming.

The first order case, naturally, corresponds to Prolog. Let us assume a set $\mathcal{ATOM}$ of atomic formulas, 0-ary, 1-ary, etc., formula constructors $\{\cdot \wedge \ldots \wedge \cdot\}_{n \in \{0,1,\ldots\}}$ and a binary (infix) formula constructor $\leftarrow$. A sequent comprises an agenda formula A and a database Γ which is a bag of program clauses $\{B_1, \ldots, B_n\}_m, n \geq 0$ (subscript m for multiset); we write $\Gamma \Rightarrow A$. In BNF, the set of agendas corresponding to the nonterminal $\mathcal{AGENDA}$ and the set of program clauses corresponding to the nonterminal $\mathcal{PCLS}$ are defined by:

$$\mathcal{AGENDA} ::= \mathcal{GOAL} \wedge \ldots \wedge \mathcal{GOAL} \qquad (6)$$
$$\mathcal{PCLS} ::= \mathcal{ATOM} \leftarrow \mathcal{AGENDA}$$

For first order programming the set $\mathcal{GOAL}$ of goals is defined by:

$$\mathcal{GOAL} ::= \mathcal{ATOM} \qquad (7)$$

Then execution is guided by the following rules.

$$\Gamma, A \Rightarrow A \quad \text{ax} \qquad (8)$$

I.e. the unit agenda is a consequence of any database containing its atomic clause.

$$\frac{\Gamma, A \leftarrow B_1 \wedge \ldots \wedge B_n \Rightarrow \begin{array}{c} B_1 \wedge \ldots \wedge B_n \wedge \\ C_1 \wedge \ldots \wedge C_m \end{array}}{\Gamma, A \leftarrow B_1 \wedge \ldots \wedge B_n \Rightarrow A \wedge C_1 \wedge \ldots \wedge C_m} \text{RES} \qquad (9)$$

I.e. we can resolve the first goal on the agenda with

the head of a program clause and then continue with the program as before and a new agenda given by prefixing the program clause subagenda to the rest of the original agenda (depth-first search).

For the higher-order case agendas and program clauses are defined as above, but the notion of $\mathcal{GOAL}$ on which they depend is generalised to include implications:

$$\mathcal{GOAL} ::= \mathcal{ATOM} \mid \mathcal{GOAL} \leftarrow \mathcal{PCLS} \qquad (10)$$

And a "deduction theorem" rule of inference is added:

$$\frac{\Gamma, B \Rightarrow A \qquad \Gamma \Rightarrow C_1 \wedge \ldots \wedge C_m}{\Gamma \Rightarrow (A \leftarrow B) \wedge C_1 \wedge \ldots \wedge C_m} \text{DT} \qquad (11)$$

I.e. we solve a higher-order goal first on the agenda by adding its precondition to the database and trying to prove its postcondition.

In linear logic programming the rules become resource conscious; in this context we write $\otimes$ for the conjunction and $\circ\!\!-$ for the implication:

$$A \Rightarrow A \quad \text{ax} \qquad (12)$$

I.e. an atomic agenda is a consequence of its unit database: all program clauses must be "used up" by the resolution rule:

$$\frac{\Gamma \Rightarrow B_1 \otimes \ldots \otimes B_n \otimes C_1 \otimes \ldots \otimes C_m}{\Gamma, A \circ\!\!- B_1 \otimes \ldots \otimes B_n \Rightarrow A \otimes C_1 \otimes \ldots \otimes C_m} \text{RES} \qquad (13)$$

I.e. a program clause disappears from the database once it is resolved upon: each is used exactly once. The deduction theorem rule for higher-order clauses also becomes sensitised to the employment of antecedent contexts:

$$\frac{\Gamma, B \Rightarrow A \qquad \Delta \Rightarrow C_1 \otimes \ldots \otimes C_m}{\Gamma, \Delta \Rightarrow (A \circ\!\!- B) \otimes C_1 \otimes \ldots \otimes C_m} \text{DT} \qquad (14)$$

