Difference Lists and Difference Bags for Logic Programming of Categorial Deduction*

F. Xavier Lloré† and Glyn Morrill‡

†Dept. I+D/NLU
Micro Focus S.A.
Còrsega, 451
08025 Barcelona
xav@iss.es

‡Dept. de Llenguatges
i Sistemes Informàtics
Univ. Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona
morrill@lsi.upc.es

Abstract

We show how difference lists can be used for systematically compiled linear clauses for Lambek categorial grammar and its generalisations, in analogy with standard Horn clauses for CF grammar. We also consider use of difference bags for partitioning of linear sequents, and methods for ambiguity and polymorphism.

Keywords: Lambek calculus, type logical grammar, linear logic, logic programming, constraint propagation.

1 Logic programming and CF grammar

Logic programming is a paradigm of computation as proof search. Programs are presented as sets of constraints expressed by logical formulas, and inputs are presented as querying whether certain formulas can be shown to follow from these constraints. Accordingly, program execution is search for proofs and output comprises the values of input variables for which proof can be shown.

Different logic programming languages can be obtained by selecting different fragments of different logics, choice depending critically on the efficiency of consequent proof search. Most famously, the PROLOG language programs exclusively through Horn clauses: formulas having the propositional form $A_0 \vee \neg A_1 \vee \ldots \vee \neg A_n, n \geq 0$ (equivalently: $A_1 \wedge \ldots \wedge A_n \rightarrow A_0$ with $A_1 \wedge \ldots \wedge A_n$ for $n{=}0$ defined as the true formula $\top$) where each A_i is an atom. In general

*To appear in the Proceedings of SEPLN XI, Deusto.

we shall refer to program clauses by $\mathcal{PCLS}$ and query clauses by $\mathcal{AGENDA}$; PROLOG has propositional structure as follows, where we use the right-to-left implication $\leftarrow$ and conjunction and the conjunction identity $\top$.

(1) $\quad \mathcal{PCLS} \quad ::= \quad \mathcal{ATOM} \leftarrow \mathcal{AGENDA}$
$\quad\quad\quad\quad\quad\quad \mathcal{AGENDA} \quad ::= \quad \top \mid \mathcal{GOAL} \wedge \mathcal{AGENDA}$

In PROLOG the notion of $\mathcal{GOAL}$ on which these depend is that of atom:

(2) $\quad \mathcal{GOAL} \quad ::= \quad \mathcal{ATOM}$

We represent a sequence of program clauses making up a program by Γ, and the query as to whether agenda A follows from Γ by $\Gamma \Rightarrow A$. Then PROLOG logic programming is guided by the following:

(3) $\quad \Gamma \Rightarrow \top$

(4) $$\frac{\Gamma \Rightarrow B_1 \wedge \ \ldots \ \wedge B_n \wedge C}{\Gamma \Rightarrow A \wedge C} A \leftarrow B_1 \wedge \ \ldots \ \wedge B_n \wedge \top \in \Gamma$$

The termination condition is to arrive at the "empty" agenda; the non-deterministic rule (4) is called resolution: working from conclusion to premise a goal is replaced by the subgoals of a program clause with head that goal.

The computational power of such a simple scheme comes from allowing atoms to be made up of predicate symbols applied to terms, the terms being constructed out of function symbols, constants and variables in the usual way. Each program clause is understood as its universal closure. In a sequent $\Gamma \Rightarrow A$, A is understood as existentially closed, and a successful execution computes terms representing values of the agenda variables such that the sequent is valid. Taking into account this predicate logical structure the resolution rule is dependent on the unifiability of the agenda goal and program clause head, and each time a program clause is selected for resolution its variables are refreshed. Where $D\sigma$ represents the result of applying to formula D a most general unifier of A and A' we have:

(5) $$\frac{\Gamma \Rightarrow B_1\sigma \wedge \ \ldots \ \wedge B_n\sigma \wedge C\sigma}{\Gamma \Rightarrow A \wedge C} A' \leftarrow B_1 \wedge \ \ldots \ \wedge B_n \wedge \top \in \Gamma$$

There is a natural relation between CF grammar and PROLOG's Horn clauses (Kowalski 1974; Peirera and Warren 1980). Consider the following sim-

ple CF grammar.

