Clausal Proof Nets and Discontinuity

Glyn Morrill

Report LSI–94–21–R

Clausal Proof Nets and Discontinuity

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 consider the task of theorem proving in Lambek calculi and their generalisation to "multimodal residuation calculi". These form an integral part of categorial logic, a logic of signs stemming from categorial grammar, on the basis of which language processing is essentially theorem proving. The demand of this application is not just for efficient processing of some or other specific calculus, but for methods that will be generally applicable to categorial logics.

It is proposed that multimodal cases be treated by dealing with the highest common factor of all the connectives as linear (propositional) validity. The prosodic (sublinear) aspects are encoded in labels, in effect the term-structure of quantified linear logic. The correctness condition on proof nets ("long trip condition") can be implemented by SLD resolution in linear logic with unification on labels/terms *limited to one-way matching.* A suitable unification strategy is obtained for calculi of discontinuity by normalisation of the ground goal term followed by recursive decent and redex pattern-matching on the head term.

$D(A)$ of L by residuation as follows.

(1) $D(A \bullet B) = \{s_1 + s_2 | s_1 \in D(A) \wedge s_2 \in D(B)\}$
$D(A \backslash B) = \{s | \forall s' \in D(A), s' + s \in D(B)\}$
$D(B / A) = \{s | \forall s' \in D(A), s + s' \in D(B)\}$

A sequent, $\Gamma \vdash A$, comprises a succedent formula A and one or more formula occurrences in the antecedent configuration Γ which is organised as a binary bracketed sequence for **NL**, and as a sequence for **L**. A sequent is valid if and only if in all interpretations, applying the prosodic construction indicated by the antecedent configuration to objects inhabiting its formulas always yields an object inhabiting the succedent formula. The Gentzen-style sequent presentations for **NL** in (2) and for **L** in (3) are sound and complete for this interpretation (Buszkowski 1986, Došen 1992); furthermore they enjoy 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.

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

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

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

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

(3) a. $A \vdash A$ id $\qquad \dfrac{\Gamma \vdash A \quad \Delta(A) \vdash B}{\Delta(\Gamma) \vdash B}$Cut

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

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

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

By way of example, "lifting" A ⊢ B/(A\B) is generated as follows in **NL**; it is similarly derivable in **L**.

(4)
$$\dfrac{\dfrac{\mathrm{A} \vdash \mathrm{A} \quad \mathrm{B} \vdash \mathrm{B}}{[\mathrm{A},\ \mathrm{A}\backslash\mathrm{B}] \vdash \mathrm{B}}\backslash\mathrm{L}}{\mathrm{A} \vdash \mathrm{B}/(\mathrm{A}\backslash\mathrm{B})}/\mathrm{R}$$

On the other hand "composition" A\B, B\C ⊢ A\C, while derivable as follows in **L**, is **NL**-underivable in its non-associative form: [A\B, B\C] ⊢ A\C.

(5)
$$\dfrac{\mathrm{A} \vdash \mathrm{A} \qquad \dfrac{\mathrm{B} \vdash \mathrm{B} \quad \mathrm{C} \vdash \mathrm{C}}{\mathrm{B},\ \mathrm{B}\backslash\mathrm{C} \vdash \mathrm{C}}\backslash\mathrm{L}}{\dfrac{\mathrm{A},\ \mathrm{A}\backslash\mathrm{B},\ \mathrm{B}\backslash\mathrm{C} \vdash \mathrm{C}}{\mathrm{A}\backslash\mathrm{B},\ \mathrm{B}\backslash\mathrm{C} \vdash \mathrm{A}\backslash\mathrm{C}}\backslash\mathrm{R}}\backslash\mathrm{L}$$

1.2 Multimodal Lambek Calculi

In a slightly different formulation of the sequent calculus for L we may configure antecedents with binary bracketing, and then use the NL rules together with an explicit structural rule of associativity (the double bar indicates bidirectionality):

(6)
$$\dfrac{\Gamma([\Delta_1, [\Delta_2, \Delta_3]]) \vdash A}{\Gamma([[\Delta_1, \Delta_2], \Delta_3]) \vdash A}\mathrm{A}$$

From here it is a small step to give sequent calculus for "multimodal" Lambek calculi in which we have several families of connectives $\{/_i, \backslash_i, \bullet_i\}_{i \in \{1,\ldots,n\}}$, each defined by residuation with respect to their adjunction in a "multigroupoid" $\langle L, \{+_i\}_{i \in \{1,\ldots,n\}}\rangle$ (Moortgat and Morrill 1991):

(7)
$$\begin{array}{lcl} D(A \bullet_i B) & = & \{s_1 +_i s_2 | s_1 \in D(A) \wedge s_2 \in D(B)\} \\ D(A \backslash_i B) & = & \{s | \forall s' \in D(A), s' +_i s \in D(B)\} \\ D(B /_i A) & = & \{s | \forall s' \in D(A), s +_i s' \in D(B)\} \end{array}$$

Sequent calculus can be given by indexing the brackets of NL-presentations to indicate mode of adjunction (and adding structural rules as appropriate):

(8)
$$\dfrac{}{A \vdash A}\mathrm{id} \qquad \dfrac{\Gamma \vdash A \quad \Delta(A) \vdash B}{\Delta(\Gamma) \vdash B}\mathrm{Cut}$$

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

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

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

In particular cases of course we may choose non-composite notations for the connectives and brackets. With two modes interpreted in a "bigroupoid" understood as distinguishing left-headed and right-headed adjunction we have a "headed" calculus (Moortgat and Morrill 1991). With families $\{/, \backslash, \bullet\}$ and $\{<, >, \circ\}$ for adjunctions + (associative) and $(.,.)$ (not assumed to be associative) respectively in a bigroupoid $\langle L, +, (.,.)\rangle$ we have a partially associative calculus (Oehrle and Zhang 1989, Morrill 1990). This forms two-thirds of the discontinuity calculus of Morrill (1993) which we shall be considering shortly.