We shall motivate compilation into linear clauses directly from simple algebraic models for the calculi. In the case of L we have first interpretation in semigroups $\langle L, + \rangle$ (i.e. sets L closed under associative binary operations $+$; intuitively: strings under concatenation). Relative to a model each type A has an interpretation as a subset $D(A)$ of L. Given that primitive types are interpreted as some such subsets, complex types receive their denotations by *residuation* as follows (cf. e.g. Lambek, 1988):

$$\begin{array}{rcl} D(A \bullet B) & = & \{s_1 + s_2 \mid s_1 \in D(A) \wedge s_2 \in D(B)\} \quad (15) \\ D(A \backslash B) & = & \{s \mid \forall s' \in D(A), s' + s \in D(B)\} \\ D(B/A) & = & \{s \mid \forall s' \in D(A), s + s' \in D(B)\} \end{array}$$

For the non-associative calculus we drop the condition of associativity and interpret in arbitrary

groupoids (intuitively: trees under adjunction[1]).

Categorial type assignment statements comprise a term α and a type A; we write $\alpha: A$. Given a set of lexical assignments, a phrasal assignment is projected if and only if in every model satisfying the lexical assignments the phrasal assignment is also satisfied. A categorial sequent has a translation given by $| \cdot |$ into a linear sequent of type assignments which can be safely read as predications. For **L** we have the following (**NL** preserves input antecedent configuration in output succedent term structure):

$$|B_0, \ldots, B_n \Rightarrow A| = \qquad (16)$$
$$k_0: B_0^+, \ldots, k_n: B_n^+ \Rightarrow k_0 + \ldots + k_n: A^-$$

Categorial type assignment statements are translated into linear logic according to the interpretation of types. The polar translation functions are identity functions on atomic assignments; on complex category predicates they are defined mutually as follows (for related unfolding, but for proof nets, see Roorda, 1991; Moortgat, 1992; Hendriks, 1993; and Oehrle, 1994); $\overline{p}$ indicates the polarity complementary to p:

$$\frac{\alpha+\gamma: B^p \quad \circ\!\!- \quad \alpha: A^{\overline{p}}}{\gamma: A\backslash B^p} \quad \begin{array}{l}\alpha \text{ new variable/} \\ \text{constant as } p \; +/- \end{array} \qquad (17)$$

$$\frac{\gamma+\alpha: B^p \quad \circ\!\!- \quad \alpha: A^{\overline{p}}}{\gamma: B/A^p} \quad \begin{array}{l}\alpha \text{ new variable/} \\ \text{constant as } p \; +/- \end{array}$$

The unfolding transformations have the same general form for the positive (configuration/database) and negative (succedent/agenda) occurrences; the polarity is used to indicate whether new symbols introduced for quantified variables in the interpretation clauses are metavariables (in italics) or Skolem constants (in boldface); we shall see examples shortly. The program clauses and agenda are read directly off the unfoldings, with the only manipulation being a flattening of positive implications into uncurried form:

$$((X^+ \circ\!\!- Y_1^-) \circ\!\!- \ldots) \circ\!\!- Y_n^- \; \triangleright \qquad (18)$$
$$X^+ \circ\!\!- Y_1^- \otimes \ldots \otimes Y_n^-$$

(This means that matching against the head of a clause and assembly of subgoals does not require any recursion or restructuring at runtime.) We shall also allow unit program clauses $X \circ\!\!-$ to be abbreviated X.

