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. We illustrate categorial calculi of discontinuity, and calculi with unary 'bracket' operators.

Higher-order Linear Logic Programming of Categorial Deduction

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), Morrill (1994a), Hepple (1993) and Moortgat and Oehrle (1993). Some previous approaches for Lambek calculus such as König (1989), Hepple (1990) and Hendriks (1993) have concentated on the possibilities of sequent proof normalisation. In Roorda (1991), Moortgat (1992), Hendriks (1993), Oehrle (1994) and Morrill (1994b) 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 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 e.g. Nadathur and Miller (1990), 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. Morrill (1994b) shows how 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. The present paper improves on this by moving from first-order to higher-order clauses which initiate hypothetical reasoning as it becomes germane at execution time rather than by inflating the resolution database with a first-order precompilation. 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. The method is extended to include unary categorial operators projecting vertical structure and thereby specifying constraints (Morrill 1992, 1994a, c; Moortgat 1994), and also to include discontinuity operators suited to the characterisation of wrapping functors and in situ binders such as quantifier phrases (Moortgat 1988, 1990; Morrill and Solias 1993; Morrill 1993, 1994a).

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 linear 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 for associative and non-associative calculi in terms of groupoid and binary relational interpretations of categorial connectives. We then go on to consider multimodal calculi for the binary connectives and address in particular discontinuity calculus. Finally we treat structural inhibition in partially associative calculus with unary operators.

1 Parsing as deduction in Lambek calculus

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 architectures and we shall focus on algorithms for showing well-formedness.

1.1 Non-associative sequent calculus

For the non-associative Lambek calculus **NL** of Lambek (1961) we assume types freely generated from a set $\mathcal{P}$ of primitive types by binary (infix) operators $\backslash$, $/$ and $\bullet$. A sequent comprises a succedent type A and a 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.

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

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

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

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

1.2 Associative sequent calculus

For the associative Lambek calculus **L** of Lambek (1958) let us again assume types freely generated from a set $\mathcal{P}$ of primitive types by binary (infix) operators $\backslash$, $/$ and $\bullet$. A sequent comprises a succedent type A and a configuration Γ which is a list of one or more types; we write $\Gamma \Rightarrow A$.

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

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

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

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

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.

1.3 Examples: lifting and composition

Lifting is derivable in **NL** as follows:

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

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:

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

Even amongst the Cut-free proofs however there is still semantic equivalence under the Curry-Howard rendering (van Benthem 1983) and in this respect redundancy in parsing as exhaustive proof search since the same readings will be found repeatedly. This derivational equivalence (or: "spu-

rious ambiguity") betrays the permutability of certain rule applications (both left/left e.g. N/CN, CN, N\S ⇒ S, and left/right, e.g. N/CN, CN ⇒ S/(N\S)); the non-determinism in partitioning by binary rules 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 ⇒ 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.

2 Uniform proof methods

The situation regarding equivalence and rule ordering is solved, at least for **L**−{•L}, by proof normalisation (König 1989, Hepple 1990, Hendriks 1993). This involves firstly ordering right rules before left rules reading from end sequent 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 (so that a necessary condition for success is that a left rule is only tried on a type which eventually yields the succedent atom as its range). The latter strategy breaks down for •L: (VP/PP)/N, N•PP ⇒ VP requires switching between configuration types. It happens that left occurrences of product are not motivated in grammar, but more critically 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.

3 Logic programming

By way of orientation let us 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 bag (or: multiset) database $\Gamma = \{B_1, \ldots, B_n\}_m, n \geq 0$ of program clauses; we write $\Gamma \Rightarrow A$. The set $\mathcal{AGENDA}$ of agendas is defined by:

(5) $\mathcal{AGENDA} ::= \mathcal{GOAL} \wedge \ldots \wedge \mathcal{GOAL}$

The set $\mathcal{PCLS}$ of program clauses is defined by:

(6) $\mathcal{PCLS} ::= \mathcal{ATOM} \leftarrow \mathcal{AGENDA}$

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

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

Then program execution is guided by the following rules.

(8) $$\frac{}{\Gamma \Rightarrow}\text{ax}$$

I.e. the empty agenda is a consequence of any database.

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

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:

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

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

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

I.e. we solve a higher-order goal first on the agenda by adding its antecedent to the database.

In linear logic programming all is as before, but the axiom and resolution rule become resource conscious; in this context we write $\otimes$ for the conjunction and $\circ\!\!-$ for the implication:

$$(12)\qquad \frac{}{\Rightarrow}\mathrm{ax}$$

I.e. the empty agenda is a consequence of the empty database; all program clauses must be "used up" by the resolution rule:

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

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:

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

4 Groupoid compilation

We shall motivate compilation into linear clauses directly from 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 $+$). 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):

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

For the non-associative calculus we drop the condition of associativity and interpret in arbitary groupoids.

Categorial type assignment statements comprise a groupoid term α and a type A; we write α: 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. Categorial type assignment statements are translated into linear logic according to the interpretation of types. A categorial sequent has a translation given by $|\cdot|$ into a linear sequent in which type assignments can be safely read as predications. For **L** we have the following (**NL** preserves input antecedent configuration in output succedent term structure):

$$(16) \quad |B_0, \ldots, B_n \Rightarrow A| = \mathbf{k}_0\colon B_0^+, \ldots, \mathbf{k}_n\colon B_n^+ \Rightarrow \mathbf{k}_0{+}\ldots{+}\mathbf{k}_n\colon A^-$$

The polar translation functions are identity functions on atomic assignements; 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:

(17) a. $$\frac{\alpha{+}\gamma\colon B^p \qquad \circ\!\!- \qquad \alpha\colon A^{\overline{p}}}{\gamma\colon A\backslash B^p}\alpha \text{ new variable/constant as } p\ +/-$$

b. $$\frac{\gamma{+}\alpha\colon B^p \qquad \circ\!\!- \qquad \alpha\colon A^{\overline{p}}}{\gamma\colon B/A^p}\alpha \text{ new variable/constant as } p\ +/-$$

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 or Skolem constants. The program clauses and agenda are read directly off the unfoldings, with the

only manipulation being a flattening of positive implications into uncurried form:

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

We shall also allow unit program clauses $X \circ\!\!-$ to be abbreviated X.

5 Examples

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):

(19)

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

(20)

database	Γ, B_i	
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 an empty agenda and empty database.

5.1 Associative case

Composition:

(21) |A\B, B\C ⇒ A\C|=
k: A\B$^+$, **l**: B\C$^+$ ⇒ **k**+**l**: A\C$^-$

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

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

(22) database $\overline{a{+}\mathbf{k}\text{: B} \circ\!\!- a\text{: A}}^{3}$,
$\overline{b{+}\mathbf{l}\text{: C} \circ\!\!- b\text{: B}}^{2}$,
$\overline{\mathbf{m}\text{: A}_1}^{4}$

agenda		
1.	$\mathbf{m}{+}\mathbf{k}{+}\mathbf{l}\text{: C} \circ\!\!- \mathbf{m}\text{: A}$	DT
2.	$\mathbf{m}{+}\mathbf{k}{+}\mathbf{l}\text{: C}$	RES $b = \mathbf{m}{+}\mathbf{k}$
3.	$\mathbf{m}{+}\mathbf{k}\text{: B}$	RES $a = \mathbf{m}$
4.	$\mathbf{m}\text{: A}$	RES

Note that unifications are all one-way, but even one-way associative (=string) unification has expensive worst cases.

5.2 Non-associative case

Models are provided as with the semigroups for $\mathbf{L}{-}\{\bullet\}$, but with groupoids $\langle L, + \rangle$. Then the term labelling provides a clausal implementation of $\mathbf{NL}{-}\{\bullet\}$ with unification now being non-associative.

Lifting:

(23) $|\text{A} \Rightarrow \text{B}/(\text{A}\backslash\text{B})|=$
$\mathbf{k}\text{: A} \Rightarrow \mathbf{k}\text{: B}/(\text{A}\backslash\text{B})$

$$\frac{\mathbf{k}{+}\mathbf{l}\text{: B} \quad \circ\!\!- \quad \dfrac{a{+}\mathbf{l}\text{: B} \quad \circ\!\!- \quad a\text{: A}}{\mathbf{l}\text{: A}\backslash\text{B}}}{\mathbf{k}\text{: B}/(\text{A}\backslash\text{B})}$$

(24) database $\overline{\mathbf{k}\colon A}^{3}$, $\overline{a{+}\mathbf{l}\colon \mathrm{B} \circ\!\!- a\colon A_1}^{1}$

agenda		
1.	$\mathbf{k}{+}\mathbf{l}\colon \mathrm{B} \circ\!\!- (a{+}\mathbf{l}\colon \mathrm{B} \circ\!\!- a\colon A)$	DT
2.	$\mathbf{k}{+}\mathbf{l}\colon \mathrm{B}$	RES $a = \mathbf{k}$
3.	$\mathbf{k}\colon \mathrm{A}$	RES

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.