(6)
$$\begin{array}{lcl} \text{S} & \rightarrow & \text{N VP} \\ \text{VP} & \rightarrow & \text{TV N} \\ \text{N} & \rightarrow & \textbf{John} \\ \text{N} & \rightarrow & \textbf{Mary} \\ \text{TV} & \rightarrow & \textbf{likes} \end{array}$$

Each rule is translated into a Horn clause thus ($j - k$: A is application of predicate A to arguments j and k, usually written $A(j,k)$):

(7)
$$\frac{i_0 - i_n\text{: } A_0 \leftarrow i_0 - i_1\text{: } A_1 \wedge \ \ldots \ \wedge i_{n-1} - i_n\text{: } A_n \wedge \top}{A_0 \rightarrow A_1 \ \ldots \ A_n} \quad \begin{array}{l} i_0, \ldots, i_n \\ \text{distinct variables} \end{array}$$

The result is a formulation in which $j - k$: A is read as stating that the difference between j and k is a string of words of category A. The markers j and k can be read either as string positions (left of the first word is position 0, right of the first word position 1, right of the second position 2, etc.) or as difference lists with head j and tail k representing the prefix of j that is the difference between j and k. It is the difference list reading that concerns us here. For the example grammar we obtain the following (for convenience we abbreviate $B_1 \wedge \ \ldots \ \wedge B_n \wedge \top$ as $B_1 \wedge \ \ldots \ \wedge B_n$ and $A \leftarrow \top$ as A):

(8)
$$\begin{array}{lcl} 1 = i - k\text{: S} & \leftarrow & i - j\text{: N} \wedge j - k\text{: VP} \\ 2 = i - k\text{: VP} & \leftarrow & i - j\text{: TV} \wedge j - k\text{: N} \\ 3 = [\textbf{John}|i] - i\text{: N} & & \\ 4 = [\textbf{Mary}|i] - i\text{: N} & & \\ 5 = [\textbf{likes}|i] - i\text{: TV} & & \end{array}$$

In general, to find out whether string i is of category A we query whether the unit agenda $i - []$: $A \wedge \top$ follows from the program obtained by compiling the grammar. To recognise 'John likes Mary' as a sentence for instance there is the following.

(9)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{1,2,3,4,5\} \Rightarrow \top}{\{1,2,3,4,5\} \Rightarrow [\textbf{Mary}] - []\text{: N}}4}{\{1,2,3,4,5\} \Rightarrow [\textbf{likes}, \textbf{Mary}] - j\text{: TV} \wedge j - []\text{: N}}5}{\{1,2,3,4,5\} \Rightarrow [\textbf{likes}, \textbf{Mary}] - []\text{: VP}}2}{\{1,2,3,4,5\} \Rightarrow [\textbf{John}, \textbf{likes}, \textbf{Mary}] - j\text{: N} \wedge j - []\text{: VP}}3}{\{1,2,3,4,5\} \Rightarrow [\textbf{John}, \textbf{likes}, \textbf{Mary}] - []\text{: S}}1$$

Building a parser for CF grammar as a Horn clause program with difference list arguments is an efficient and elegant way of combining the top down classical proof procedure with the bottom up constraint propagation of the input string

terminal categories. However there is a price to pay: left recursive rules (rules with the same category mother and left daughter) lead to infinite cycling in PROLOG's depth-first search strategy, and such left recursion can be implicit in a chain of rules.

2 Generalised logic programming

A first kind of generalisation is one allowing goals to have universal quantifier prefixes. Then $GOAL$ is redefined thus:

(10) $\mathcal{GOAL} \quad ::= \quad \mathcal{ATOM} \mid \forall x \mathcal{GOAL}$

In the case that an agenda goal is of the form $\forall x A$ it is required to show that A obtains for all values of x. This can be done by proving the result of substituting $\mathbf{k}$ for x in A, $A[\mathbf{k}/x]$, where $\mathbf{k}$ is a new constant (a so-called Skolem constant): since it is a symbol bearing no special relation to any other (it is new), and represents an arbitrary value, showing $A[\mathbf{k}/x]$ is sufficient to show $\forall x A$.[1]

(11) $$\frac{\Gamma \Rightarrow A[\mathbf{k}/x] \wedge B}{\Gamma \Rightarrow \forall x A \wedge B} \forall,\ \mathbf{k} \text{ new constant}$$

A second generalisation is to also allow goals to be higher order, i.e. implicational:

(12) $\mathcal{GOAL} \quad ::= \quad \mathcal{ATOM} \mid \forall x \mathcal{GOAL} \mid (\mathcal{AGENDA} \leftarrow \mathcal{PCLS})$

For this the added rule is essentially the deduction (meta)theorem:

(13) $$\frac{\Gamma, A \Rightarrow B \qquad \Gamma \Rightarrow C}{\Gamma \Rightarrow (B \leftarrow A) \wedge C} \mathrm{DT}$$

Using this expanded formalism an attempt can be made to model filler-gap dependencies by coding fronted elements such as relative pronouns with implicational clauses (cf. Pareschi 1989, Pareschi and Miller 1990). Thus, suppose we wish to state that $[\mathbf{that}|i] - j$ is a relative clause if $i - k$ would be an S if $j - k$ was an N, i.e. that the body of a relative clause may be a sentence lacking an object at its right periphery. Then we may add (14).

(14) $[\mathbf{that}|i] - j$: R $\leftarrow \forall k (i - k$: S $\leftarrow j - k$: N$)$

Now 'that John likes' can be shown as in Figure 1.