Starting from the initial database and agenda, a proof will be represented as a list of agendas, avoiding the context repetition of sequent proofs by indicating where the resolution rule retracts from the database (superscript coindexed overline), and where the deduction theorem rule adds to it (subscript coindexation):

---

[1]Though **NL** with product is incomplete with respect to finite trees as opposed to groupoids in general.

database	$\Gamma, \overline{A \circ\!\!- B_1 \otimes \ldots \otimes B_n}^i$	(19)
agenda		
i.	$A \otimes C_1 \otimes \ldots \otimes C_m$	RES
$i{+}1$.	$B_1 \otimes \ldots \otimes B_n \otimes C_1 \otimes \ldots \otimes C_m$	

database	Γ, B_i	(20)
agenda		
i.	$(A \circ\!\!- B) \otimes C_1 \otimes \ldots \otimes C_m$	DT
$i{+}1$.	$A \otimes C_1 \otimes \ldots \otimes C_m$	

The sharing of a Skolem constant between A and B in (20) ensures that B can and must be used to prove A so that a mechanism for the lazy splitting of contexts is effected. The termination condition is met by a unit agenda with its unit database.

By way of illustration for **L** consider composition given the sequent translation (21).

$$|A\backslash B, B\backslash C \Rightarrow A\backslash C| = \qquad (21)$$
$$k: A\backslash B^+, \; l: B\backslash C^+ \Rightarrow k{+}l: A\backslash C^-$$

The assignments are unfolded thus:

$$\frac{a{+}k: B \quad \circ\!\!- \quad a: A}{k: A\backslash B^+} \qquad \frac{b{+}l: C \quad \circ\!\!- \quad b: B(22)}{l: B\backslash C^+}$$

$$\frac{m{+}(k{+}l): C \quad \circ\!\!- \quad m: A}{k{+}l: A\backslash C^-}$$

Then the proof runs as follows.

database	$\overline{a{+}k: B \circ\!\!- a: A}^3,$	(23)
	$\overline{b{+}l: C \circ\!\!- b: B}^2,$	
	$\overline{m: A_1}^4$	
agenda		
1.	$m{+}(k{+}l): C \circ\!\!- m: A$	DT
2.	$m{+}(k{+}l): C$	RES $b = m{+}k$
3.	$m{+}k: B$	RES $a = m$
4.	$m: A$	RES

The unification at line 2 relies on associativity. Note that unifications are all one-way, but even one-way associative ($=$string) unification has expensive worst cases.

For **NL** the term labelling provides a clausal implementation with unification being nonassociative. Consider lifting:

$$|A \Rightarrow B/(A\backslash B)| = k: A \Rightarrow k: B/(A\backslash B) \qquad (24)$$

$$\frac{a{+}l: B \quad \circ\!\!- \quad a: A}{k{+}l: B \quad \circ\!\!- \quad l: A\backslash B^+} \qquad (25)$$
$$\frac{}{k: B/(A\backslash B)^-}$$

The proof is as follows.

database	$\overline{k: A}^3,$	(26)
	$\overline{a{+}l: B \circ\!\!- a: A_1}^2$	
agenda		
1.	$k{+}l: B \circ\!\!- (a{+}l: B \circ\!\!- a: A)$	DT
2.	$k{+}l: B$	RES $a = k$
3.	$k: A$	RES

136

The simple one-way term unification is very fast but it is unnatural from the point of view of parsing that, as for the sequent approach, a hierarchical binary structure on the input string needs to be posited before inference begins, and exhaustive search would require all possibilities to be tried. Later we shall see how hierarchical structure can be discovered rather than conjectured by factoring out horizontal structure.

Let us note here the relation to $\boxed{L}$. $\boxed{L}$ applies (working back from the target sequent) right rules before left rules. Here, when a higher-order goal is found on the agenda its precondition is added to the database by DT. This precedes applications of the RES rule (hence the uniformity character) which corresponds to the left sequent inferences. It applies when the agenda goal is atomic and picks out antecedent types which yields that atom (cf. the eventual range condition of $\boxed{L}$). The focusing character is embodied by creating in one step the objective of seeking all the arguments of an uncurried functor.

By way of further example consider the following in **L**, with terms and types as indicated.

(a book from which) the references are missing (27)

the references	are missing	(28)
r: N	m: $((S/(N\backslash S))\backslash S)/PP$	

$$\Rightarrow \text{r+m: S/PP}$$

We have compilation for 'are missing' as in Figure 1 yielding (29).