5.3 Further example

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

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

(26)
$$\frac{\text{the references}}{\mathbf{r}\colon \mathrm{N}} \qquad \frac{\text{are missing}}{\mathbf{m}\colon ((\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S})/\mathrm{PP}} \qquad \Rightarrow \mathbf{r}{+}\mathbf{m}\colon \mathrm{S}/\mathrm{PP}$$

Clearly 'the references' yields the unit clause $\mathbf{r}\colon \mathrm{N}$. In addition we have:

(27)
$$\dfrac{b{+}(\mathbf{m}{+}a)\colon \mathrm{S}^{+} \quad \circ\!\!- \quad \dfrac{b\colon \mathrm{S}/(\mathrm{N}\backslash\mathrm{S})^{-}}{\mathbf{m}{+}a\colon (\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S}^{+}} \qquad \dfrac{b{+}\mathbf{k}\colon \mathrm{S}^{-} \quad \circ\!\!- \quad \dfrac{c{+}\mathbf{k}\colon \mathrm{S}^{+} \quad \circ\!\!- \quad c\colon \mathrm{N}^{-}}{\mathbf{k}\colon \mathrm{N}\backslash\mathrm{S}^{+}}}{\mathbf{m}\colon ((\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S})/\mathrm{PP}^{+}} \quad \circ\!\!- \quad a\colon \mathrm{PP}^{-}}$$

(28) $\blacktriangleright\, b{+}(\mathbf{m}{+}a)\colon \mathrm{S} \circ\!\!- (b{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- (c{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- c\colon \mathrm{N})) \otimes a\colon \mathrm{PP}$

(29) $$\frac{(\mathbf{r}{+}\mathbf{m}){+}\mathbf{l}\colon \mathrm{S}^{-} \quad \circ\!\!- \quad \mathbf{l}\colon \mathrm{PP}^{+}}{\mathbf{r}{+}\mathbf{m}\colon \mathrm{S}/\mathrm{PP}^{-}} \quad \triangleright\ (\mathbf{r}{+}\mathbf{m}){+}\mathbf{l}\colon \mathrm{S} \circ\!\!- \mathbf{l}\colon \mathrm{PP}$$

Derivation is thus:

(30) database $\overline{\mathbf{r}\colon \mathrm{N}}^{5}$,
$\overline{b{+}(\mathbf{m}{+}a)\colon \mathrm{S} \circ\!\!- (b{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- (c{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- c\colon \mathrm{N})) \otimes a\colon \mathrm{PP}}^{2}$,
$\overline{\mathbf{l}\colon \mathrm{PP}_1}^{6}$,
$\overline{c{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- c\colon \mathrm{N}_3}^{4}$,

	agenda	
1.	$(\mathbf{r}{+}\mathbf{m}){+}\mathbf{l}\colon \mathrm{S} \circ\!\!- \mathbf{l}\colon \mathrm{PP}$	DT
2.	$(\mathbf{r}{+}\mathbf{m}){+}\mathbf{l}\colon \mathrm{S}$	RES
3.	$(\mathbf{r}{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- (c{+}\mathbf{k}\colon \mathrm{S} \circ\!\!- c\colon \mathrm{N})) \otimes \mathbf{l}\colon \mathrm{PP}$	DT
4.	$\mathbf{r}{+}\mathbf{k}\colon \mathrm{S} \otimes \mathbf{l}\colon \mathrm{PP}$	RES
5.	$\mathbf{r}\colon \mathrm{N} \otimes \mathbf{l}\colon \mathrm{PP}$	RES
6.	$\mathbf{l}\colon \mathrm{PP}$	RES

The unification in passing from 3 to 4 relies on the associativity of + and as always atomic goals on the agenda are ground. But we have to try subproofs for different unifiers. We shall see shortly that this is not necessary, and that associative unification can be avoided.

6 Product

There is a further problem which we shall solve in the same move. Unfolding of left occurrences of product 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:

(31) $$\frac{\alpha\colon A^{-} \quad \otimes \quad \beta\colon B^{-}}{\gamma\colon A \bullet B^{-}}\gamma = \alpha{+}\beta?$$

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**.

7 Relational compilation

The associative calculus has also 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:

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

This induces unfolding as follows:

(33) a. $$\frac{i - k\text{: } B^p \qquad \circ\!\!- \qquad i - j\text{: } A^{\overline{p}}}{j - k\text{: } A\backslash B^p} i \text{ new variable/constant as } p\ +/-$$

b. $$\frac{i - k\text{: } B^p \qquad \circ\!\!- \qquad j - k\text{: } A^{\overline{p}}}{i - j\text{: } B/A^p} k \text{ new variable/constant as } p\ +/-$$

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

(34) $$\frac{i - j\text{: } A^- \qquad \otimes \qquad j - k\text{: } B^-}{i - k\text{: } A \bullet B^-} j \text{ new variable}$$

Composition is now treated thus:

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

$$\frac{i - 1\text{: B} \qquad \circ\!\!- \qquad i - 0\text{: A}}{0 - 1\text{: A}\backslash\mathrm{B}^+} \qquad \frac{j - 2\text{: C} \qquad \circ\!\!- \qquad j - 1\text{: B}}{1 - 2\text{: B}\backslash\mathrm{C}^+}$$

$$\frac{3-2\text{: C} \qquad \circ\!\!- \qquad 3-0\text{: A}}{0-2-\text{A}\backslash\text{C}^{-}}$$

(36)

database	$\overline{i-1\text{: B} \circ\!\!- i-0\text{: A}}^{3}$,	
	$\overline{j-2\text{: C} \circ\!\!- j-1\text{: B}}^{2}$,	
	$\overline{3-0\text{: A}_1}^{4}$	
agenda		
1.	$3-2\text{: C} \circ\!\!- 3-0\text{: A}$	DT
2.	$3-2\text{: C}$	RES $j=3$
3.	$3-1\text{: B}$	RES $i=3$
4.	$3-0\text{: A}$	RES

In this way associative unification is avoided; indeed the only matching is trivial unification between constants and variables.

7.1 Non-associativity via simultaneous groupoid and relational compilation

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, and exhaustive search would require all possibilities to be tried. 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 its own groupoid algebra and the associative 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

  compute the prosodic form induced

I.e. the end sequent 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. 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 vertical structure (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 $L \times V \times V$:

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

Unfolding is thus:

(38) $$\frac{\alpha+\gamma - i - k\colon B^p \qquad \circ\!\!- \qquad \alpha - i - j\colon A^{\overline{p}}}{\gamma - j - k\colon A\backslash B^p}\alpha, i \text{ new variables/constants as } p\ +/-$$

$$\frac{\gamma+\alpha - i - k\colon B^p \qquad \circ\!\!- \qquad \alpha - j - k\colon A^{\overline{p}}}{\gamma - i - j\colon B/A^p}\alpha, k \text{ new variables/constants as } p\ +/-$$

(39) $$\frac{\alpha - i - j\colon A^- \qquad \otimes \qquad \beta - j - k\colon B^-}{\alpha+\beta - i - k\colon A\bullet B^-}\alpha, \beta, j \text{ new variables}$$

By way of example consider the following:

(40)
$$\dfrac{\dfrac{\dfrac{\text{the references}}{\mathbf{r} - 0 - 1\colon \mathrm{N}} \qquad \dfrac{\text{are missing}}{\mathbf{m} - 1 - 2\colon ((\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S})/\mathrm{PP}} \qquad \dfrac{\text{from this book}}{\mathbf{f} - 2 - 3\colon \mathrm{PP}} \qquad \Rightarrow d - 0 - 3\colon \mathrm{S}}{\dfrac{b+(\mathbf{m}+a)\text{-}i\text{-}k_1\colon \mathrm{S}^+ \qquad \circ\!\!- \qquad \dfrac{b+\mathbf{k}\text{-}i\text{-}4\colon \mathrm{S}^- \qquad \circ\!\!- \qquad \dfrac{c+\mathbf{k}\text{-}l\text{-}4\colon \mathrm{S}^+ \qquad \circ\!\!- \qquad c\text{-}l\text{-}1\colon \mathrm{N}^-}{\mathbf{k}\text{-}1\text{-}4\colon \mathrm{N}\backslash\mathrm{S}^+}}{b\text{-}i\text{-}1\colon \mathrm{S}/(\mathrm{N}\backslash\mathrm{S})^-}}{\mathbf{m}+a\text{-}1\text{-}k_1\colon (\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S}^+} \qquad \circ\!\!- \qquad a\text{-}2\text{-}k_1\colon \mathrm{PP}^-}{\mathbf{m}\text{-}1\text{-}2\colon ((\mathrm{S}/(\mathrm{N}\backslash\mathrm{S}))\backslash\mathrm{S})/\mathrm{PP}^+}$$

(41) $$\triangleright\, b+(\mathbf{m}+a) - i - k_1\colon \mathrm{S} \circ\!\!- (b+\mathbf{k} - i - 4\colon \mathrm{S} \circ\!\!- (c+\mathbf{k} - l - 4\colon \mathrm{S} \circ\!\!- c - l - 1\colon \mathrm{N})) \otimes a - 2 - k_1\colon \mathrm{PP}$$

(42) database $\overline{\mathbf{r}\text{-}0\text{-}1\text{: N}}^{4}$,
$\overline{b{+}(\mathbf{m}{+}a)\text{-}i\text{-}k_1\text{: S} \circ\!\!- (b{+}\mathbf{k}\text{-}i\text{-}4\text{: S} \circ\!\!- (c{+}\mathbf{k}\text{-}l\text{-}4\text{: S} \circ\!\!- c\text{-}l\text{-}1\text{: N})) \otimes a\text{-}2\text{-}k_1\text{: PP}}^{1}$,
$\overline{c{+}\mathbf{k}\text{-}l\text{-}4\text{: S} \circ\!\!- c\text{-}l\text{-}1\text{: N}_2}^{3}$,
$\overline{\mathbf{f}\text{-}2\text{-}3\text{: PP}}^{5}$

agenda

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

Note how the term unification computing the hierarchical structure can be carried out one-way in the reverse order to the forward segment matchings.