1.3 Labelled Sequent Presentations

"Labelling" (Gabbay 1991) is a means of presenting proof theory which will enable us to factor out the antecedent formulas of a sequent, and its associated prosodic construction, which is made more explicit. No essential use is made, in that the labelled presentations of calculi given here

are just notational variants of ordered presentations which can be given. However, labelling is a step on the path to implementing residuation calculi in proof nets. We notate a sequent $\Gamma \vdash A$ as $a_1\colon A_1, \ldots, a_n\colon A_n \vdash \alpha\colon A$ where the multiset $\{A_1, \ldots, A_n\}_m$ comprises the formula occurrences in Γ, $a_1, \ldots, a_n$ are distinct atomic labels, and α is a term over these labels representing explicitly the prosodic construction that was represented implicitly by the structured configuration Γ. The labelled sequent calculus for **NL** is as follows:

(10) a. $a\colon A \vdash a\colon A$ id

b. $$\frac{\Gamma \vdash \alpha\colon A \quad a\colon A, \Delta \vdash \beta(a)\colon B}{\Gamma, \Delta \vdash \beta(\alpha)\colon B}\text{Cut}$$

c. $$\frac{\Gamma \vdash \alpha\colon A \quad b\colon B, \Delta \vdash \gamma(b)\colon C}{\Gamma, d\colon A\backslash B, \Delta \vdash \gamma((\alpha + d))\colon C}\backslash\text{L}$$

d. $$\frac{\Gamma, a\colon A \vdash (a + \gamma)\colon B}{\Gamma \vdash \gamma\colon A\backslash B}\backslash\text{R}$$

e. $$\frac{\Gamma \vdash \alpha\colon A \quad b\colon B, \Delta \vdash \gamma(b)\colon C}{\Gamma, d\colon B/A, \Delta \vdash \gamma((d + \alpha))\colon C}/\text{L}$$

f. $$\frac{\Gamma, a\colon A \vdash (\gamma + a)\colon B}{\Gamma \vdash \gamma\colon B/A}/\text{R}$$

g. $$\frac{a\colon A, b\colon B, \Delta \vdash \gamma((a + b))\colon C}{c\colon A\bullet B, \Delta \vdash \gamma(c)\colon C}\bullet\text{L}$$

h. $$\frac{\Gamma \vdash \alpha\colon A \quad \Delta \vdash \beta\colon B}{\Gamma, \Delta \vdash (\alpha + \beta)\colon A\bullet B}\bullet\text{R}$$

To obtain **L** an associativity equation on terms may be added, or equivalence classes of terms represented by flattening terms into lists.

1.4 Labelled Natural Deduction

For labelled Fitch-style categorial derivation (Morrill 1993), there are lexical assignment, subderivation hypothesis, and term label equation rules thus (we include Curry-Howard semantic annotation intermittently in what follows; full explications are available in the references):

(11) a. $n.\quad \alpha - \phi\colon A$ for any lexical entry

b.
$$\begin{array}{lll} n. & a_1 - x_1\colon A_1 & \text{H} \\ & \vdots & \vdots \\ n+m. & a_m - x_m\colon A_n & \text{H} \end{array}$$

c. $n.\quad \alpha - \phi\colon A$
$\alpha' - \phi'\colon A \quad = n, \text{if } \alpha = \alpha' \ \&\ \phi = \phi'$

The logical rules are:

(12) a. $n.\quad \alpha - \phi\colon A$
$m.\quad \gamma - \chi\colon A\backslash B$
$(\alpha+\gamma) - (\chi\ \phi)\colon B \quad \text{E}\backslash\ n, m$

b. n. $\boxed{a - x\colon A}$ H
m. $(a{+}\gamma) - \psi\colon B$ unique a as indicated
$\gamma - \lambda x\psi\colon A\backslash B$ I\ n, m

(13) a. n. $\alpha - \phi\colon A$
m. $\gamma - \chi\colon B/A$
$(\gamma{+}\alpha) - (\chi\ \phi)\colon B$ E/ n, m

b. n. $\boxed{a - x\colon A}$ H
m. $(\gamma{+}a) - \psi\colon B$ unique a as indicated
$\gamma - \lambda x\psi\colon B/A$ I/ n, m

(14) a. n. $\gamma - \chi\colon A{\bullet}B$
m. $\boxed{a - x\colon A}$ H
$m+1$. $\boxed{b - y\colon B}$ H
p. $\delta((a{+}b)) - \omega(x, y)\colon D$ unique a and b as indicated
$\delta(\gamma) - \omega(\pi_1\chi, \pi_2\chi)\colon D$ E$\bullet$ $n, m, m+1, p$

b. n. $\alpha - \phi\colon A$
m. $\beta - \psi\colon B$
$(\alpha{+}\beta) - (\phi, \psi)\colon A{\bullet}B$ I$\bullet$ n, m