[1] Skolemisation is usually seen as applying to existential quantifiers; the occurrence of the universal in succedent position however is implicitly negative, with a consequent alternation. In general Skolemisation requires not just constants but Skolem function symbols the arguments of which are the variables of the universal quantifiers having the existential within their scope. In this respect what is presented is a slight simplification.

$$
\begin{array}{lcl}
1 = i - k\text{: S} & \leftarrow & i - j\text{: N} \wedge j - k\text{: VP} \\
2 = i - k\text{: VP} & \leftarrow & i - j\text{: TV} \wedge j - k\text{: N} \\
3 = [\mathbf{John}|i] - i\text{: N} & & \\
4 = [\mathbf{Mary}|i] - i\text{: N} & & \\
5 = [\mathbf{likes}|i] - i\text{: TV} & & \\
6 = [\mathbf{that}|i] - j\text{: R} & \leftarrow & \forall k(i - k\text{: S} \leftarrow j - k\text{: N}) \\
7 = [] - \mathbf{k}\text{: N} & &
\end{array}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{1,2,3,4,5,6,7\} \Rightarrow \top}{\{1,2,3,4,5,6,7\} \Rightarrow [] - \mathbf{k}\text{: N}}\,7}{\{1,2,3,4,5,6,7\} \Rightarrow [\mathbf{likes}] - j\text{: TV}, j - \mathbf{k}\text{: N}}\,5}{\{1,2,3,4,5,6,7\} \Rightarrow [\mathbf{likes}] - \mathbf{k}\text{: VP}}\,2}{\{1,2,3,4,5,6,7\} \Rightarrow [\mathbf{John}, \mathbf{likes}] - j\text{: N}, j - \mathbf{k}\text{: VP}}\,3}{\{1,2,3,4,5,6,7\} \Rightarrow [\mathbf{John}, \mathbf{likes}] - \mathbf{k}\text{: S}}\,1 \qquad \dfrac{\{1,2,3,4,5,6\} \Rightarrow \top}{\{1,2,3,4,5,6\} \Rightarrow [\mathbf{John}, \mathbf{likes}] - \mathbf{k}\text{: S} \leftarrow [] - \mathbf{k}\text{: N})}\,\forall}{\{1,2,3,4,5,6\} \Rightarrow \forall k([\mathbf{John}, \mathbf{likes}] - k\text{: S} \leftarrow [] - k\text{: N})}\,6}{\{1,2,3,4,5,6\} \Rightarrow [\mathbf{that}, \mathbf{John}, \mathbf{likes}] - []\text{: R}}\,\mathrm{DT}
$$

Figure 1: Derivation with implicational goal

The paradigm of logic programming considered here falls within intuitionistic logic. Each resource of inference (e.g. the Horn clauses compiled from rewrite rules) may be used once, many times, or not at all in a derivation. This corresponds to the way phrase structure grammar locates grammatical properties in reusable phrase structure rules; the association of words with their lexical categories of itself says nothing about those words' distribution. In categorial grammar, by contrast, distributional properties are encoded in the structured lexical categories of words. Parsing as deduction inferences based on such properties are therefore quantified by the occurrences of words in the string to be analysed; resources must be used, and used but once. This allows us to adopt as a framework linear logic (Girard 1987) which is much more restrictive than intuitionistic logic, meaning that the space of proof search in logic programming is narrowed at the level of implementation. In particular for example the non-reusability of resources eliminates the possibility of cycling arising from re-use of left-recursive Horn clauses. It is to linear logic (programming; see Hodas and Miller 1994) then that we now turn.

3 Linear logic programming

We work with the multiplicative fragment of linear logic: the (logic programming right-to-left) implication is written $\circ\!\!-$; the conjunction $\otimes$; and the conjunction identity $\mathbf{1}$. Program clauses, agendas and goals are defined as before, but with these new connectives. For example, allowing implicational goals:

$$\begin{array}{lll} \mathcal{PCLS} & ::= & \mathcal{ATOM} \circ\!\!- \mathcal{AGENDA} \\ \mathcal{AGENDA} & ::= & \mathbf{1} \mid \mathcal{GOAL} \otimes \mathcal{AGENDA} \\ \mathcal{GOAL} & ::= & \mathcal{ATOM} \mid (\mathcal{AGENDA} \circ\!\!- \mathcal{PCLS}) \end{array} \tag{15}$$

The propositional linear logic programming rules are as follows. The termination condition now requires all the program clauses to have been consumed so that the empty agenda only follows from the empty program database:

$$\Rightarrow \mathbf{1} \tag{16}$$

The resolution rule uses up and therefore removes from the program database the program clause against which resolution is performed:

$$\frac{\Gamma \Rightarrow B_1 \otimes \ldots \otimes B_n \otimes C}{A \circ\!\!- B_1 \otimes \ldots \otimes B_n \otimes \mathbf{1}, \Gamma \Rightarrow A \otimes C} \tag{17}$$