(43) $d = b{+}(\mathbf{m}{+}a) = c{+}(\mathbf{m}{+}a) = \mathbf{r}{+}(\mathbf{m}{+}a) = \mathbf{r}{+}(\mathbf{m}{+}\mathbf{f})$

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

8 Multimodal Lambek calculi

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

(44)
$$\begin{array}{lcl} D(A\bullet_i B) & = & \{s_1{+}_i s_2 | s_1 \in D(A) \wedge s_2 \in D(B)\} \\ D(A\backslash_i B) & = & \{s | \forall s' \in D(A), s'{+}_i s \in D(B)\} \\ D(B/_i A) & = & \{s | \forall s' \in D(A), s{+}_i s' \in D(B)\} \end{array}$$

Multimodal groupoid compilation is immediate:

(45) a.
$$\frac{\alpha{+}_i\gamma\text{: } B^p \qquad \circ\!\!- \qquad \alpha\text{: } A^{\overline{p}}}{\gamma\text{: } A\backslash_i B^p}\,\alpha \text{ new variable/constant as } p\ +/-$$

b.
$$\frac{\gamma{+}_i\alpha\text{: } B^p \qquad \circ\!\!- \qquad \alpha\text{: } A^{\overline{p}}}{\gamma\text{: } B/_i A^p}\,\alpha \text{ new variable/constant as } p\ +/-$$

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. By way of example we shall deal with mulimodality for discontinuity which involves varying internal structural properties (associativity vs. non-associativity) as well as "split/wrap" interaction between modes. We shall also consider unary operators projecting bracketed string structure. In both cases simultaneous compilation including binary relational labelling provides additional advantages.

8.1 Discontinuity

Calculi of discontinuity for discontinuous functors, quantifier phrases, gapping, subject and object oriented reflexives etc. are developed in Solias (1992), Morrill and Solias (1993) and Morrill (1993), surmounting technical difficulties with Moortgat (1988, 1990, 1991). The treatment of Morrill (1993) is a pure multimodal residuation calculus with three families of connectives, $\{/, \backslash, \bullet\}, \{<, >, \diamond\}$ and $\{\uparrow, \downarrow, \odot\}$ each defined with respect to their adjunction in a total algebra: $+$ (associative, "surface"), $(\cdot, \cdot)$ (non-associative, "split"), and W (non-associative, "interpolate") with the structural interaction $s_1{+}s_2{+}s_3 = (s_1, s_3)Ws_2$. In Morrill (1994b) the product-free fragment is implemented using groupoid labelling in (first order) linear clauses. The unidirectionality of unification makes it managable through normalisation of and recursive decent through ground terms.

The following example illustrates with a quantifier in subject position. The higher type allows quantifier phrases to occur in situ while taking semantic scope at superordinate S nodes.

(46) $\mathbf{k}$: $(\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}$, $\mathbf{l}$: $\mathrm{N}\backslash\mathrm{S} \Rightarrow \mathbf{k}{+}\mathbf{l}$: S

(47)
$$\dfrac{\dfrac{}{aW\mathbf{k}:\mathrm{S} \quad \circ\!\!- \quad a:\mathrm{S}{\uparrow}\mathrm{N}^{-}}}{\mathbf{k}:(\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}^{+}} \qquad \dfrac{\dfrac{}{aW\mathbf{m}:\mathrm{S} \quad \circ\!\!- \quad \mathbf{m}:\mathrm{N}} \qquad \dfrac{}{b{+}\mathbf{l}:\mathrm{S} \quad \circ\!\!- \quad b:\mathrm{N}}}{\mathbf{l}:\mathrm{N}\backslash\mathrm{S}^{+}}$$

(48)

	database	$\overline{aW\mathbf{k}\text{: S} \circ\!\!- (aW\mathbf{m}\text{: S} \circ\!\!- \mathbf{m}\text{: N})}^{1}$,	
		$\overline{b{+}\mathbf{l}\text{: S} \circ\!\!- b\text{: N}}^{3}$,	
		$\overline{\mathbf{m}\text{: N}_2}^{4}$	
	agenda		
1.		$\mathbf{k}{+}\mathbf{l}$: S	RES $aW\mathbf{k} = \mathbf{k}{+}\mathbf{l} \Rightarrow a = (\epsilon, \mathbf{l})$
2.		$\mathbf{m}{+}\mathbf{l}$: S $\circ\!\!-$ $\mathbf{m}$: N	DT
3.		$\mathbf{m}{+}\mathbf{l}$: S	RES $b = \mathbf{m}$
4.		$\mathbf{m}$: N	RES

8.2 Sorted discontinuity

The calculus of Morrill (1993) just treated provides a natural space within which to work, but it includes abstractions which are not required, and while it is conceptually harmless, computationally such unwanted abstraction creates unwarrented complexity. Intuitively we want to work more concretely with strings and split strings rather than with a homogenous total algebra. Quite generally, when this situation arises it is normal to impose a discipline of typing or sorting on a formalism.

We implement a formulation of discontinuity which is a variant and refinement of that of Solias (1992) and Morrill and Solias (1993) with prosodic data types (or: sorts). This sorting restricts the class of types available, while still including all those of linguistic significance, and in so doing reduces the complexity of computation involved in what is otherwise associative and partially commutative unification.

Rather than deal with a total algebra $\langle L, +, (.,.), W, \epsilon \rangle$ we take a two-sorted algebra $\langle L, L \times L, +, (.,.), W, \epsilon \rangle$ with domains or sorts L (strings) and $L \times L$ (split strings) and operations, with the following functionalities, which obey the same association, identity and interpolation laws that they do in the unsorted algebra: $+ : L, L \to L$, $(.,.) : L, L \to L \times L, (s, s') = \langle s, s' \rangle$; $W : L \times L, L \to L, sWs' = \pi_1 s{+}s'{+}\pi_2 s$, $\epsilon \in L$. Interpretation of formulas is by residuation with respect to $+, (.,.)$ and W as before but they now come in two sorts: string ($\mathcal{F}$; prosodically interpreted as subsets of L) and split string ($\mathcal{G}$; prosodically interpreted as subsets of $L \times L$), and some previous formulas will be lost. Let us assume that atomic formulas $\mathcal{A}$ are of sort string. Then

the sorted formulas are defined by mutual recursion as follows:

$$(49)\quad \begin{array}{lcl} \mathcal{F} & ::= & \mathcal{A} \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}\bullet\mathcal{F} \mid \mathcal{G}<\mathcal{F} \mid \mathcal{F}>\mathcal{G} \mid \mathcal{G}\downarrow\mathcal{F} \mid \mathcal{G}\odot\mathcal{F} \\ \mathcal{G} & ::= & \mathcal{F}\uparrow\mathcal{F} \mid \mathcal{F}\diamond\mathcal{F} \end{array}$$

Groupoid interpretation of sorted discontinuity is thus:

$$(50)\quad \begin{array}{lcl} D(A\bullet B) & = & \{s_1{+}s_2 | s_1 \in D(A) \wedge s_2 \in D(B)\} \\ D(A\backslash B) & = & \{s | \forall s' \in D(A), s'{+}s \in D(B)\} \\ D(B/A) & = & \{s | \forall s' \in D(A), s{+}s' \in D(B)\} \\ & & \\ D(A\diamond B) & = & \{\langle s_1, s_2\rangle | s_1 \in D(A) \wedge s_2 \in D(B)\} \\ D(A>B) & = & \{s | \forall s' \in D(A), \langle s', s\rangle \in D(B)\} \\ D(B<A) & = & \{s | \forall s' \in D(A), \langle s, s'\rangle \in D(B)\} \\ & & \\ D(A\odot B) & = & \{s_1{+}s{+}s_2 | \langle s_1, s_2\rangle \in D(A) \wedge s \in D(B)\} \\ D(A\downarrow B) & = & \{s | \forall \langle s_1, s_2\rangle \in D(A), s_1{+}s{+}s_2 \in D(B)\} \\ D(B\uparrow A) & = & \{\langle s_1, s_2\rangle | \forall s \in D(A), s_1{+}s{+}s_2 \in D(B)\} \end{array}$$

The main advantage however depends on the possibility of a relational interpretation of sorted discontinuity. In this formulas of sort $\mathcal{F}$ (strings) are interpreted as subsets of $V \times V$; formulas of sort $\mathcal{G}$ (split strings) subsets of $V \times V \times V \times V$.