$\triangleright$ (29)
$b+(\text{m}+a)$: S o— $(b+$k: S o— $(c+$k: S o— c: N$))\otimes a$: PP

And the succedent unfolds as follows:

(r+m)+l: S   o—   l: PP   (30)
—————————————————
r+m: S/PP⁻

$\triangleright$ (r+m)+l: S o— l: PP

Derivation is as in figure 2. The unification at line 2 relies on associativity and as always atomic goals on the agenda are ground. But in general we have to try subproofs for different unifiers, that is, we effectively still have to guess partitioning for left rules. We shall see that this is not necessary, and that associative unification can be avoided.

There is a further problem which will be solved in the same move. Unfolding of left products would create two positive subformulas and thus fall outside the scope of Horn clause programming. However, the term-labelled implementation as it has been given also fails for right products:

α: A^- $\otimes$ β: B^- (31)
—————————————————$\gamma = \alpha + \beta$?
γ: $A \bullet B^-$

The problem is that α and β are not deterministically given by γ at the "compile time" of unfolding. The best we could manage seems to be to try different partitionings of γ at execution time; but even if this could work it would still amount to trying different partitionings for $\bullet$R as in the sequent calculus: a source of non-determinism we seek to reduce. This limitation combines with the other difficulties with groupoid labelling of worst case of (even) one-way associative unification for **L**, and the need for a priori hypothesis of non-associative structure for **NL**.

The method of solution resides in looking at an alternative model: the associative calculus has relational algebraic models (van Benthem, 1991) which interpret types as relations on some set V, i.e. as sets of ordered pairs. Given denotations for primitive types, those of compound types are fixed as subsets of $V \times V$ by:

$$
\begin{aligned}
D(A\backslash B) &= \{\langle v_2, v_3 \rangle | \forall \langle v_1, v_2 \rangle \in D(A), \quad (32)\\
&\qquad \langle v_1, v_3 \rangle \in D(B)\}\\
D(B/A) &= \{\langle v_1, v_2 \rangle | \forall \langle v_2, v_3 \rangle \in D(A),\\
&\qquad \langle v_1, v_3 \rangle \in D(B)\}\\
D(A\bullet B) &= \{\langle v_1, v_3 \rangle | \exists v_2, \langle v_1, v_2 \rangle \in D(A) \, \&\\
&\qquad \langle v_2, v_3 \rangle \in D(B)\}
\end{aligned}
$$

Points in V intuitively corresponds to string positions (as in definite clause grammars, and charts) and ordered pairs to the vertices of substrings pertaining to the categories to which they are assigned. This induces unfolding as follows:

$i - k$: B^p o— $i - j$: $A^{\overline{p}}$ i new variable/   (33)
—————————————————— constant as p +/—
$j - k$: $A\backslash B^p$

$i - k$: B^p o— $j - k$: $A^{\overline{p}}$ k new variable/
—————————————————— constant as p +/—
$i - j$: B/A^p

Furthermore right product (though still not non-Horn left product) unfolding can be expressed:

$i - j$: A^- $\otimes$ $j - k$: B^- (34)
——————————————————j new variable
$i - k$: $A\bullet B^-$

Composition is now treated as follows. Assume sequent translation thus:

$|A\backslash B, B\backslash C \Rightarrow A\backslash C| =$ (35)
$0 - 1$: $A\backslash B^+$, $1 - 2$: $B\backslash C^+ \Rightarrow 0 - 2$: $A\backslash C^-$

The assignments are compiled as shown in (36).