The deduction theorem rule now distributes the program clauses between its two premises:

$$\frac{A, \Gamma \Rightarrow B \qquad \Delta \Rightarrow C}{\Gamma, \Delta \Rightarrow (B \circ\!\!- A) \otimes C}\mathrm{DT} \tag{18}$$

This last rule creates problems as it stands, for if the conclusion program database contains n clauses, there are 2^n ways of choosing Γ and Δ. We shall resolve this by means of constraint propagation through "difference bags" (see also the "lazy splitting" mechanism of Hodas and Miller 1994). The method involves firstly naming with a unique index each of the program clauses to appear in a derivation. Re-presenting the programming rules as they are when the antecedents are coded by these bags of indices we obtain this:

$$\{\} \Rightarrow \mathbf{1} \tag{19}$$

$$\frac{I \Rightarrow B_1 \otimes \ldots \otimes B_n \otimes C}{\{i, I\} \Rightarrow A \otimes C} i = A \circ\!\!- B_1 \otimes \ldots \otimes B_n \otimes \mathbf{1} \tag{20}$$

$$\frac{\{i, I\} \Rightarrow B \qquad J \Rightarrow C}{I \cup J \Rightarrow (B \circ\!\!- A) \otimes C}\mathrm{DT},\ i = A \tag{21}$$

Now the antecedents can be re-coded not as bags, but as "difference bags", so that $I - J \Rightarrow A$ says that A can be (linearly) proved using up those clauses in I not appearing in J. The task of demonstrating that agenda A follows from clauses Γ is now rendered as that of demonstrating $I - \{\} \Rightarrow A$ where I is the bag of indices of Γ.

(22) $$I - I \Rightarrow \mathbf{1}$$

(23) $$\frac{I - J \Rightarrow B_1 \otimes \ldots \otimes B_n \otimes C}{\{i, I\} - J \Rightarrow A \otimes C} i = A \circ\!\!- B_1 \otimes \ldots \otimes B_n \otimes \mathbf{1}$$

(24) $$\frac{\{i, I\} - J \Rightarrow B \qquad J - K \Rightarrow C}{I - K \Rightarrow (B \circ\!\!- A) \otimes C} \text{DT},\ i = A, \neg i \in J$$

The DT condition requires that the hypothetical clause i is used up in proving the first premise. Assuming a depth first search starting with the first premise, a check is made before passing to the second that it is not in the residue J being passed on as available. We shall see examples when we come on to proofs with linear clauses compiled from categorial logic.

4 Categorial grammar

The types (or: formulas) of implicational Lambek calculus are freely generated from primitives by binary infix connectives / ("over") and \ ("under"). A sequent, $\Gamma \Rightarrow A$, comprises a succedent formula A and one or more formula occurrences in the antecedent configuration Γ which is organised as a sequence for the associative calculus **L** of Lambek (1958). The Gentzen-style sequent presentation enjoys Cut-elimination: every theorem can be generated without the use of Cut. In the following the parenthetical notation $\Gamma(\Delta)$ represents a configuration containing a distinguished subconfiguration Δ.

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

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

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

By way of example, the theorems "lifting" A $\Rightarrow$ B/(A\B) and "composition" A\B, B\C $\Rightarrow$ A\C are generated as follows.

(26)
$$\dfrac{\dfrac{\text{A} \Rightarrow \text{A} \quad \text{B} \Rightarrow \text{B}}{\text{A},\ \text{A}\backslash\text{B} \ \Rightarrow \text{B}}\backslash\text{L}}{\text{A} \Rightarrow \text{B}/(\text{A}\backslash\text{B})}/\text{R} \qquad \dfrac{\text{A} \Rightarrow \text{A} \quad \dfrac{\text{B} \Rightarrow \text{B} \quad \text{C} \Rightarrow \text{C}}{\text{B},\ \text{B}\backslash\text{C} \Rightarrow \text{C}}\backslash\text{L}}{\dfrac{\text{A},\ \text{A}\backslash\text{B},\ \text{B}\backslash\text{C} \Rightarrow \text{C}}{\text{A}\backslash\text{B},\ \text{B}\backslash\text{C} \ \Rightarrow \text{A}\backslash\text{C}}\backslash\text{R}}\backslash\text{L}$$

In Morrill (1995c) it is shown how **L** can be compiled into linear logic clauses according to relational models for **L** (van Benthem 1991). Each formula is interpreted as a relation (= set of ordered pairs) over a set V:

(27)
$$\begin{array}{lcl} D(A\backslash B) & = & \{\langle v_2, v_3\rangle | \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \\ D(B/A) & = & \{\langle v_1, v_2\rangle | \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \end{array}$$