$$(51)\quad \begin{array}{lcl} D(A\backslash B) & = & \{\langle v_2, v_3\rangle | \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \\ D(B/A) & = & \{\langle v_1, v_2\rangle | \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \\ D(A\bullet B) & = & \{\langle v_1, v_3\rangle | \exists v_2, \langle v_1, v_2\rangle \in D(A) \;\&\; \langle v_2, v_3\rangle \in D(B)\} \\ & & \\ D(A>B) & = & \{\langle v_3, v_4\rangle | \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_2, v_3, v_4\rangle \in D(B)\} \\ D(B<A) & = & \{\langle v_1, v_2\rangle | \forall \langle v_3, v_4\rangle \in D(A), \langle v_1, v_2, v_3, v_4\rangle \in D(B)\} \\ D(A\diamond B) & = & \{\langle v_1, v_2, v_3, v_4\rangle | \langle v_1, v_2\rangle \in D(A) \;\&\; \langle v_3, v_4\rangle \in D(B)\} \\ & & \\ D(A\downarrow B) & = & \{\langle v_2, v_3\rangle | \forall \langle v_1, v_2, v_3, v_4\rangle \in D(A), \langle v_1, v_4\rangle \in D(B)\} \\ D(B\uparrow A) & = & \{\langle v_1, v_2, v_3, v_4\rangle | \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_4\rangle \in D(B)\} \\ D(A\odot B) & = & \{\langle v_1, v_4\rangle | \exists \langle v_2, v_3\rangle \in D(B) \;\&\; \langle v_1, v_2, v_3, v_4\rangle \in D(A)\} \end{array}$$

Neither scheme on its own is fully satisfactory as a basis for compilation. The groupoid labelling cannot express left or right products as before (succedent substring product is needed for gapping). But relational labelling alone encounters a new problem with succedent $\uparrow$:

(52) $$\frac{\langle i,l\rangle\text{: B}^- \quad \circ\!\!- \quad \langle j,k\rangle\text{: A}^+}{\langle i,j,k,l\rangle\text{: B}\uparrow\text{A}^-}??$$

The problem is that there is no place for Skolemisation in the term, and the hypothetical reasoning is therefore not properly controlled.

However matters falls into place through simultaneous groupoid and relational interpretation for discontinuity, wherein the groupoid term is skolemised. As before the computational problems are solved by labelling according to interpretation in the product of the two algebras. This allows (succedent) products and all the divisions. Formulas of sort $\mathcal{F}$ (strings) are interpreted as subsets of $L \times V \times V$; formulas of sort $\mathcal{G}$ (split strings) subsets of $L \times V \times V \times L \times V \times V$.

(53)
$$\begin{array}{lcl}
D(A\backslash B) & = & \{\langle s,v_2,v_3\rangle | \forall \langle s',v_1,v_2\rangle \in D(A), \langle s'{+}s,v_1,v_3\rangle \in D(B)\} \\
D(B/A) & = & \{\langle s,v_1,v_2\rangle | \forall \langle s',v_2,v_3\rangle \in D(A), \langle s{+}s',v_1,v_3\rangle \in D(B)\} \\
D(A{\bullet}B) & = & \{\langle s_1{+}s_2,v_1,v_3\rangle | \exists v_2, \langle s_1,v_1,v_2\rangle \in D(A) \ \&\ \langle s_2,v_2,v_3\rangle \in D(B)\} \\
 & & \\
D(A{>}B) & = & \{\langle s,v_3,v_4\rangle | \forall \langle s',v_1,v_2\rangle \in D(A), \langle s',v_1,v_2,s,v_3,v_4\rangle \in D(B)\} \\
D(B{<}A) & = & \{\langle s,v_1,v_2\rangle | \forall \langle s',v_3,v_4\rangle \in D(A), \langle s,v_1,v_2,s',v_3,v_4\rangle \in D(B)\} \\
D(A{\circ}B) & = & \{\langle s_1,v_1,v_2,s_2,v_3,v_4\rangle | \langle s_1,v_1,v_2\rangle \in D(A) \ \&\ \langle s_2,v_3,v_4\rangle \in D(B)\} \\
 & & \\
D(A{\downarrow}B) & = & \{\langle s,v_2,v_3\rangle | \forall \langle s_1,v_1,v_2,s_2,v_3,v_4\rangle \in D(A), \langle s_1{+}s{+}s_2,v_1,v_4\rangle \in D(B)\} \\
D(B{\uparrow}A) & = & \{\langle s_1,v_1,v_2,s_2,v_3,v_4\rangle | \forall \langle s,v_2,v_3\rangle \in D(A), \langle s_1{+}s{+}s_2,v_1,v_4\rangle \in D(B)\} \\
D(A{\odot}B) & = & \{\langle s_1{+}s{+}s_2,v_1,v_4\rangle | \exists \langle s,v_2,v_3\rangle \in D(B) \ \&\ \langle s_1,v_1,v_2,s_2,v_3,v_4\rangle \in D(A)\}
\end{array}$$

For the sake of completeness we spell out all the unfoldings in all their detail.

(54)
$$\frac{\alpha{+}\gamma - i - k\text{: } B^p \quad \circ\!\!- \quad \alpha - i - j\text{: } A^{\overline{p}}}{\gamma - j - k\text{: } A\backslash B^p}\alpha, i \text{ new variables/constants as } p +/-$$

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

(55)
$$\frac{\alpha - i - j\text{: } A^- \quad \otimes \quad \beta - j - k\text{: } B^-}{\alpha{+}\beta - i - k\text{: } A{\bullet}B^-}\alpha, \beta, j \text{ new variables}$$

(56) $$\frac{\alpha - i - j, \gamma - k - l\colon B^{p} \quad \circ\!\!- \quad \alpha - i - j\colon A^{\overline{p}}}{\gamma - k - l\colon A{>}B^{p}}\alpha, i, j \text{ new variables/constants as } p\ +/-$$

$$\frac{\gamma - i - j, \alpha - k - l\colon B^{p} \quad \circ\!\!- \quad \alpha - k - l\colon A^{\overline{p}}}{\gamma - i - j\colon B{<}A^{p}}\alpha, k, l \text{ new variables/constants as } p\ +/-$$

(57) $$\frac{\alpha - i - j\colon A^{-} \quad \otimes \quad \beta - k - l\colon B^{-}}{\alpha - i - j, \beta - k - l\colon A{\diamond}B^{-}}\alpha, \beta \text{ new variables}$$

(58) $$\frac{\alpha{+}\gamma{+}\alpha' - i - k\colon B^{p} \quad \circ\!\!- \quad \alpha - i - j, \alpha' - k - l\colon A^{\overline{p}}}{\gamma - j - k\colon A{\downarrow}B^{p}}\alpha, \alpha', i, l \text{ new variables/constants as } p\ +/-$$

$$\frac{\gamma{+}\alpha{+}\gamma' - i - l\colon B^{p} \quad \circ\!\!- \quad \alpha - j - k\colon A^{\overline{p}}}{\gamma - i - j, \gamma' - k - l\colon B{\uparrow}A^{p}}\alpha \text{ new variable/constant as } p\ +/-$$

(59) $$\frac{\alpha - j - k\colon A^{-} \quad \otimes \quad \beta - i - j, \beta' - k - l\colon B^{-}}{\beta{+}\alpha{+}\beta' - i - l\colon A{\odot}B^{-}}\alpha, \beta, \beta', j, k \text{ new variables}$$

The previous subject quantifier example is now as follows.

(60) $\mathbf{k} - 0 - 1\colon (\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}, \mathbf{l} - 1 - 2\colon \mathrm{N}\backslash\mathrm{S} \Rightarrow \mathbf{k}{+}\mathbf{l} - 0 - 2\colon \mathrm{S}$

(61) $$\frac{a{+}\mathbf{k}{+}a' - i - j\colon \mathrm{S} \quad \circ\!\!- \quad \dfrac{a{+}\mathbf{m}{+}a' - i - j\colon \mathrm{S} \quad \circ\!\!- \quad \mathbf{m} - 0 - 1\colon \mathrm{N}}{a - i - 0, a' - 1 - j\colon \mathrm{S}{\uparrow}\mathrm{N}^{-}}}{\mathbf{k} - 0 - 1\colon (\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}^{+}}$$

$$\frac{b{+}\mathbf{l} - q - 2\text{: S} \qquad \circ\!\!- \qquad b - q - 1\text{: N}}{\mathbf{l} - 1 - 2\text{: N}\backslash\text{S}^{+}}$$

(62)

database	$\overline{a{+}\mathbf{k}{+}a' - i - j\text{: S} \circ\!\!- (a{+}\mathbf{m}{+}a' - i - j\text{: S} \circ\!\!- \mathbf{m} - 0 - 1\text{: N})}^{1}$, $\overline{b{+}\mathbf{l} - q - 2\text{: S} \circ\!\!- b - q - 1\text{: N}}^{3}$, $\overline{\mathbf{m} - 0 - 1\text{: N}_2}^{4}$		
agenda			
1.	$c - 0 - 2$: S	RES $i = 0, j = 2, c = a{+}\mathbf{k}{+}a' \Rightarrow c = \mathbf{k}{+}\mathbf{l}$	
2.	$a{+}\mathbf{m}{+}a' - 0 - 2\text{: S} \circ\!\!- \mathbf{m} - 0 - 1$: N	DT	
3.	$a{+}\mathbf{m}{+}a' - 0 - 2$: S	RES $q = 0, b{+}\mathbf{l} = a{+}\mathbf{m}{+}a' \Rightarrow a = \epsilon, a' = \mathbf{l}$	
4.	$b - 0 - 1$: N	RES $b = \mathbf{m}$	

9 Bracket Operators

As final illustration of the present methods generality we consider unary residuation operators which can project structure blocking associativity and thereby effect island constraints such as the sentential subject constraint and coordinate structure constraint.