A Fitch-style labelled calculus for the associative Lambek calculus **L** can be obtained from that for the non-associative calculus by adding a prosodic label equation:

(15) $((\alpha_1{+}\alpha_2){+}\alpha_3) = (\alpha_1{+}(\alpha_2{+}\alpha_3))$

Alternatively, the associative Lambek calculus can be given by dropping parentheses in prosodic labels. By way of example, a simple instance of relativisation can be derived by hypothetical deduction as follows:

(16)
1. *which* – $\lambda x\lambda y\lambda z[(y\ z) \wedge (x\ z)]$: (CN\CN)/(S/N)
2. *John* – **j**: N
3. *talked* – **talk**: (N\S)/PP
4. *about* – **about**: PP/N
5. $\boxed{a - x\colon \text{N}}$ H
6. *about*+a – (**about** x): PP 4, 5 E/
7. *talked*+*about*+a – (**talk** (**about** x)): N\S 3, 6 E/
8. *John*+*talked*+*about*+a – ((**talk** (**about** x)) **j**): S 2, 7 E\
9. *John*+*talked*+*about* – λx((**talk** (**about** x)) **j**): S/N 5, 8 I/
10. *which*+*John*+*talked*+*about* – $(\lambda x\lambda y\lambda z[(y\ z) \wedge (x\ z)]\ \lambda x((\textbf{talk}\ (\textbf{about}\ x))\ \textbf{j}))$: CN\CN 1, 9 E/
11. *which*+*John*+*talked*+*about* – $\lambda y\lambda z[(y\ z) \wedge ((\textbf{talk}\ (\textbf{about}\ z))\ \textbf{j})]$: CN\CN = 10

1.4.1 Multimodal Fitch natural deduction

Multimodal calculi can be presented Fitch-style by giving the same rules for each family of connectives with their associated adjunctions:

(17) a. n. $\alpha - \phi\colon A$
m. $\gamma - \chi\colon A\backslash_i B$
$(\alpha{+}_i\gamma) - (\chi\ \phi)\colon B$ E\$_i$ n, m

b. $n.$ $\boxed{a - x\colon A}$ H
$m.$ $(a{+_i}\gamma) - \psi\colon B$ unique a as indicated
$\gamma - \lambda x\psi\colon A\backslash_i B$ $\mathrm{I}\backslash_i\ n, m$

(18) a. $n.$ $\alpha - \phi\colon A$
$m.$ $\gamma - \chi\colon B/_i A$
$(\gamma{+_i}\alpha) - (\chi\ \phi)\colon B$ $\mathrm{E}/_i\ n, m$

b. $n.$ $\boxed{a - x\colon A}$ H
$m.$ $(\gamma{+_i}a) - \psi\colon B$ unique a as indicated
$\gamma - \lambda x\psi\colon B/_i A$ $\mathrm{I}/_i\ n, m$

(19) a. $n.$ $\gamma - \chi\colon A\bullet_i B$
$m.$ $\boxed{a - x\colon A}$ H
$m+1.$ $\boxed{b - y\colon B}$ H
$p.$ $\delta((a{+_i}b)) - \omega(x, y)\colon D$ unique a and b as indicated
$\delta(\gamma) - \omega(\pi_1\chi, \pi_2\chi)\colon D$ $\mathrm{E}\bullet_i\ n, m, m+1, p$

b. $n.$ $\alpha - \phi\colon A$
$m.$ $\beta - \psi\colon B$
$(\alpha{+_i}\beta) - (\phi, \psi)\colon A\bullet_i B$ $\mathrm{I}\bullet_i\ n, m$

Label equations are to be added according to the algebras of interpretation.

1.4.2 Multimodal labelled Prawitz natural deduction

Labelled deduction can also be presented Prawitz-style; for the multimodal case (without semantics) there is the following.

(20)
$$\frac{\begin{array}{cc}\vdots & \vdots\\ \gamma\colon B/_i A & \alpha\colon A\end{array}}{\gamma{+_i}\alpha\colon B}/_i\mathrm{E} \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ \alpha\colon A & \gamma\colon A\backslash_i B\end{array}}{\alpha{+_i}\gamma\colon B}\backslash_i\mathrm{E}$$

(21)
$$\frac{\begin{array}{c}\\ \gamma{+_i}a\colon B\end{array}}{\gamma\colon B/_i A}/_i\mathrm{I}^n \qquad \frac{\begin{array}{c}\\ a{+_i}\gamma\colon B\end{array}}{\gamma\colon A\backslash_i B}\backslash_i\mathrm{I}^n$$

(22)
$$\frac{\begin{array}{cc}\vdots & \vdots\\ \alpha\colon A & \beta\colon B\end{array}}{\alpha{+_i}\beta\colon A\bullet_i B}\bullet\mathrm{I}_i \qquad \frac{\begin{array}{c}\vdots\\ \delta\colon A\bullet_i B\end{array}}{\begin{array}{cc}a\colon A & b\colon B\end{array}}\bullet_i\mathrm{E}^n \quad \begin{array}{c}\vdots\\ \gamma((a{+_i}b))\colon C\end{array}\ n$$
$$\gamma(\delta)\colon C$$

2 Discontinuity

We consider residuation calculi for two kinds of discontinuity: *regular*, for discontinuous functors, and for infix binders as in quantifier raising, reflexivisation, pied piping and gapping, and *head-oriented* such as head infixation and head extraction in Germanic verb clusters and verb fronting.