We can compile the categorial logic of Lambek calculus into linear logic by coding the directional information in the interpretation of the former in the predicate-logical structure of the latter. Hence:

(28) a.
$$\dfrac{\forall i(i - \gamma\text{: } B \quad \multimap \quad i - \beta\text{: } A)}{\beta - \gamma\text{: } A\backslash B}$$

b.
$$\dfrac{\forall k(\alpha - k\text{: } B \quad \multimap \quad \beta - k\text{: } A)}{\alpha - \beta\text{: } B/A}$$

Clausal form can be obtained by omitting positive universal quantifiers, leaving their variables free and implicitly universally quantified. For negative universal (i.e. existential) quantifiers we can substitute Skolem constants at compile time.[2] Then the compilation appears as follows (see also Moortgat 1990a, 1992, Oehrle 1994), regulated according to polarity p; $\overline{p}$ is the polarity complementary to p.

(29) a.
$$\dfrac{\alpha - \gamma\text{: } B^p \quad \multimap \quad \alpha - \beta\text{: } A^{\overline{p}}}{\beta - \gamma\text{: } A\backslash B^p}\ \alpha \text{ new variable/constant as } p +/-$$

b.
$$\dfrac{\alpha - \gamma\text{: } B^p \quad \multimap \quad \beta - \gamma\text{: } A^{\overline{p}}}{\alpha - \beta\text{: } B/A^p}\ \gamma \text{ new variable/constant as } p +/-$$

The clausal compilation of CF grammar in first order Horn clauses applies to a representation in either difference lists or string positions. The present article aims to show that the compilation of categorial logic in higher order linear clauses likewise applies not only for string positions (as in Morrill 1995c) but also for difference lists.

[2] Again, a simplification of normal Skolemisation.

We compile a type assignment α: A by unfolding $[\alpha|j]$ – j: A^+ where j is a tail variable. Consider compilation as follows for a transitive verb assignment to 'likes'.

(30)
$$\dfrac{\dfrac{l - k\text{: } \mathrm{S}^+ \quad \circ\!\!- \quad l - [\mathbf{likes}|j]\text{: } \mathrm{N}^-}{[\mathbf{likes}|j] - k\text{: } \mathrm{N}\backslash\mathrm{S}^+} \quad \circ\!\!- \quad j - k\text{: } \mathrm{N}^-}{[\mathbf{likes}|j] - j\text{: } (\mathrm{N}\backslash\mathrm{S})/\mathrm{N}^+}$$

The nested implications will be flattened into "uncurried" form so that the subgoals needed to demonstrate the head of a clause already form a unit. Otherwise there is no further manipulation. For an atomic assignment such as **John**: N the result of compilation is simply $[\mathbf{John}|j]$ – j: N. The entire lexicon can be precompiled in this way, schematising over the tail variable. For the analysis of an actual string however the tail variables for each word occurence are instantiated for the position of occurrence. By way of example then for 'John likes Mary' there is the following.

(31) $1 = [\mathbf{John}, \mathbf{likes}, \mathbf{Mary}] - [\mathbf{likes}, \mathbf{Mary}]\text{: N}$
$2 = l - k\text{: S} \circ\!\!- l - [\mathbf{likes}, \mathbf{Mary}]\text{: N} \otimes [\mathbf{Mary}] - k\text{: N}$
$3 = [\mathbf{Mary}] - []\text{: N}$

$$\dfrac{\dfrac{\dfrac{}{\{\} - \{\} \Rightarrow \mathbf{1}}}{\{3\} - \{\} \Rightarrow [\mathbf{Mary}] - []\text{: N}}\,3}{\{1,3\} - \{\} \Rightarrow [\mathbf{John}, \mathbf{likes}, \mathbf{Mary}] - [\mathbf{likes}, \mathbf{Mary}]\text{: N} \otimes [\mathbf{Mary}] - []\text{: N}}\,1}{\{1,2,3\} - \{\} \Rightarrow [\mathbf{John}, \mathbf{likes}, \mathbf{Mary}] - []\text{: S}}\,2$$

Consider now compilation for the higher order type of a relative pronoun.

(32)
$$\dfrac{[\mathbf{that}|j] - k\text{: } \mathrm{R}^+ \quad \circ\!\!- \quad \dfrac{j - \mathbf{k}\text{: } \mathrm{S}^- \quad \circ\!\!- \quad k - \mathbf{k}\text{: } \mathrm{N}^+}{j - k\text{: } \mathrm{S}/\mathrm{N}^-}}{[\mathbf{that}|j] - j\text{: } \mathrm{R}/(\mathrm{S}/\mathrm{N})^+}$$

There is the derivation of Figure 2 for the relative clause 'that John likes'.