The inability of associative calculus to represent hierarchical structure, and its inability to express constraints with respect to such structure is addressed in Morrill (1992, 1994a, c) (see also Moortgat 1994) by addition of "bracket" operators. In addition to the binary operators, types are generated by unary (prefix) operators $[\,]$ ("bracket") and $[\,]^{-1}$ ("antibracket"). Functional models are obtained by interpretation in $\langle L, +, [\,.\,]\rangle$ where $[\,.\,]$ is a unary operation; for symmetric proof theory we require that this is a 1-1 function (permutation) so that its inverse $[\,.\,]^{-1}$ is total.

(63) $[\,[\,s\,]^{-1}\,] = [\,[\,s\,]\,]^{-1} = s$

(64)
$$\begin{array}{lcl} D([\,]A) & = & \{[\,s\,] | s \in D(A)\} \\ D([\,]^{-1}A) & = & \{s | [\,s\,] \in D(A)\} \quad = \quad \{[\,s\,]^{-1} | s \in D(A)\} \end{array}$$

Intuitively $[\,]A$ is the result of appointing or crystalising prosodic objects as domains or constituents, and $[\,]^{-1}A$ the result of annulling or dissolving

appointment as a domain. Groupoid and groupoid plus interval unfolding is thus:

(65) $$\frac{[\,\alpha\,]^{-1}\colon A^p}{\alpha\colon [\,]A^p} \qquad \frac{[\,\alpha\,]\colon A^p}{\alpha\colon [\,]^{-1}A^p}$$

(66) $$\frac{[\,\alpha\,]^{-1} - i - j\colon A^p}{\alpha - i - j\colon [\,]A^p} \qquad \frac{[\,\alpha\,] - i - j\colon A^p}{\alpha - i - j\colon [\,]^{-1}A^p}$$

The advantages of including segments are the same as those for **NL**: horizontal structure is factored out and corresponds to parsing input; dominance structure is computed. The following example shows how vertical structure is projected over a sentencial subject by assignment of a sentential complement verb phrase to $[\,]\mathrm{SP}\backslash\mathrm{S}$, and over a coordinate structure by assignment of a coordinator to $(\mathrm{S}\backslash[\,]^{-1}\mathrm{S})/\mathrm{S}$.

(67) $\mathbf{m} - 0 - 1\colon \mathrm{SP},\ \mathbf{k} - 1 - 2\colon [\,]\mathrm{SP}\backslash\mathrm{S},\ \mathbf{l} - 2 - 3\colon \mathrm{S}\backslash[\,]^{-1}\mathrm{S} \Rightarrow$
$[\,[\,\mathbf{m}\,]{+}\mathbf{k}{+}\mathbf{l}\,]\colon \mathrm{S}$

$$\dfrac{a{+}\mathbf{k} - i - 2\colon \mathrm{S} \quad \circ\!\!- \quad \dfrac{[\,a\,]^{-1} - \mathrm{i} - 1\colon \mathrm{SP}}{a - i - 1\colon [\,]\mathrm{SP}^-}}{\mathbf{k} - 1 - 2\colon [\,]\mathrm{SP}\backslash\mathrm{S}^+} \qquad \dfrac{\dfrac{[\,b{+}\mathbf{l}\,] - j - 3\colon \mathrm{S}}{b{+}\mathbf{l} - j - 3\colon [\,]^{-1}\mathrm{S}^+} \quad \circ\!\!- \quad b - j - 2\colon \mathrm{S}}{\mathbf{l} - 2 - 3\colon \mathrm{S}\backslash[\,]^{-1}\mathrm{S}}$$

(68)

	database	$\overline{\mathbf{m} - 0 - 1\colon \mathrm{SP}}^{3}$,	
		$\overline{a{+}\mathbf{k} - i - 2\colon \mathrm{S} \circ\!\!- [\,a\,]^{-1} - i - 1\colon \mathrm{SP}}^{2}$,	
		$\overline{[\,b{+}\mathbf{l}\,] - j - 3\colon \mathrm{S} \circ\!\!- b - j - 2\colon \mathrm{S}}^{1}$	
	agenda		
1.		$c - 0 - 3\colon \mathrm{S}$	RES $j = 0, c = [\,b{+}\mathbf{l}\,]$ $\Rightarrow\ c = [\,[\,\mathbf{m}\,]{+}\mathbf{k}{+}\mathbf{l}\,]$
2.		$b - 0 - 2\colon \mathrm{S}$	RES $i = 0, b = a{+}\mathbf{k}$ $\Rightarrow\ b = [\,\mathbf{m}\,]{+}\mathbf{k}$
3.		$[\,a\,]^{-1} - 0 - 1\colon \mathrm{SP}$	RES $[\,a\,]^{-1} = \mathbf{m}$ $\Rightarrow\ a = [\,\mathbf{m}\,]$

10 Summary

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 and 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.

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, 'Normal form theorem proving for the Lambek calculus', in H. Karlgren (ed.), *Proceedings of COLING 1990*, Stockholm.

Hepple, Mark: 1993, 'Labelled deduction and discontinuous constituency', ms. University of Pennsylvania.

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: 1990, 'The Quantification Calculus: Questions of Axiomatisation', in Deliverable R1.2.A of DYANA Dynamic Interpretation of Natural Language, ESPRIT Basic Research Action BR3175.

Moortgat, Michael: 1991, 'Generalised Quantification and Discontinuous type constructors', to appear in Sijtsma and Van Horck (eds.) *Proceedings Tilburg Symposium on Discontinuous Constituency*, Walter de Gruyter, Berlin.

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: 1994, 'Residuation in mixed Lambek systems', ms. OTS, Rijksuniversiteit Utrecht.

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: 1992, 'Categorial Formalisation of Relativisation: Pied Piping, Islands, and Extraction Sites', Report de Recerca LSI–92–23–R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

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.

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

Morrill, Glyn: 1994b, 'Clausal Proof Nets and Discontinuity', Report de Recerca LSI–94–21–R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, to appear in .

Morrill, Glyn: 1994c, 'Structural Facilitation and Structural Inhibition', Report de Recerca LSI–94–26–R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

Morrill, Glyn and Teresa Solias: 1993, 'Tuples, Discontinuity and Gapping', in *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht, 287–297.

Nadathur, D, and D. Miller: 1990, 'Higher-order Horn Clauses, *Journal of the ACM***37(4)**, 777–814.

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.

Solias, Teresa: 1992, *Gramáticas Categoriales, Coordinación Generalizada y Elisión*, Ph.D. dissertation, Universidad Autónoma de Madrid.