In each case the essential strategy is to specify discontinuous adjunction as a *primitive* (as opposed to derived) operation in the multigroupoid prosodic algebra of multimodal Lambek calculi, with respect to which discontinuity operators are defined by residuation.

2.1 Regular Discontinuity

In the discontinuity calculus of Morrill (1993) connectives $\{/, \backslash, \bullet\}$, $\{<, >, \circ\}$ and $\{\uparrow, \downarrow, \odot\}$ are interpreted by residuation with respect to adjunctions $+$, $(.,.)$ and W respectively in a trigroupoid $\langle L^*, +, (.,.), W, \epsilon\rangle$ where $+$ is associative and has (left and right) identity $\epsilon \in L^*$, and $(.,.)$ and W satisfy the "split-wrap" equation: $(s_1, s_3)Ws_2 = s_1{+}s_2{+}s_3$. We see that (ϵ, ϵ) is a left identity for W; ϵ is desired in the interest of linguistic generalisation: to include peripherality as a special case of discontinuity. Also for linguistic reasons however, formulas are interpreted as subsets of $L = L^*\backslash\{\epsilon\}$, preventing ϵ (but not (ϵ, ϵ)) from inhabiting types. The prosodic label equations are as follows:

(23) a. $s_1{+}(s_2{+}s_3) = (s_1{+}s_2){+}s_3$
b. $s{+}\epsilon = \epsilon{+}s = s$
c. $(s_1, s_3)Ws_2 = s_1{+}s_2{+}s_3$

A verb-particle construction is derived as in (24).

(24)

1.	$(rang, up)$ – **phone**: (N\S)↑N	
2.	*John* – **j**: N	
3.	*Mary* – **m**: N	
4.	$((rang, up)W John)$ – (**phone j**): N\S	1, 2 E↑
5.	*rang*+*John*+*up* – (**phone j**): N\S	= 4
6.	*Mary*+*rang*+*John*+*up* – ((**phone j**) **m**): S	3, 5 E\

Discussion of semantics would take us outside the direct concerns of the present article; the reader is referred to e.g. Moortgat (1988, 1991) and Morrill (1992a, 1993) for explication. The effect of quantifier-raising, whereby quantifiers are to take sentential scope, is achieved by assignment of a quantifier phrase such as 'everyone' to a "quantifying-in" type (S↑N)↓S. A simple instance of quantifier-raising is shown in (25).

(25)

1.	*John* – **j**: N	
2.	*likes* – **like**: (N\S)/N	
3.	*everything* – $\lambda x \forall y(x\ y)$: (S↑N)↓S	
4.	a – x: N	H
5.	*likes*+a – (**like** x): N\S	2, 4 E/
6.	*John*+*likes*+a – ((**like** x) **j**): S	1, 5 E\
7.	*John*+*likes*+a+ϵ – ((**like** x) **j**): S	= 6
8.	$((John{+}likes, \epsilon)Wa)$ – ((**like** x) **j**): S	= 7
9.	$(John{+}likes, \epsilon)$ – λx((**like** x) **j**): S↑N	4, 8 I↑
10.	$((John{+}likes, \epsilon)W everything)$ – $(\lambda x \forall y(x\ y)\ \lambda x((\textbf{like}\ x)\ \textbf{j}))$: S	3, 9 E↓
11.	*John*+*likes*+*everything* – $\forall y$((**like** y) **j**): S	= 10

2.2 Bracket Operators

To treat head-oriented discontinuity we shall require bracket operators, interpreted in a prosodic algebra such as $\langle L, +, [.]\rangle$ where $[.]$ is a unary operation (Morrill 1992b); for nice symmetric proof theory we require that this is a 1-1 function (permutation) so that its inverse $[.]^{-1}$ is total.

(26) $[[s]^{-1}] = [[s]]^{-1} = s$

(27) $D([\,]A) = \{[s] | s \in D(A)\}$
$D([\,]^{-1}A) = \{s | [s] \in D(A)\} = \{[s]^{-1} | s \in D(A)\}$

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. Labelled Prawitz-style natural deduction for bracket operators is as follows, including $[.]$ and its inverse $[.]^{-1}$ in prosodic labels, for which there is a prosodic label equation $[[\alpha]^{-1}] = [[\alpha]]^{-1} = \alpha$.

(28)
$$\frac{\begin{array}{c}\vdots\\ \alpha\colon A\end{array}}{[\alpha]\colon [\,]A}\mathrm{I}[\,] \qquad \frac{\begin{array}{c}\vdots\\ \alpha\colon [\,]A\end{array}}{[\alpha]^{-1}\colon A}\mathrm{E}[\,]$$

(29)
$$\frac{\begin{array}{c}\vdots\\ \alpha\colon A\end{array}}{[\alpha]^{-1}\colon [\,]^{-1}A}\mathrm{I}[\,]^{-1} \qquad \frac{\begin{array}{c}\vdots\\ \alpha\colon [\,]^{-1}A\end{array}}{[\alpha]\colon A}\mathrm{E}[\,]^{-1}$$