The compilation given above makes no special ordering of the goals on an agenda. In the case of Horn clauses for CF rules there was a left-to-right ordering according to which the head list of the head atom equals the head of the first (leftmost) subgoal, the tail list of the head atom equals the tail of the last (rightmost) subgoal, and the tail of the nth subgoal equals the head of the n+1th. This gives a flow of information whereby the head of the n+1th goal at the moment of call is instantiated to the tail of the nth goal just computed. We note here that the same heuristic ordering of linear clauses can be chosen: choosing as first subgoal the goal sharing its head list with that of the head

$1 = [\mathbf{that}, \mathbf{John}, \mathbf{likes}] - k\colon \mathrm{R} \circ\!\!- ([\mathbf{John}, \mathbf{likes}] - \mathbf{k}\colon \mathrm{S} \circ\!\!- k - \mathbf{k}\colon \mathrm{N})$
$2 = [\mathbf{John}, \mathbf{likes}] - [\mathbf{likes}]\colon \mathrm{N}$
$3 = m - n\colon \mathrm{S} \circ\!\!- m - [\mathbf{likes}]\colon \mathrm{N} \otimes [] - n\colon \mathrm{N}$
$4 = [] - \mathbf{k}\colon \mathrm{N}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\} - J \Rightarrow \mathbf{1}}{\{4\} - J \Rightarrow [] - \mathbf{k}\colon \mathrm{N}}\,4,\ J = \{\}}{\{4,2\} - J \Rightarrow [\mathbf{John}, \mathbf{likes}] - [\mathbf{likes}]\colon \mathrm{N} \otimes [] - \mathbf{k}\colon \mathrm{N}}\,2}{\{4,2,3\} - J \Rightarrow [\mathbf{John}, \mathbf{likes}] - \mathbf{k}\colon \mathrm{S} \qquad \{\} - \{\} \Rightarrow \mathbf{1}}\,3\ \mathrm{DT}}{\{2,3\} - \{\} \Rightarrow [\mathbf{John}, \mathbf{likes}] - \mathbf{k}\colon \mathrm{S} \circ\!\!- [] - \mathbf{k}\colon \mathrm{N}}\,1}{\{1,2,3\} - \{\} \Rightarrow [\mathbf{that}, \mathbf{John}, \mathbf{likes}] - []\colon \mathrm{R}}$$

Figure 2: Linear derivation with implicational goal

atom, and choosing as $n+1$th that sharing its head list with the tail list of the nth.

5 Discontinuity

We turn now to treatment of categorial logics generalising Lambek calculus (Morrill 1994b). One broad kind of generalisation gives sequent calculus for "multimodal" Lambek calculi in which we have several families of connectives $\{/_i, \backslash_i\}_{i \in \{1,\ldots,n\}}$ (Moortgat and Morrill 1991). The sequent configurations are structured by a labelled binary bracketing; further *structural* rules can be added:

(33) $$\dfrac{}{A \Rightarrow A}\,\mathrm{id} \qquad \dfrac{\Gamma \Rightarrow A \quad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}\,\mathrm{Cut}$$

(34) a. $$\dfrac{\Gamma \Rightarrow A \quad \Delta(B) \Rightarrow C}{\Delta([_i\Gamma, A\backslash_i B]) \Rightarrow C}\backslash_i \mathrm{L} \qquad \dfrac{[_iA, \Gamma] \Rightarrow B}{\Gamma \Rightarrow A\backslash_i B}\backslash_i \mathrm{R}$$

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

Multimodal 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 (1994b, 1995b), surmounting technical difficulties with Moortgat (1988, 1990b, 1991). The treatment of

Morrill (1994b, 1995b) is a multimodal calculus with three families of connectives: in addition to the implications, $\{/, \backslash\}$ of **L**, there are implications forming "split strings", $\{<, >\}$, and implications for "interpolation" $\{\uparrow, \downarrow\}$, 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 $s_1{+}s_2{+}s_3 = (s_1, s_3)Ws_2$. The discontinuity calculus formulates the logic of split strings and interpolation in split strings, adding this to the **L** logic of concatenation. In Morrill (1995a) the implicational fragment is implemented using groupoid labelling in higher order linear clauses.

The discontinuity calculus of Morrill (1994b, ch.4) provides a natural space within which to work, but in constructing types freely under the type-forming operators it includes abstractions which are not required, and while it is conceptually harmless, computationally such unwanted abstraction creates unwarranted complexity. Intuitively we want to work more concretely with strings and split strings rather than with a homogeneous total algebra. Quite generally, when this situation arises it is normal to impose a discipline of typing or sorting on a formalism. The appendix of Morrill (1995b) therefore presents a formulation of the discontinuity calculus 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 partially associative and partially commutative groupoid unification.

In Morrill (1994a) this version with sortal restrictions on the formation of category formulas (corresponding to the prosodic data types string and split string) is implemented using simultaneous string position/relational interpretation and groupoid labelling. The groupoid labelling was needed because groupoid Skolemisation was needed to control use of hypothetical resources. However, groupoid terms require some computation of unification under associativity. In the present work the indexical management of resource use allows us to dispense with the groupoids; and as above, we represent substrings not by string positions, but by difference lists.