$i - 1$: B   o— $i - 0$: A   (36)
—————————————
$0 - 1$: $A\backslash B^+$

$j - 2$: C   o— $j - 1$: B $3 - 2$: C   o— $3 - 0$: A
——————————————   ——————————————
$1 - 2$: $B\backslash C^+$ $0 - 2$: $A\backslash C^-$

The proof is thus:

137

$$\frac{\dfrac{c+k:\ S\ \circ\!\!-\ c:\ N}{b+k:\ S\ \circ\!\!-\ k:\ N\backslash S^{+}}}{\dfrac{b+(m+a):\ S\ \circ\!\!-\ b:\ S/(N\backslash S)^{-}}{\dfrac{m+a:\ (S/(N\backslash S))\backslash S^{+}\ \circ\!\!-\ a:\ PP}{m:\ ((S/(N\backslash S))\backslash S)/PP^{+}}}}$$

Figure 1: Groupoid compilation of the assignment to 'are missing'

database $\quad \overline{r:\ N}^{5}$,

$\overline{b+(m+a):\ S\circ\!\!-\ (b+k:\ S\circ\!\!-\ (c+k:\ S\circ\!\!-\ c:\ N))\otimes a:\ PP}^{-2}$,

$\overline{1:\ PP_1}^{6}$,

$\overline{c+k:\ S\circ\!\!-\ c:\ N_3}^{4}$,

agenda

1.	$(r+m)+1:\ S\circ\!\!-\ 1:\ PP$	DT
2.	$(r+m)+1:\ S$	RES $b=r$, $a=1$
3.	$(r+k:\ S\circ\!\!-\ (c+k:\ S\circ\!\!-\ c:\ N))\otimes 1:\ PP$	DT
4.	$r+k:\ S\otimes 1:\ PP$	RES $c=r$
5.	$r:\ N\otimes 1:\ PP$	RES
6.	$1:\ PP$	RES

Figure 2: Groupoid execution for 'the references are missing'

database $\quad \overline{i-1:\ B\circ\!\!-\ i-0:\ A}^{3}$, $\qquad$ (37)

$\overline{j-2:\ C\circ\!\!-\ j-1:\ B}^{2}$,

$\overline{3-0:\ A_1}^{4}$

agenda

1.	$3-2:\ C\circ\!\!-\ 3-0:\ A$	DT
2.	$3-2:\ C$	RES $j=3$
3.	$3-1:\ B$	RES $i=3$
4.	$3-0:\ A$	RES

In this way associative unification is avoided; indeed the only matching is trivial unification between constants and variables. So for **L** the relational compilation allows partitioning by the binary rules to be discovered by simple constraint propagation rather than by the generate-and-test strategy of normalised sequent proof.

Although the (one-way) term unification for groupoid compilation of the non-associative calculus is very fast we want to get round the fact that a hierarchical binary structure on the input string needs to be posited before inference begins. We can do this through observation of the following:

• All non-associative theorems are associative theorems (ignore brackets)

• Interpret non-associative operators in the product algebra of **NL** groupoid algebra and **L** relational algebra, and perform labelled compilation accordingly

• Use the (efficient) relational labelling to check associative validity

• Use the groupoid labelling to check non-associative validity and compute the prosodic form induced

I.e. the endsequent succedent groupoid term can

be left as a variable and the groupoid unification performed on the return trip from axiom leaves after associative validity has been assured, as will be seen in our final example. The groupoid unification will now be one-way in the opposite direction.

The simultaneous compilation separates horizontal structure (word order) represented by interval segments, and horizontal-and-vertical structure (linear and hierarchical organisation) represented by groupoid terms, and uses the efficient segment labelling to compute **L**-validity, and then the term labelling both to check the stricter **NL**-validity, and to calculate the hierarchical structure. In this way we use the fact that models for **NL** are given by intersection in the product of relational and groupoid models. Each type A has an interpretation $D(A)$ as a subset of $L \times V \times V$:

$$D(A\backslash B) = \{\langle s,v_2,v_3\rangle | \forall \langle s',v_1,v_2\rangle \in D(A),$$
$$\langle s'+s,v_1,v_3\rangle \in D(B)\} \quad (38)$$
$$D(B/A) = \{\langle s,v_1,v_2\rangle | \forall \langle s',v_2,v_3\rangle \in D(A),$$
$$\langle s+s',v_1,v_3\rangle \in D(B)\}$$
$$D(A\bullet B) = \{\langle s_1+s_2,v_1,v_3\rangle | \exists v_2,\langle s_1,v_1,v_2\rangle \in D(A)$$
$$\&\ \langle s_2,v_2,v_3\rangle \in D(B)\}$$