2.3 Head-Oriented Discontinuity

Head-oriented discontinuity is obtained by combining a bracketing operation and a primitive head adjunction in a headed calculus (Moortgat and Morrill 1991): the prosodic algebra is $\langle L, +_l, +_r, +_h, [_v.], \epsilon\rangle$ with implications $\{\backslash_l, /_l\}$, $\{\backslash_r, /_r\}$, and $\{\uparrow, \downarrow\}$ interpreted by residuation w.r.t. $+_l$, $+_r$ and $+_h$ respectively. There are the following interaction axioms:

(30) $(\alpha +_r \beta) +_h \gamma = \alpha +_r (\beta +_h \gamma)$
$(\alpha +_l \beta) +_h \gamma = (\alpha +_h \beta) +_l \gamma$

And $[v]$ and $[v]^{-1}$ are bracketing operators with respect to $[_v.]$; ϵ is a left and right identity for $+_l$ and $+_r$; the bottoming-out interaction for head adjunction is (for our Dutch example):

(31) $[_v \alpha] +_h \beta = [_v \beta +_l \alpha]$

The following is a simple example of Dutch head-infixation:

(32) *boeken* $+_r [_v$ *kan* $+_l$ *lezen*$]$
books can read
"is able to read books"

It has the Prawitz-style derivation (33).

(33)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\textit{boeken}\colon \mathrm{N}} \quad \dfrac{\dfrac{}{\textit{lezen}\colon [\,v\,]^{-1}(\mathrm{N}\backslash_r\mathrm{IVi})}\mathrm{E}[\,v\,]^{-1}}{[_v \textit{lezen}]\colon \mathrm{N}\backslash_r\mathrm{IVi}}}{\textit{boeken} +_r [_v \textit{lezen}]\colon \mathrm{IVi}}\mathrm{E}\backslash_r \quad \dfrac{}{\textit{kan}\colon \mathrm{IVi}\downarrow\mathrm{VP}}}{(\textit{boeken} +_r [_v \textit{lezen}]) +_h \textit{kan}\colon \mathrm{VP}}\mathrm{E}\downarrow}{\textit{boeken} +_r ([_v \textit{lezen}] +_h \textit{kan})\colon \mathrm{VP}}=}{\textit{boeken} +_r [_v \textit{kan} +_l \textit{lezen}]\colon \mathrm{VP}}=$$
boeken lezen kan

3 Decidability

Backward-chaining Cut-free labelled sequent proof search admits only a finite number of possible rule applications for a given sequent, eliminating the principle connective of one of the (finite number of) formulas, and choosing (one of the finite number of) antecedent partitionings in the case of binary rules. This creates subgoals the complexity of which in terms of connective occurrences totals exactly one connective occurrence less. The situation obtains even with an unknown succedent

term. In a labelled sequent proof the succedent term is instantiated by backward-chaining proof search. Thus theoremhood is decidable in that, for example, to determine whether antecedent formulas under a given configuration yield a succedent formula we may compute the finitary labelled search space and then check whether one of the prosodic constructions obtained is the one desired. This holds generally for multimodal residuation calculi with interaction axioms.

3.1 Desiderata for Efficiency

There are two sources of non-determinism however in (labelled or ordered) backward-chaining sequent proof search: in choosing on which formula's principle connective to key rule application, and in choosing how to partition sequents in binary rules. With respect to the former, many sequences of choice can yield proofs with the same construction; with respect to the latter, considerable space may need to be searched before determining whether a partitioning terminated in initial identity axiom sequents or not. The former, but not the latter, problem is addressed by "proof normalisation" (for the case of Lambek calculus see Hepple 1990, which refines König 1989): fixing priorities of rule ordering to determine distinguished representatives of equivalence classes of proofs. Both drawbacks are addressed by proof nets (in linear logic), and the matrix methods of Bibal (1981) and Wallen (1990). In these, formulas are unfolded, and proofs built *from the initial sequents*. Through unfolding, the parts comprising a formula are made available for examination in parallel, rather than only in serial according to the particular embedding of connectives. By building from initial sequents we ensure effectively that only rule applications are tried which are already known to terminate successfully in initial sequents.

In the context of linear logic then proof nets have been developed as a method of eliminating redundancy in the sequent representation of proofs. For these however a correctness check (the "long trip condition") is required. Corresponding proposals have been made for Lambek calculus by Roorda (1991), in which correctness is checked by semantic labelling, and Roorda (1991) and Moortgat (1991), in which correctness is checked by prosodic labelling. The latter approach appears to apply generally to residuation calculi and constitutes the point of departure of the present proposals. As it stands, the method reduces proof net correctness to checking, by unification, satisfiability of equations in groupoids, semigroups, and so on. Yet such problems as semigroup unification are in general intractable, even though the sequent formulations of the calculi show decidability. Somewhere the method loses control of constraints, and improved management is required in order to achieve efficiency. We shall provide the necessary structure by organising the proof nets used as clauses, in fact Horn clauses, of linear logic, for which a resolution strategy is available in which at each unification step one term is ground, i.e. variable-free. This prepares the way for computational theorem proving in residuation calculi generally and illustration includes the regular and head-oriented discontinuity calculi.

4 Sequent Calculus for Classical Linear Logic

The multiplicative fragment of linear logic, with which we shall be concerned, contains binary infix connectives $\otimes$ (a conjunction "times") and $\parr$ (a disjunction "par") and a unary postfix negation $^{\perp}$ ("neg"). Sequents are of the form $\Gamma \vdash \Delta$ where configurations Γ and Δ are sequences of zero or more formulas. There are the following sequent rules, which are sound for classical logic, (and which would also be complete for classical logic if the structural rules of contraction and weakening were included). The calculus enjoys Cut-elimination.