We give the sorted discontinuity a relational interpretation with respect to set V as follows. Let us assume that atomic formulas $\mathcal{A}$ are of sort string. The formulas $\mathcal{F}$ of sort string and $\mathcal{G}$ of sort split string are defined by mutual recursion thus:

$$\begin{array}{lll} \mathcal{F} & ::= & \mathcal{A} \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{G}<\mathcal{F} \mid \mathcal{F}>\mathcal{G} \mid \mathcal{G}\downarrow\mathcal{F} \\ \mathcal{G} & ::= & \mathcal{F}\uparrow\mathcal{F} \end{array} \tag{35}$$

Formulas of sort $\mathcal{F}$ (strings) are interpreted as subsets of $V \times V$; formulas of

sort $\mathcal{G}$ (split strings) subsets of $V \times V \times V \times V$.

(36)
$$\begin{array}{lcl} D(A\backslash B) & = & \{\langle v_2, v_3\rangle | \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \\ D(B/A) & = & \{\langle v_1, v_2\rangle | \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_3\rangle \in D(B)\} \\ \\ D(A{>}B) & = & \{\langle v_3, v_4\rangle | \forall \langle v_1, v_2\rangle \in D(A), \langle v_1, v_2, v_3, v_4\rangle \in D(B)\} \\ D(B{<}A) & = & \{\langle v_1, v_2\rangle | \forall \langle v_3, v_4\rangle \in D(A), \langle v_1, v_2, v_3, v_4\rangle \in D(B)\} \\ \\ D(A{\downarrow}B) & = & \{\langle v_2, v_3\rangle | \forall \langle v_1, v_2, v_3, v_4\rangle \in D(A), \langle v_1, v_4\rangle \in D(B)\} \\ D(B{\uparrow}A) & = & \{\langle v_1, v_2, v_3, v_4\rangle | \forall \langle v_2, v_3\rangle \in D(A), \langle v_1, v_4\rangle \in D(B)\} \end{array}$$

We spell out all the unfoldings:

(37)
$$\frac{\alpha - \gamma\colon B^p \quad \circ\!\!- \quad \alpha - \beta\colon A^{\overline{p}}}{\beta - \gamma\colon A\backslash B^p}\,\alpha \text{ new variable/constant as } p +/-$$

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

(38)
$$\frac{\alpha - \beta, \gamma - \delta\colon B^p \quad \circ\!\!- \quad \alpha - \beta\colon A^{\overline{p}}}{\gamma - \delta\colon A{>}B^p}\,\alpha, \beta \text{ new variables/constants as } p +/-$$

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

(39)
$$\frac{\alpha - \delta\colon B^p \quad \circ\!\!- \quad \alpha - \beta, \gamma - \delta\colon A^{\overline{p}}}{\beta - \gamma\colon A{\downarrow}B^p}\,\alpha, \delta \text{ new variables/constants as } p +/-$$

$$\frac{\alpha - \delta\colon B^p \quad \circ\!\!- \quad \beta - \gamma\colon A^{\overline{p}}}{\alpha - \beta, \gamma - \delta\colon B{\uparrow}A^p}$$

A subject quantifier example is as follows.

(40) $[\mathbf{someone}|k] - k\colon (\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}, [\mathbf{runs}|j] - j\colon \mathrm{N}\backslash\mathrm{S} \Rightarrow [\mathbf{someone}, \mathbf{runs}] - []\colon \mathrm{S}$

The quantifier phrase and intransitive verb unfold thus:

(41)
$$\frac{\dfrac{i - l\colon \mathrm{S}^+ \quad \circ\!\!- \quad [\mathbf{someone}|k] - k\colon \mathrm{N}^-}{i - [\mathbf{someone}|k], k - l\colon \mathrm{S}{\uparrow}\mathrm{N}^-} \quad \circ\!\!- \quad i - l\colon \mathrm{S}^+}{[\mathbf{someone}|k] - k\colon (\mathrm{S}{\uparrow}\mathrm{N}){\downarrow}\mathrm{S}^+}$$

$$\frac{m - j\colon \mathrm{S}^+ \quad \circ\!\!- \quad m - [\mathbf{runs}|j]\colon \mathrm{N}^-}{[\mathbf{runs}|j] - j\colon \mathrm{N}\backslash\mathrm{S}^+}$$

The derivation using difference bag indexed linear logic programming rules is as shown in (42).