Unfolding is thus:

$$\frac{\alpha+\gamma\text{-}i\text{-}k:\ B^p\quad\circ\!\!-\quad\alpha\text{-}i\text{-}j:\ A^{\bar{p}}}{\gamma\text{-}j\text{-}k:\ A\backslash B^p} \quad (39)$$
α, i new variables/constants as $p +/-$

$$\frac{\gamma+\alpha\text{-}i\text{-}k:\ B^p\quad\circ\!\!-\quad\alpha\text{-}j\text{-}k:\ A^{\bar{p}}}{\gamma\text{-}i\text{-}j:\ B/A^p}$$
α, k new variables/constants as $p +/-$

$$\frac{\alpha\text{-}i\text{-}j:\ A^{-}\quad\otimes\quad\beta\text{-}j\text{-}k:\ B^{-}}{\alpha+\beta\text{-}i-k:\ A\bullet B^{-}}$$
α, β, j new variables

$$c\text{+}k\text{-}l\text{-}4\colon S \quad \text{o--} \quad c\text{-}l\text{-}1\colon N$$
$$b\text{+}k\text{-}i\text{-}4\colon S \quad \text{o--} \quad k\text{-}1\text{-}4\colon N\backslash S^+$$
$$b\text{+}(m\text{+}a)\text{-}i\text{-}k_1\colon S \quad \text{o--} \quad b\text{-}i\text{-}1\colon S/(N\backslash S)^-$$
$$m\text{+}a\text{-}1\text{-}k_1\colon (S/(N\backslash S))\backslash S^+ \qquad\qquad \text{o--} \quad a\text{-}2\text{-}k_1\colon PP$$
$$m\text{-}1\text{-}2\colon ((S/(N\backslash S))\backslash S)/PP^+$$

Figure 3: Groupoid-relational compilation of the assignment to 'are missing'

database $\quad \overline{\text{r-0-1}\colon N}^4$,

$\overline{b\text{+}(m\text{+}a)\text{-}i\text{-}k_1\colon S\,\text{o--}\,(b\text{+}k\text{-}i\text{-}4\colon S\,\text{o--}\,(c\text{+}k\text{-}l\text{-}4\colon S\,\text{o--}\,c\text{-}l\text{-}1\colon N))\otimes a\text{-}2\text{-}k_1\colon PP}^1$,

$\overline{c\text{+}k\text{-}l\text{-}4\colon S\,\text{o--}\,c\text{-}l\text{-}1\colon N_2}^{-3}$,

$\overline{\text{f-2-3}\colon PP}^5$

agenda

1.	d-0-3: S	RES $d = b\text{+}(m\text{+}a)$
2.	$(b\text{+}k\text{-}0\text{-}4\colon S\,\text{o--}\,(c\text{+}k\text{-}l\text{-}4\colon S\,\text{o--}\,c\text{-}l\text{-}1\colon N))\otimes a\text{-}2\text{-}3\colon PP$	DT
3.	$b\text{+}k\text{-}0\text{-}4\colon S\otimes a\text{-}2\text{-}3\colon PP$	RES $b = c$
4.	$c\text{-}0\text{-}1\colon N\otimes a\text{-}2\text{-}3\colon PP$	RES $c = r$
5.	$a\text{-}2\text{-}3\colon PP$	RES $a = f$

Figure 4: Groupoid-relational execution for 'the references are missing from this book'

By way of example consider the following:

the references	are missing	(40)
r-0-1: N	m-1-2: ((S/(N\S))\S)/PP	

from this book	
f-2-3: PP	$\Rightarrow$ d-0-3: S

The unfolding compilation yielding (41) for 'are missing' is given in Figure 3.