(34) a. $\dfrac{}{A \vdash A}\,\mathrm{id}$ b. $\dfrac{\Gamma \vdash A, \Delta \quad A, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\mathrm{Cut}$

(35) a. $\dfrac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\,\mathrm{P}_L$ b. $\dfrac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\,\mathrm{P}_R$

(36) a. $\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta} \otimes \mathrm{L}$ b. $\dfrac{\Gamma \vdash A, \Delta \quad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \otimes B, \Delta, \Delta'} \otimes \mathrm{R}$

(37) a. $\dfrac{A, \Delta \vdash \Gamma \quad B, \Delta' \vdash \Gamma'}{A \parr B, \Delta, \Delta' \vdash \Gamma, \Gamma'} \parr \mathrm{L}$ b. $\dfrac{\Delta \vdash \Gamma, A, B}{\Delta \vdash \Gamma, A \parr B} \parr \mathrm{R}$

(38) a. $\dfrac{\Gamma \vdash A, \Delta}{\Gamma, A^\perp \vdash \Delta} {}^\perp\mathrm{L}$ b. $\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^\perp, \Delta} {}^\perp\mathrm{R}$

The linear implication $\multimap$ is defined by $A \multimap B = A^\perp \parr B$, so any $\{\otimes, \parr, {}^\perp, \multimap\}$ formula can be considered an abbreviation for a $\{\otimes, \parr, {}^\perp\}$ one.

4.1 Negation Normal Form

A $\{\otimes, \parr, {}^\perp\}$ formula is in negation normal form if and only if no connective occurrence falls within the scope of a negation, i.e. negations may only be immediately attached to (unnegated) atoms. We have the following proofs of the involution of negation:

(39) a. $\dfrac{\dfrac{A \vdash A}{A, A^\perp \vdash} {}^\perp\mathrm{L}}{A \vdash A^{\perp\perp}} {}^\perp\mathrm{R}$ b. $\dfrac{\dfrac{A \vdash A}{\vdash A^\perp, A} {}^\perp\mathrm{R}}{A^{\perp\perp} \vdash A} {}^\perp\mathrm{L}$

And there are the following proofs of a de Morgan law:

(40) a. $\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A}{\vdash A, A^\perp} {}^\perp\mathrm{R} \quad \dfrac{B \vdash B}{\vdash B, B^\perp} {}^\perp\mathrm{L}}{\vdash A \otimes B, A^\perp, B^\perp} \otimes \mathrm{R}}{\vdash A \otimes B, A^\perp \parr B^\perp} \parr \mathrm{R}}{(A \otimes B)^\perp \vdash A^\perp \parr B^\perp} {}^\perp\mathrm{L}$ b. $\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A}{A^\perp, A \vdash} {}^\perp\mathrm{L} \quad \dfrac{B \vdash B}{B^\perp, B \vdash} {}^\perp\mathrm{L}}{A^\perp \parr B^\perp, A, B \vdash} \parr \mathrm{L}}{A^\perp \parr B^\perp, A \otimes B \vdash} \otimes \mathrm{L}}{A^\perp \parr B^\perp \vdash (A \otimes B)^\perp} {}^\perp\mathrm{R}$

The other de Morgan law is obtained similarly. Hence using the equivalences (41) to reduce redexes of the form on the left to contractums of the form on the right, any formula is converted to a negation normal form.

(41)
$$\begin{array}{rcl} A^{\perp\perp} & = & A \\ (A \otimes B)^\perp & = & A^\perp \parr B^\perp \\ (A \parr B)^\perp & = & A^\perp \otimes B^\perp \end{array}$$

So ${}^\perp$ need not really be considered a connective in formulas but as a means of abbreviating $\{\otimes, \parr\}$ formulas in which atoms come in two flavours: positive and negative.

5 Proof Nets for Linear Logic

To validate a sequent using proof nets all formulas are first put on one side of the sequent turnstyle using the negation rules:

(42) $\Gamma \vdash \Delta$ if and only if $\Gamma, \Delta^\perp \vdash$ if and only if $\vdash \Gamma^\perp, \Delta$

They are converted to negation normal form, after which a phase of "unfolding" ensues:

(43) a. $\dfrac{A \quad B}{A \otimes B}$ b. $\dfrac{A \quad B}{A \parr B}$

We refer to the result of unfolding until all leaves are atomic as a *proof frame*. Then an attempt must be made to connect (or: link) all the (positive and negative) atoms in such as way that each is connected with exactly one other, of complementary polarity. Such connections correspond to initial sequent axioms. A connection of all the atoms is called a *proof structure*. The existence of a proof structure is a necessary, but not sufficient, condition for theoremhood (note for example that conjunction and disjunction are not distinguished!): we shall further require the "long trip condition" which effectively checks partitioning, i.e. that we have connected as initial sequent axioms atomic formula occurrences which were meant to be in the same sequent subproofs, and not ones meant to be in different subproofs.