Program Listing

```
/*
Operators for lexical assignments Alpha - Phi := A
and assignments Alpha - Phi: A
*/

:- op(500, xfx, :).
:- op(500, xfx, :=).

:- op(450, xfy, -).

/*
Operators over, under, product, to, from, wrap, infix,
discontinuous product, and antibracket
*/

:- op(400, xfx, /).
:- op(400, xfx, \).

:- op(400, xfx, >).
:- op(400, xfx, <).

:- op(400, xfx, wr).
:- op(400, xfx, in).

:- op(400, xfx, pr).

:- op(400, xfx, dp).

:- op(300, fx, #).

% Surface adjunction
```

```
:- op(400, xfy, +).

% Lexical assignments

about   -  about
        := pp/n.
and     -  [lmd, X, [lmd, Y, [and, Y, X]]]
        := (s\ #s)/s.
and     -  [lmd, X, [lmd, Y, [and, [app, [fst, Y], [snd, Y]],
                                   [app, [app, [snd, Y], [snd, X]], [fst, X]]]]]
        := (((s wr ((n\s)/n)) dp ((n\s)/n))\s)/(n pr n).
believes-  believe
        := (n\s)/s.
bill     -  b
        := n.
book    -  book
        := cn.
dog     -  dog
        := cn.
for     -  [lmd, X, X]
        := pp/n.
(gives, the+cold+shoulder)
        -  shun
        := (n\s)wr n.
(either, or)
        -  [lmd, X, [lmd, Y, [or, X, Y]]]
       := (s/s)wr s.
everyone-  [lmd, X, [all, Y, [app, X, Y]]]
       := (s wr n)in s.
herself -  [lmd, U, [lmd,X,[lmd,Y,[app,[app,X,Y],[app, U, Y]]]]]
       := (X wr n)in((((n\s)/X)/n)>((n\s)wr n))
               :- X=n; X=pp. % obj. antec. n and pp pied-piping
himself -  [lmd,X,[lmd,Y,[app,[app,X,Y],Y]]]
        := ((n\s)wr n)in(n\s). % Sbj. antec.
him     -  [lmd, Y, [lmd, X, [app, [app, [fst, X], [snd, X]],
                                   [app, Y, [snd, X]]]]]
        := (s wr n) in ((((s/s) wr n) dp n)\s).
itself  -  [lmd, X, [app, [app, [fst, X], [snd, X]], [snd, X]]]
        := (((s/n) wr n) dp n)\s.
kicks+the+bucket
        - die
        := n\s.
john    -  j
```

```
        := n.
likes   -  like
        := (n\s)/n.
man     -  man
        := cn.
mary    -  m
        := n.
(neither, nor)
        -  [lmd, X, [lmd, Y, [lmd, Z, [not, [or, [app, X, Z], [app, Y, Z]]]]]]
        := ((n\s)/(n\s))wr (n\s).
of      -  of
        := (cn\cn)/n.
or      -  [lmd, X, [lmd, Y, [lmd, Z, [or, [app, Z, Y], [app, Z, X]]]]]
        := (n\((s wr n)in s))/n. % "wide-scope 'or'" assignment
picture -  picture
        := cn.
(rings, up)
        -  phone
        := (n\s)wr n.
seeks   -  seek
        := (n\s)/(((n\s)/n)\(n\s)).
sings   -  sing
        := n\s.
shows   -  show
        := ((n\s)/n)/n.
shows   -  [lmd, X, [lmd, Y, [[app, show, Y], X]]]
        := ((n\s)/pp)/n.
some    -  [lmd,Z,[lmd,X,[xst,Y,[and,[app,Z,Y],[app,X,Y]]]]]
        := ((s wr n)in s)/cn.
someone -  [lmd,X,[xst,Y,[app,X,Y]]]
        := (s wr n)in s.
talks   -  talk
        := ((n\s)/pp)/pp.
that    -  [lmd, X, [lmd, Y, [lmd, Z, [app, [app, and, [app, Y, Z]],
                [app, X, Z]]]]]
        := #(cn\cn)/(s/n). % non-pied-piping relative pronoun
the     -  [lmd, X, [iota, Y, [app, X, Y]]]
        := n/cn.
thinks  -  think
        := (n\s)/s.
to      -  to
        := pp/n.
votes   -  vote
        := (n\s)/pp.
```

```
walks   -  walk
        := n\s.
whom    -  [lmd, X, [lmd, Y, [lmd, Z, [lmd, W, [and, [app, Z, W],
                [app, Y,[app, X, W]]]]]]] % pied-piping assignment
        := (n wr n) in (#(cn\cn)/(s/n)).
whose   -  [lmd, U, [lmd, X, [lmd, Y, [lmd, Z, [lmd, W, [and, [app, Z, W],
                [app, Y,[app, X, [iota, V, [and, [app, U, V],
                [app, poss, W, V]]]]]]]]]]]
        := ((n wr n) in (#(cn\cn)/(s/n)))/cn. % pied-piping assignment
woman   -  woman
        := cn.

% reset resets the gensymb record to 0

reset :-
        clear,
        assert(rec(0)), !.

% clear removes any gensymb records

clear :-
        retract(rec(_)),
        clear, !.

clear :- !.

% gensymb(-N1) generates a new symbol (integer) N1

gensymb(N) :-
        retract(rec(N)),
        N1 is N+1,
        assert(rec(N1)), !.

unfposlist([], []).

unfposlist([X|Xs], [Y|Ys]) :-
        unfpos([X], Y),
        unfposlist(Xs, Ys).

unfpos([S: A|Gs], [S: A|G1s]) :-
        atom(A),
        unfneglist(Gs, G1s).

unfpos([(I-J)-Gamma-Chi: B/A|Gs], Pcls) :-
```

```
        unfpos([(I-K)-Gamma+Alpha-[app, Chi, Phi]: B,
                (J-K)-Alpha-Phi: A|Gs], Pcls).

unfpos([(J-K)-Gamma-Chi: A\B|Gs], Pcls) :-
        unfpos([(I-K)-Alpha+Gamma-[app, Chi, Phi]: B,
                (I-J)-Alpha-Phi: A|Gs], Pcls).

unfpos([(I-J)-Gamma-Chi: B<A|Gs], Pcls) :-
        unfpos([(I-J, K-L)-(Gamma, Alpha)-[app, Chi, Phi]: B,
                (K-L)-Alpha-Phi: A|Gs], Pcls).

unfpos([(K-L)-Gamma-Chi: A>B|Gs], Pcls) :-
        unfpos([(I-J, K-L)-(Alpha, Gamma)-[app, Chi, Phi]: B,
                (I-J)-Alpha-Phi: A|Gs], Pcls).

unfpos([(I-J, K-L)-(Gamma1, Gamma2)-Chi: B wr A|Gs], Pcls) :-
        unfpos([(I-L)-Gamma1+Alpha+Gamma2-[app, Chi, Phi]: B,
                (J-K)-Alpha-Phi: A|Gs], Pcls).

unfpos([(J-K)-Gamma-Chi: A in B|Gs], Pcls) :-
        unfpos([(I-L)-Alpha1+Gamma+Alpha2-[app, Chi, Phi]: B,
                 (I-J, K-L)-(Alpha1, Alpha2)-Phi: A|Gs], Pcls).

unfpos([(I-J)-Alpha-Phi: #A| Gs], Pcls) :-
        unfpos([(I-J)-b(Alpha)-Phi: A|Gs], Pcls).

unfneglist([], []).

unfneglist([X|Xs], ZsWs) :-
        unfneg(X, Zs),
        unfneglist(Xs, Ws),
        append(Zs, Ws, ZsWs).

unfneg(S: A, [S: A]) :-
        atom(A).

unfneg((I-J)-Gamma-[lmd, X, Psi]: B/A, [[Goal|Pcls]]) :-
        gensymb(K), gensymb(Alpha),
        unfneg((I-K)-Gamma+Alpha-Psi: B, Goal),
        unfpos([(J-K)-Alpha-X: A], Pcls).

unfneg((J-K)-Gamma-[lmd, X, Psi]: A\B, [[Goal|Pcls]]) :-
        gensymb(I), gensymb(Alpha),
        unfneg((I-K)-Alpha+Gamma-Psi: B, Goal),
```

```
        unfpos([(I-J)-Alpha-X: A], Pcls).

unfneg((I-K)-Alpha+Beta-[pair, Phi, Psi]: A pr B, XsYs) :-
        unfneg((I-J)-Alpha-Phi: A, Xs),
        unfneg((J-K)-Beta-Psi: B, Ys),
        append(Xs, Ys, XsYs).

unfneg((I-J)-Gamma-[lmd, X, Psi]: B<A, [[Goal|Pcls]]) :-
        gensymb(K), gensymb(L), gensymb(Alpha),
        unfneg((I-J, K-L)-(Gamma, Alpha)-Psi: B, Goal),
        unfpos([(K-L)-Alpha-X: A], Pcls).

unfneg((K-L)-Gamma-[lmd, X, Psi]: A>B, [[Goal|Pcls]]) :-
        gensymb(I), gensymb(J), gensymb(Alpha),
        unfneg((I-J, K-L)-(Alpha, Gamma)-Psi: B, Goal),
        unfpos([(I-J)-Alpha-X: A], Pcls).

unfneg((I-J, K-L)-(Gamma1, Gamma2)-[lmd, X, Psi]: B wr A, [[Goal|Pcls]]) :-
        gensymb(Alpha),
        unfneg((I-L)-Gamma1+Alpha+Gamma2-Psi: B, Goal),
        unfpos([(J-K)-Alpha-X: A], Pcls).

unfneg((J-K)-Gamma-[lmd, X, Psi]: A in B, [[Goal|Pcls]]) :-
        gensymb(I), gensymb(L),
        gensymb(Alpha1), gensymb(Alpha2),
        unfneg((I-L)-Alpha1+Gamma+Alpha2-Psi: B, Goal),
        unfpos([(I-J, K-L)-(Alpha1, Alpha2)-X: A], Pcls).

unfneg((I-L)-Alpha1+Beta+Alpha2-[pair, Phi, Psi]: A dp B, YsXs) :-
        unfneg((I-J, K-L)-(Alpha1, Alpha2)-Phi: A, Xs),
        unfneg((J-K)-Beta-Psi: B, Ys),