(42) $1 = i - l$: S $\circ\!\!-$ ($i - l$: S $\circ\!\!-$ [**someone**, **runs**] − [**runs**]: N)
$2 = m$ − []: S $\circ\!\!-$ m − [**runs**]: N
3 = [**someone**, **runs**] − [**runs**]: N

$$\dfrac{\dfrac{\dfrac{\{\} - J \Rightarrow \mathbf{1}}{\{3\} - J \Rightarrow [\mathbf{someone}, \mathbf{runs}] - [\mathbf{runs}]: \mathrm{N}}\,3,\ J = \{\}}{\{2,3\} - J \Rightarrow [\mathbf{someone}, \mathbf{runs}] - []: \mathrm{S}}\,2 \qquad \{\} - \{\} \Rightarrow []}{\{2\} - \{\} \Rightarrow [\mathbf{someone}, \mathbf{runs}] - []: \mathrm{S} \circ\!\!- [\mathbf{someone}, \mathbf{runs}] - [\mathbf{runs}]: \mathrm{N}}\,\mathrm{DT}$$
$$\dfrac{}{\{1,2\} - \{\} \Rightarrow [\mathbf{someone}, \mathbf{runs}] - []: \mathrm{S}}\,1$$

6 Lexical ambiguity and polymorphism

We conclude with a couple of observations on the prospects of the present methods regarding treatment of lexical ambiguity and polymorphism.

In cases of homonymy, such as the ambiguity of English 'that' between article, relative pronoun, etc., in which the different roles are not a manifestation of polymorphism, i.e. combinatorial flexibility of the same semantics, it is natural to assume distinct lexical entries. The method of indexing then provides a simple way to manage lexical ambiguity without restarting the entire search process for each combination of lexical categorisations of words in the string to be analysed. All of the lexical categorisations for each word can be entered in the program database under the *same* index. Use of any one of these categories removes the index from the bag of available indices, and thereby excludes use of the other categories. Thus a successful derivation will use exactly one lexical assignment for each word, but the choice is delayed and subject to the search for proof.

Finally, in relation to polymorphism itself, Morrill (1994b, ch.6) advocates using conjunction in positive position (e.g. (((N\S)\(N\S))∧(CN\CN))/N for a preposition, projecting prepositional phrase modification of both nouns and verbs) and disjunction in negative position (e.g. (N\S)/(N∨(CN/CN)) for 'is', allowing both identification—'John is the chief'—and predication—'John is rich'). These require addition to the linear logic programming scheme of non-multiplicative conjunction and disjunction. Whether there is a consequent improvement in efficiency depends on how the proof algorithm operates. It can be seen for instance that the former method should work well bottom-up, when the preposition's object, once found, would serve equally for both the verb modifying possibilities and the noun modifying possibilities. The latter method on the

other hand works well top-down, for then we need to unify a goal against the head only once, as opposed to twice for two distinct clauses. For this then the notion of goal may be generalised with say the additive linear disjunction $\oplus$:

(43) $\mathcal{GOAL} \quad ::= \quad \mathcal{ATOM} \mid (\mathcal{AGENDA} \circ\!\!- \mathcal{PCLS}) \mid (\mathcal{GOAL} \oplus \mathcal{GOAL})$

The unfolding of negative disjunction is given in (44).

(44) $$\frac{\alpha - \beta\colon A^- \qquad \oplus \qquad \alpha - \beta\colon B^-}{\alpha - \beta\colon A \vee B^-}$$

The rules of proof are as follows.

(45) $$\frac{I - J \Rightarrow A \otimes C}{I - J \Rightarrow (A \oplus B) \otimes C}\oplus \qquad \frac{I - J \Rightarrow B \otimes C}{I - J \Rightarrow (A \oplus B) \otimes C}\oplus$$

References

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.

Girard, J-Y.: 1987, 'Linear Logic', *Theoretical Computer Science* **50**, 1–102.

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

Kowalski, R.A.: 1974, 'Logic for problem solving', DCL Memo 75, Dept. of AI, University of Edinburgh.

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.

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

Moortgat, M.: 1990a, 'Categorial Logics: a computational perspective', Proceedings *Computer Science in the Netherlands*.

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

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

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

Morrill, G.: 1995a, 'Clausal Proofs and Discontinuity', to appear in the *Bulletin of the Interest Group in Propositional and Predicate Logic*.

Morrill, G.: 1995b, 'Discontinuity in Categorial Grammar', to appear in *Linguistics and Philosophy*.

Morrill, G.: 1995c, 'Higher-order Linear Logic Programming of Categorial Deduction', Proceedings *Meeting of the European Chapter of the Association for Computational Linguistics*, Dublin.

Morrill, G. and T. Solias: 1993, 'Tuples, Discontinuity and Gapping', Proceedings *Meeting of the European Chapter of the Association for Computational Linguistics*, Utrecht, 287–297.

Oehrle, R.T.: 1994, 'Term-Labeled Categorial Type Systems, *Linguistics and Philosophy* **17**, 633–678.

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.

Pereira, F.C.N. and D.H.D. Warren: 1980, 'Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks', *Artificial Intelligence* **13**, 231–278.

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