By way of example of unfolding and linking, there is the following:

(44) $A \vdash (A \multimap B) \multimap B$ if and only if
$A \vdash (A^{\perp} \parr B)^{\perp} \parr B$ if and only if
$\vdash A^{\perp}, (A \otimes B^{\perp}) \parr B$

(45)
$$\dfrac{A^{\perp} \qquad \dfrac{\dfrac{A \quad B^{\perp}}{A \otimes B^{\perp}} \quad B}{(A \otimes B^{\perp}) \parr B}}{}$$

The proof structure (45) is also a proof net. The proof structure (46) is a proof net for @= $\otimes$ (A, B $\vdash$ A $\otimes$ B is a theorem) but not for @= $\parr$ (A, B $\vdash$ A $\parr$ B is not a theorem).

(46)
$$\dfrac{A^{\perp} \quad \dfrac{A \quad B}{A@B} \quad B^{\perp}}{}$$

A restriction to *planar* proof nets, ones with nested (i.e. non-crossing) connections, characterises (together with the long trip condition) the theorems of *cyclic* linear logic, i.e. linear logic in which exchange is limited to circular permutations.

6 Proof Nets for Lambek Calculus

In the previous section proof nets were given by moving all formulas to the right of the sequent turnstyle. Here we shall do the converse: move all formulas to the left. that is we shall perform refutation proofs. From considerations of symmetry we see that the choice is not important, but it will enable us to present our proposal in the familiar context of resolution refutation. We consider the presentation of proof nets for Lambek calculus of Roorda (1991), but our polarities are reversed. Formulas composed from the implicational connectives / and \ are signed positive for antecedent occurrences and negative for succedent occurrences, and unfolded as follows.

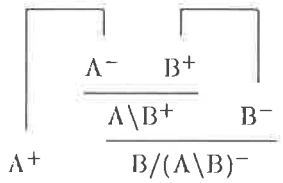
(47) $\dfrac{B^{+} \quad A^{-}}{B/A^{+}}$ $\dfrac{A^{-} \quad B^{+}}{A\backslash B^{+}}$ $\dfrac{A^{+} \quad B^{-}}{B/A^{-}}$ $\dfrac{B^{-} \quad A^{+}}{A\backslash B^{-}}$

The transmission of polarities can be understood when we see an implication as a disjunction of its consequent with the negation of its antecedent. The steps given are compilations of decomposition

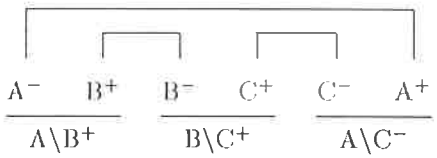
accordingly, with unfolding, involution of negation, and de Morgan laws for multiplicative conjunction or disjunction compiled in. The ordering given, which swaps the components of negative (i.e. succedent) occurrences of implications allows restriction to planar connections.

The following, for example, are proof nets for lifting A $\vdash$ B/(A\B) and composition A\B, B\C $\vdash$ A\C in **L**.

(48)



(49)



Just by considerations of symmetry however we can see that there will also be a proof structure for the invalid "lowering": B/(A\B) $\vdash$ A. A long trip condition can express the required constraint. Roorda (1991) expresses such a condition in terms of the lambda terms that are notational variants of intuitionistic natural deductions corresponding to Lambek proofs, and which provide the semantic dimension of categorial logic. But which condition on axiom linking corresponds to **NL**? How would we express the **L**+**NL** hybrid? And what about the discontinuity calculi? Since these varieties relate more directly to the groupoid prosodic dimension we shall follow Roorda/Moortgat in using prosodic terms to express the correctness conditions.

6.1 Labelled Proof Nets

Roorda (1991) and Moortgat (1991) present unfolding with prosodic labelling as follows.

(50) a. $\dfrac{\gamma{+}a\colon B^{+} \quad a\colon A^{-}}{\gamma\colon B/A^{+}}$ $\qquad$ $\dfrac{a\colon A^{-} \quad a{+}\gamma\colon B^{+}}{\gamma\colon A\backslash B^{+}}$ $\qquad$ a new variable

b. $\dfrac{k\colon A^{+} \quad \gamma{+}k\colon B^{-}}{\gamma\colon B/A^{-}}$ $\qquad$ $\dfrac{k{+}\gamma\colon B^{-} \quad k\colon A^{+}}{\gamma\colon A\backslash B^{-}}$ $\qquad$ k new constant

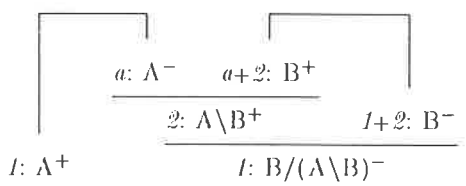
The succedent (negative) unfoldings introduce (Skolem-like) constants. The antecedent (positive) unfoldings introduce variables. Linking identifies the labels of linked atoms and the correctness condition is that a proof structure is a proof net if and only if the set of equations induced by linking is satisfiable. This can be checked by unification.

Consider lifting, for which we assume labelling thus by a constant *1*:

(51) *1*: A $\vdash$ *1*: B/(A\B)

Then there is the proof net (52).

(52)

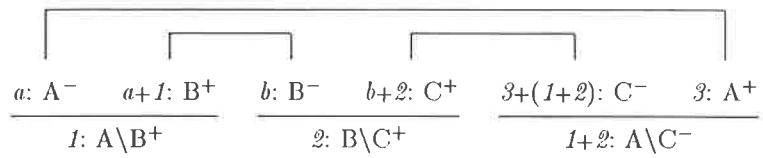


The linking yields the equations *1*=*a* and *a*+*2*= *1*+*2* which are clearly satisfied. For composition

as in (53) we obtain (54).