        append(Ys, Xs, YsXs).

prove([], []).

prove(Database, [[X|Pcls]|Goals]) :- append(X, Goals, XGoals),
        prove([Pcls|Database], XGoals), !.

prove(DB1XGsDB2, [(I-J)-Alpha-Phi: A|Goals]) :-
        append(DB1, [[(I-J)-Alpha1-Phi: A|Gs]|DB2], DB1XGsDB2),
        append(DB1, DB2, DB1DB2),
        append(Gs, Goals, GsGoals),
        prove(DB1DB2, GsGoals),
        eq1(Alpha1, Alpha).
```

```
/*
eq(Alpha, Alpha1) means that the normal form ground prosodic term
Alpha is unifiable with the prosodic term Alpha1, which on exit
is itself grounded accordingly
*/

eq(Alpha, V) :-
        var(V), !, V = Alpha.

eq(Alpha, X) :-
        integer(X), !, Alpha = X.

eq(b(Alpha), b(Beta)) :-
        eq(Alpha, Beta).

eq(Alpha, Beta+0) :-
        eq(Alpha, Beta).

eq(Alpha, 0+Beta) :-
        eq(Alpha, Beta).

eq(AlphaBeta, Gamma+Delta) :-
        eq2(AlphaBeta, Alpha, Beta),
        eq(Alpha, Gamma),
        eq(Beta, Delta).

/*
eq1(Alpha, Alpha1) means that the ground prosodic term Alpha
is unifiable with the prosodic term Alpha1, which on exit is
itself grounded accordingly
*/

eq1(Alpha, Alpha1) :-
        pnorm(Alpha, AlphaN),
        eq(AlphaN, Alpha1).

/*
eq2(+AlphaBeta, -Alpha, -Beta) means that (ground, normal form)
prosodic term AlphaBeta is equal to the result of associative
surface adjunction of Alpha and Beta
*/

eq2(Alpha+Beta, Alpha, Beta).
```

```
eq2(Alpha+Beta, Alpha1, Alpha2+Beta) :-
        eq2(Alpha, Alpha1, Alpha2).

eq2(Alpha+Beta, Alpha+Beta1, Beta2) :-
        eq2(Beta, Beta1, Beta2).

/*
pnorm(+Alpha, -Gamma) means that Gamma is the result of normalising
the ground prosodic term Alpha
*/

pnorm(Alpha, Gamma) :-
        pcontract(Alpha, Beta), !,
        pnorm(Beta, Gamma).

pnorm(Alpha, Alpha).

/*
pcontract(Alpha, Gamma) means that Gamma is the result of performing
one contraction step on the ground prosodic term Alpha
*/

pcontract(Alpha+0, Alpha).

pcontract(0+Alpha, Alpha).

pcontract(Alpha+Beta, AlphaN+Beta) :-
        pcontract(Alpha, AlphaN).

pcontract(Alpha+Beta, Alpha+BetaN) :-
        pcontract(Beta, BetaN).

pcontract(b(Alpha), b(AlphaN)) :-
        pcontract(Alpha, AlphaN).

show(An, Su) :-
        unfposlist(An, An1),
        unfneg(Su, Su1),
        prove(An1, Su1).

lookup([], [], N, N).

lookup([W|L], [(I-J)-W-Phi: C|Cs], I, K) :-
```

```
        W - Phi := C,
        lookup(L, Cs, J, K),
        gensymb(I).

lookup([W|L], [(I-J)-(W+Alpha)-Phi: C|Cs], I, K) :-
        (W+Alpha) - Phi := C,
        checkoff(L, Alpha, L1),
        lookup(L1, Cs, J, K),
        gensymb(I).

lookup(L, [(I-J, K-M)-(Alpha, Beta)-Phi: C|C1sC2s], I, N) :-
        (Alpha, Beta)-Phi := C,
        checkoff(L, Alpha, L1L2),
        append(L1, L2, L1L2),
        checkoff(L2, Beta, L3),
        lookup(L1, C1s, J, K),
        lookup(L3, C2s, M, N),
        gensymb(I), gensymb(K),
        append(C1s, C2s, C1sC2s).

checkoff([W|L], W, L).

checkoff([W|L], W+Alpha, L1) :-
        checkoff(L, Alpha, L1).

t(N) :-
        reset,
        str(N, L, C),
        nl, nl, write(N), write('. '), write(L: C),
        gensymb(K),
        lookup(L, Cs, I, K),
        show(Cs, (I-K)-Alpha-Phi: C),
        nl, nl, write(Alpha), write(' - '), nl,
        eval(Phi, NF), write(NF), fail.

/*
eval(+Phi, -NF) means that NF is the result of normalising the
semantic form Phi
*/

eval(Phi, NF) :-
        numbervars(Phi, 0, _),
        eval1(Phi, NF).
```

```
/*
eval1(+Phi, -NF) means that NF is the result of normalising
the frozen semantic form Phi
*/

eval1(Phi, NF) :-
        contract(Phi, Phi1), !,
        eval1(Phi1, NF).

eval1(Phi, Phi).

/*
contract(+Phi, -Phi1) means that Phi1 is the result of applying one
contraction step to the frozen semantic form Phi
*/

contract([app, [lmd, X, Phi], Psi], Chi) :-
        subst(Psi, X, Phi, Chi).

contract([fst, [pair, X, _]], X).
contract([snd, [pair, _, Y]], Y).

contract([H|T], [H|T1]) :-
        contractlist(T, T1).

contractlist([Phi|Phis], [Phi1|Phis]) :-
        contract(Phi, Phi1).

contractlist([Phi|Phis], [Phi|Phis1]) :-
        contractlist(Phis, Phis1).

/*
subst(+Phi, +X, +Psi, -NPsi) means that NPsi is the result of replacing
by Phi all Xs in the frozen semantic form Psi
*/

subst(Phi, X, X, Phi).

subst(_, _, C, C) :-
        atom(C).

subst(_, _, X, X) :-
        X = '$VAR'(_).
```

```
subst(Phi, X, [H|T], [H1|T1]) :-
subst(Phi, X, H, H1),
subst(Phi, X, T, T1).

% Test strings

% Simple sentences

str(1, [john, walks], s).
str(2, [john, likes, mary], s).
str(3, [john, seeks, mary], s).
str(4, [mary, shows, the, woman, the, book], s).

str(a4, [john, kicks, the, bucket], s).

% Discontinous functors

str(5, [john, rings, mary, up], s).
str(6, [john, gives, mary, the, cold, shoulder], s).
str(7, [either, john, walks, or, mary, sings], s).
str(8, [john, neither, walks, nor, sings], s).

% Relativisation

str(9, [the, man, that, mary, likes, walks], s).
str(10, [the, man, that, john, thinks, mary, likes, walks], s).
str(11, [the, man, that, john, thinks, bill, believes, mary, likes, walks], s).

% Coordinate Structure Constraint

str(12, [john, walks, and, mary, sings], s).
str(13, [that, john, walks, and, mary, likes], cn\cn).

str(a13, [that, john, thinks, bill, likes, the, man, that, shows], cn\cn).

% Partee/Rooth "wide-scope 'or'"

str(14, [john, thinks, bill, or, mary, walks], s).

% Subject-antecedent reflexivisation

str(15, [john, likes, himself], s).
str(16, [john, votes, for, himself], s).
str(17, [john, shows, himself, the, book], s).
```

```
% Object antecedent reflexivisation

str(18, [john, shows, mary, herself], s).
str(19, [john, shows, herself, mary], s).

% Pied-Piping object antecedent reflexivisation

str(20, [john, shows, mary, the, picture, of, herself], s).

% Non-c-command pied-piping object antecedent reflexivisation

str(21, [john, talks, to, mary, about, herself], s).

% Quantification

str(22, [someone, walks], s).
str(23, [some, man, walks], s).
str(24, [everyone, likes, someone], s).
str(25, [john, seeks, someone], s).
str(26, [everyone, seeks, someone], s).
str(27, [bill, thinks, someone, walks], s).
str(28, [bill, thinks, some, man, shows, everyone, john], s).
str(29, [the, book, that, john, shows, everyone], n).

% Pied-Piping

str(30, [the, man, whom, john, likes, the, picture, of], n).
str(31, [the, man, the, picture, of, whom, john, likes], n).
str(32, [the, man, whose, book, john, likes, the, picture, of], n).
str(33, [the, man, the, picture, of, whose, book, john, likes], n).

str(34, [john, likes, mary, and, mary, bill], s).
str(35, [mary, shows, the, dog, itself], s).
str(36, [mary, shows, john, to, himself], s).
str(37, [mary, talks, to, john, about, himself], s).

str(38, [john, thinks, bill, likes, him], s).
str(39, [john, likes, him], s).
str(40, [john, seeks, him], s).
```