$\triangleright$ $\qquad\qquad\qquad\qquad\qquad\qquad$ (41)
$b\text{+}(m\text{+}a)\text{-}i\text{-}k_1\colon S\,\text{o--}$
$(b\text{+}k\text{-}i\text{-}4\colon S\,\text{o--}\,(c\text{+}k\text{-}l\text{-}4\colon S\,\text{o--}\,c\text{-}l\text{-}1\colon N))\otimes a\text{-}2\text{-}k_1\colon PP$

The derivation is given in Figure 4. Note how the term unification computing the hierarchical structure can be carried out one-way in the reverse order to the forward segment matchings:

$d = b\text{+}(m\text{+}a) = c\text{+}(m\text{+}a) = r\text{+}(m\text{+}a) =$ $\qquad$ (42)
$r\text{+}(m\text{+}f)$

In the case of **NL**-invalidity the term unification would fail.

We mention finally multimodal generalisations. In multimodal calculi families of connectives $\{/_i, \backslash_i, \bullet_i\}_{i\in\{1,\dots,n\}}$ are each defined by residuation with respect to their adjunction in a "polygroupoid" $\langle L, \{+_i\}_{i\in\{1,\dots,n\}}\rangle$ (Moortgat and Morrill, 1991):

$$
\begin{aligned}
D(A\bullet_i B) &= \{s_1\text{+}_i s_2 | s_1 \in D(A) \wedge s_2 \in D(B)\} \ (43)\\
D(A\backslash_i B) &= \{s | \forall s' \in D(A), s'\text{+}_i s \in D(B)\}\\
D(B/_i A) &= \{s | \forall s' \in D(A), s\text{+}_i s' \in D(B)\}
\end{aligned}
$$

Multimodal groupoid compilation for implications is immediate:

$$\frac{\alpha\text{+}_i\gamma\colon B^p \quad \text{o--} \quad \alpha\colon A^{\overline{p}}}{\gamma\colon A\backslash_i B^p} \quad \begin{array}{l}\alpha \text{ new variable}/\\ \text{constant as } p \text{ +}/-\end{array} \quad (44)$$

$$\frac{\gamma\text{+}_i\alpha\colon B^p \quad \text{o--} \quad \alpha\colon A^{\overline{p}}}{\gamma\colon B/_i A^p} \quad \begin{array}{l}\alpha \text{ new variable}/\\ \text{constant as } p \text{ +}/-\end{array}$$

This is entirely general. Any multimodal calculus can be implemented this way provided we have a (one-way) unification algorithm specialised according to the structural communication axioms. For example Morrill (1993) deals with multimodality for discontinuity which involves varying internal structural properties (associativity vs. non-associativity) as well as "split/wrap" interaction between modes. This is treated computationally in the current manner in Morrill (1994a) which also considers head-oriented discontinuity and unary operators projecting bracketed string structure. In these cases also simultaneous compilation including binary relational labelling can provide additional advantages.

Labelled unfolding of categorial formulas has been invoked in the references cited as a way of checking well-formedness of proof nets for categorial calculi by unification of labels on linked formulas. This offers improvements over sequent formulations but raises alternative problems; for example associative unification in general can have infinite solutions and is undecidable. Taking linear validity as the highest common factor of sublinear categorial calculi we have been able to show a strategy based on resolution in which the flow of information is such that one term in unification is always ground. Furthermore binary relational labelling propagates constraints in such a way that

computation of unifiers may be reduced to a subset of cases or avoided altogether. Higher-order coding allows emission of hypotheticals to be postponed until they are germane. Simultaneous compilation allows a factoring out of horizontal structure from vertical structure within the sublinear space in such a way that the partial information of word order can drive computation of hierarchical structure for the categorial parsing problem in the presence of non-associativity. The treatments for the calculi above and their multimodal generalisations have been implemented in Prolog (Morrill, 1994a).