Log

```
?- t(_).


1. [john,walks]:s

john+walks -
[app,walk,j]

2. [john,likes,mary]:s

john+likes+mary -
[app,[app,like,m],j]

3. [john,seeks,mary]:s

john+seeks+mary -
[app,[app,seek,[lmd,$VAR(1),[lmd,$VAR(0),[app,[app,$VAR(1),m],$VAR(0)]]]],j]

4. [mary,shows,the,woman,the,book]:s

mary+ (shows+the+woman)+the+book -
[app,[app,[app,show,[iota,$VAR(2),[app,woman,$VAR(2)]]],[iota,$VAR(0),
[app,book,$VAR(0)]]],m]

a4. [john,kicks,the,bucket]:s

john+kicks+the+bucket -
[app,die,j]

5. [john,rings,mary,up]:s

john+rings+mary+up -
[app,[app,phone,m],j]

6. [john,gives,mary,the,cold,shoulder]:s

john+gives+mary+the+cold+shoulder -
[app,[app,shun,m],j]

7. [either,john,walks,or,mary,sings]:s
```

```
 (either+ (john+walks)+or)+mary+sings -
[or,[app,walk,j],[app,sing,m]]

8. [john,neither,walks,nor,sings]:s

john+ (neither+walks+nor)+sings -
[not,[or,[app,walk,j],[app,sing,j]]]

9. [the,man,that,mary,likes,walks]:s

 (the+man+b(that+mary+likes))+walks -
[app,walk,[iota,$VAR(4),[app,[app,and,[app,man,$VAR(4)]],[app,[app,
like,$VAR(4)],m]]]]

10. [the,man,that,john,thinks,mary,likes,walks]:s

 (the+man+b(that+john+thinks+mary+likes))+walks -
[app,walk,[iota,$VAR(4),[app,[app,and,[app,man,$VAR(4)]],[app,[app,
think,[app,[app,like,$VAR(4)],m]],j]]]]

11. [the,man,that,john,thinks,bill,believes,mary,likes,walks]:s

 (the+man+b(that+john+thinks+bill+believes+mary+likes))+walks -
[app,walk,[iota,$VAR(4),[app,[app,and,[app,man,$VAR(4)]],[app,[app,
think,[app,[app,believe,[app,[app,like,$VAR(4)],m]],b]],j]]]]

12. [john,walks,and,mary,sings]:s

b( (john+walks)+and+mary+sings) -
[and,[app,walk,j],[app,sing,m]]

13. [that,john,walks,and,mary,likes]:cn\cn

a13. [that,john,thinks,bill,likes,the,man,that,shows]:cn\cn

14. [john,thinks,bill,or,mary,walks]:s

john+thinks+ (bill+or+mary)+walks -
[app,[app,think,[or,[app,walk,b],[app,walk,m]]],j]

 (john+thinks)+ (bill+or+mary)+walks -
[or,[app,[app,think,[app,walk,b]],j],[app,[app,think,[app,walk,m]],j]]

15. [john,likes,himself]:s
```

```
john+likes+himself -
[app,[app,like,j],j]

16. [john,votes,for,himself]:s

john+ (votes+for)+himself -
[app,[app,vote,j],j]

17. [john,shows,himself,the,book]:s

john+shows+himself+the+book -
[app,[app,[app,show,j],[iota,$VAR(1),[app,book,$VAR(1)]]],j]

18. [john,shows,mary,herself]:s

john+shows+mary+herself -
[app,[app,[app,show,m],m],j]

19. [john,shows,herself,mary]:s

20. [john,shows,mary,the,picture,of,herself]:s

john+shows+mary+ (the+picture+of)+herself -
[app,[app,[app,show,m],[iota,$VAR(4),[app,[app,[app,of,m],picture],
$VAR(4)]]],j]

21. [john,talks,to,mary,about,herself]:s

john+ (talks+to)+mary+about+herself -
[app,[app,[app,talk,[app,to,m]],[app,about,m]],j]

22. [someone,walks]:s

someone+walks -
[xst,$VAR(1),[app,walk,$VAR(1)]]

23. [some,man,walks]:s

 (some+man)+walks -
[xst,$VAR(1),[and,[app,man,$VAR(1)],[app,walk,$VAR(1)]]]

24. [everyone,likes,someone]:s
```

```
everyone+likes+someone -
[all,$VAR(4),[xst,$VAR(2),[app,[app,like,$VAR(2)],$VAR(4)]]]

 (everyone+likes)+someone -
[xst,$VAR(4),[all,$VAR(2),[app,[app,like,$VAR(4)],$VAR(2)]]]

25. [john,seeks,someone]:s

john+seeks+someone -
[app,[app,seek,[lmd,$VAR(2),[lmd,$VAR(0),[xst,$VAR(3),[app,[app,
$VAR(2),$VAR(3)],$VAR(0)]]]]],j]

 (john+seeks)+someone -
[xst,$VAR(3),[app,[app,seek,[lmd,$VAR(2),[lmd,$VAR(0),[app,[app,
$VAR(2),$VAR(3)],$VAR(0)]]]],j]]

26. [everyone,seeks,someone]:s

everyone+seeks+someone -
[all,$VAR(6),[app,[app,seek,[lmd,$VAR(3),[lmd,$VAR(1),[xst,$VAR(4),
[app,[app,$VAR(3),$VAR(4)],$VAR(1)]]]]],$VAR(6)]]

 (everyone+seeks)+someone -
[xst,$VAR(6),[all,$VAR(4),[app,[app,seek,[lmd,$VAR(3),[lmd,$VAR(1),
[app,[app,$VAR(3),$VAR(6)],$VAR(1)]]]],$VAR(4)]]]

27. [bill,thinks,someone,walks]:s

bill+thinks+someone+walks -
[app,[app,think,[xst,$VAR(1),[app,walk,$VAR(1)]]],b]

 (bill+thinks)+someone+walks -
[xst,$VAR(1),[app,[app,think,[app,walk,$VAR(1)]],b]]

28. [bill,thinks,some,man,shows,everyone,john]:s

bill+thinks+ (some+man)+shows+everyone+john -
[app,[app,think,[xst,$VAR(4),[and,[app,man,$VAR(4)],[all,$VAR(2),
[app,[app,[app,show,$VAR(2)],j],$VAR(4)]]]]],b]

bill+thinks+ ( (some+man)+shows)+everyone+john -
[app,[app,think,[all,$VAR(5),[xst,$VAR(2),[and,[app,man,$VAR(2)],
[app,[app,[app,show,$VAR(5)],j],$VAR(2)]]]]],b]
```

```
 (bill+thinks)+ (some+man)+shows+everyone+john -
[xst,$VAR(4),[and,[app,man,$VAR(4)],[app,[app,think,[all,$VAR(2),
[app,[app,[app,show,$VAR(2)],j],$VAR(4)]]],b]]]

 (bill+thinks+ (some+man)+shows)+everyone+john -
[all,$VAR(5),[app,[app,think,[xst,$VAR(2),[and,[app,man,$VAR(2)],
[app,[app,[app,show,$VAR(5)],j],$VAR(2)]]]],b]]

29. [the,book,that,john,shows,everyone]:n

the+book+b(that+ (john+shows)+everyone) -
[iota,$VAR(7),[app,[app,and,[app,book,$VAR(7)]],[all,$VAR(2),[app,
[app,[app,show,$VAR(2)],$VAR(7)],j]]]]

30. [the,man,whom,john,likes,the,picture,of]:n

the+man+b(whom+john+likes+the+picture+of) -
[iota,$VAR(8),[and,[app,man,$VAR(8)],[app,[app,like,[iota,$VAR(1),
[app,[app,[app,of,$VAR(8)],picture],$VAR(1)]]],j]]]

31. [the,man,the,picture,of,whom,john,likes]:n

the+man+b( ( (the+picture+of)+whom)+john+likes) -
[iota,$VAR(8),[and,[app,man,$VAR(8)],[app,[app,like,[iota,$VAR(2),
[app,[app,[app,of,$VAR(8)],picture],$VAR(2)]]],j]]]

32. [the,man,whose,book,john,likes,the,picture,of]:n

the+man+b( (whose+book)+john+likes+the+picture+of) -
[iota,$VAR(10),[and,[app,man,$VAR(10)],[app,[app,like,[iota,$VAR(1),
[app,[app,[app,of,[iota,$VAR(4),[and,[app,book,$VAR(4)],[app,poss,
$VAR(10),$VAR(4)]]]],picture],$VAR(1)]]],j]]]

33. [the,man,the,picture,of,whose,book,john,likes]:n

the+man+b( ( (the+picture+of)+whose+book)+john+likes) -
[iota,$VAR(10),[and,[app,man,$VAR(10)],[app,[app,like,[iota,$VAR(2),
[app,[app,[app,of,[iota,$VAR(4),[and,[app,book,$VAR(4)],[app,poss,
$VAR(10),$VAR(4)]]]],picture],$VAR(2)]]],j]]]

34. [john,likes,mary,and,mary,bill]:s

 (john+likes+mary)+and+mary+bill -
[and,[app,[app,like,m],j],[app,[app,like,b],m]]
```

```
35. [mary,shows,the,dog,itself]:s

 (mary+shows+the+dog)+itself -
[app,[app,[app,show,[iota,$VAR(2),[app,dog,$VAR(2)]]],m],m]

 ( (mary+shows)+the+dog)+itself -
[app,[app,[app,show,[iota,$VAR(0),[app,dog,$VAR(0)]]],[iota,$VAR(0),
[app,dog,$VAR(0)]]],m]

36. [mary,shows,john,to,himself]:s

mary+ ( (shows+john)+to)+himself -
[app,[[app,show,[app,to,m]],j],m]

37. [mary,talks,to,john,about,himself]:s

mary+ ( (talks+to+john)+about)+himself -
[app,[app,[app,talk,[app,to,j]],[app,about,m]],m]

38. [john,thinks,bill,likes,him]:s

 (john+thinks)+ (bill+likes)+him -
[app,[app,think,[app,[app,like,j],b]],j]

39. [john,likes,him]:s

40. [john,seeks,him]:s
    no
?-
```