References

van Benthem, Johan: 1983, 'The Semantics of Variety in Categorial Grammar', Report 83-29, Department of Mathematics, Simon Fraser University, also in Buszkowski, W., W. Marciszewski, and J. van Benthem (eds.): 1988, *Categorial Grammar*, Linguistic & Literary Studies in Eastern Europe Volume 25, John Benjamins, Amsterdam, 37–55.

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

Hendriks, Herman: 1993, *Studied Flexibility: Categories and Types in Syntax and Semantics*, Ph.D dissertation, Institute for Logic, Language and Computation, Universiteit van Amsterdam.

Hepple, Mark: 1990, *The Grammar and Processing of Order and Dependency: A Categorial Approach*, Ph.D. dissertation, University of Edinburgh.

Hepple, Mark: 1995, 'Mixing Modes of Linguistic Description in Categorial Grammar', this volume.

Hodas, J.: 1992, 'Specifying Filler-Gap Dependency Parsers in a Linear-Logic Programming Language', in *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 622–636.

Hodas, Joshua and Dale Miller: 1994, 'Logic Programming in a Fragment of Intuitionistic Linear Logic', to appear in *Journal of Information and Computation*.

König, E.: 1989, 'Parsing as natural deduction', in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Vancouver.

Lambek, J.: 1958, 'The mathematics of sentence structure', *American Mathematical Monthly* **65**, 154–170, also in Buszkowski, W., W. Marciszewski, and J. van Benthem (eds.): 1988, *Categorial Grammar*, Linguistic & Literary Studies in Eastern Europe Volume 25, John Benjamins, Amsterdam, 153–172.

Lambek, J.: 1961, 'On the calculus of syntactic types', in R. Jakobson (ed.) *Structure of language and its mathematical aspects*, Proceedings of the Symposia in Applied Mathematics **XII**, American Mathematical Society, 166–178.

Lambek, J.: 1988, 'Categorial and Categorical Grammars', in Richard T. Oehrle, Emmon Bach, and Deidre Wheeler (eds.) *Categorial Grammars and Natural Language Structures*, Studies in Linguistics and Philosophy Volume 32, D. Reidel, Dordrecht, 297–317.

Miller, D., G. Nadathur, F. Pfenning, and A. Scedrov: 1991, 'Uniform Proofs as a Foundation for Logic Programming', *Annals of Pure and Applied Logic* **51**, 125–157.

Moortgat, Michael: 1988, *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*, Foris, Dordrecht.

Moortgat, Michael: 1992, 'Labelled Deductive Systems for categorial theorem proving', OTS Working Paper OTS-WP-CL-92-003, Rijksuniversiteit Utrecht, also in *Proceedings of the Eighth Amsterdam Colloquium*, Institute for Language, Logic and Information, Universiteit van Amsterdam.

Moortgat, Michael and Glyn Morrill: 1991, 'Heads and Phrases: Type Calculus for Dependency and Constituent Structure', to appear in *Journal of Language, Logic, and Information*.

Moortgat, Michael and Dick Oehrle: 1994, 'Adjacency, dependency and order', in *Proceedings of the Ninth Amsterdam Colloquium*, 447–466.

Morrill, Glyn: 1993, *Discontinuity and Pied-Piping in Categorial Grammar*, Report de Recerca LSI-93-18-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, to appear in *Linguistics and Philosophy*.

Morrill, Glyn: 1994a, 'Higher-Order Linear Logic Programming of Categorial Deduction', Report de Recerca LSI-94-42-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya

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

Oehrle, Dick: 1994, 'Term labelled categorial type systems', to appear in *Linguistics and Philosophy*.

Pareschi, R.: 1989, *Type-driven Natural Language Analysis*, Ph.D. thesis, University of Edinburgh.

Pareschi, R. and D, Miller: 1990, 'Extending Definite Clause Grammars with Scoping Constructs', in D.H.D. Warren and P. Szeredi (eds.) *1990 International Conference in Logic Programming*, MIT Press, 373–389.

Roorda, Dirk: 1991, *Resource Logics: proof-theoretical investigations*, Ph.D. dissertation, Universiteit van Amsterdam.