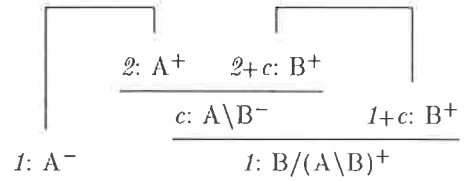
(53) 1: A\B, 2: B\C $\vdash$ $1{+}2$: A\C

(54)



This yields the equations $a{=}3$, $a{+}1{=}b$ and $b{+}2 = 3{+}(1{+}2)$ which are satisfied with $b{=}3{+}1$ in the associative case, but not in the non-associative one. For the invalid lowering however we have:

(55)



The equations $1{=}2$ and $1{+}c{=}2{+}c$ are clearly not satisfiable.

The method is attractive because we can see how it adapts to different residuation calculi, such as the partially associative calculus or the discontinuity calculi, just by unifying prosodic terms according to the laws of the groupoid algebras of interpretation. Such direct association of proof theory with interpretation is precisely the point of labelling: "bringing semantics back into syntax". Generality is obtained because we have identified the highest common factor of sublinear residuation calculi, linear validity, (a lower common factor such as mere intuitionistic validity, for example, would not provide much informative structure). The degrees of variation are in effect built in as non-logical properties of term structure of quantifier-free first-order linear logic: unfolded signed labelled atoms (literals) α: A^+ and β: B^- are atomic formulas of predicational linear logic $A(\alpha)$ and $B(\beta)^\perp$ respectively. Linking corresponds to an application of the resolution principle, together with unification of terms. It is this relation which we shall exploit to resolve the computational shortcoming of the method as it stands: that testing satisfiability of the linking equations appears to demand solution to such problems as semigroup unification, which are quite intractable.

6.2 Clausal Proof Nets

In the course of unfolding no attempt is made to preserve the relations between parts of a formula. Such information, however, can serve to indicate restrictions on possible linkings, and hence guide the instantiation of unknowns.

Observe that the subformulas of a positive (i.e. antecedent) implicational occurrence are allocated to different subproofs, for example with /L in **L**:

(56) $$\frac{\Gamma \vdash A \quad \Delta_1, B, \Delta_2 \vdash C}{\Delta_1, B/A, \Gamma, \Delta_2 \vdash C}/\mathrm{L}$$

This means that neither the A occurrence and the B occurrence, nor any of their subformula occurrences, should ever be connected by an axiom link, for that would correspond to putting them in the same subproof. (The labelling already ensures the eventual failure of any such attempts, but the point is that this information can be used to preempt such failure.) The same is not true of the right rules however; in the following A and B are not allocated to distinct subproofs:

(57) $$\frac{\Gamma, A \vdash B}{\Gamma \vdash B/A}/\mathrm{R}$$

What we shall do is group formulas in the course of unfolding in such a way that attention can be limited to *intergroup* linking because *intragroup* linking is always known to fail. Since positive implications are compiled into linear disjunctions (plus negation) we use the term *clauses* for the groups, and we shall see how their form enables a particular resolution strategy.

The unfolding compilation is now not simply recursive because while positive unfolding steps preserve subformula structure in clause structure, we shall require a negative unfolding step to break out of its clause: the subformulas occur in the same sequent subproof so that the no linking discipline is not to be imposed. We represent a *pseudo-clause*, i.e. one not necessarily free of compound categorial types, by $X, Y, \ldots$ and use parenthetical notation to indicate subparts. The unfolding schemata are the following (ordering of clauses is not required).

(58) a. $$\frac{X(\gamma{+}a\colon B^+ \Cross a\colon A^-)}{X(\gamma\colon B/A^+)} \qquad \frac{X(a{+}\gamma\colon B^+ \Cross a\colon A^-)}{X(\gamma\colon A\backslash B^+)}$$ a new variable

b. $$\frac{X(\gamma{+}k\colon B^-) \quad k\colon A^+}{X(\gamma\colon B/A^-)} \qquad \frac{X(k{+}\gamma\colon B^-) \quad k\colon A^+}{X(\gamma\colon A\backslash B^-)}$$ k new constant

Linking (and unification) is now seen as resolution between clauses:

(59) $$\frac{\alpha\colon A^- \Cross Y \quad \alpha'\colon A^+ \Cross X}{X \Cross Y[\mathrm{MGU}(\alpha, \alpha')]}\ \alpha, \alpha' \text{ unifiable}$$

6.3 Resolution Strategy

The horizontal representations used so far will not accommodate on the page larger linguistic examples, so we re-present the unfolding schemata in a vertical indented style:

(60)
$$\left.\begin{array}{l} X(\gamma\colon B/A^+) \text{ iff} \\ \quad X(\gamma{+}a\colon B^+ \Cross a\colon A^-) \\ X(\gamma\colon A\backslash B^+) \text{ iff} \\ \quad X(a{+}\gamma\colon B^+ \Cross a\colon A^-) \end{array}\right\} a \text{ new variable}$$

$$\left.\begin{array}{l} X(\gamma\colon B/A^-) \text{ iff} \\ \quad X(\gamma{+}k\colon B^-) \\ \quad k\colon A^+ \\ X(\gamma\colon A\backslash B^-) \text{ iff} \\ \quad X(k{+}\gamma\colon B^-) \\ \quad k\colon A^+ \end{array}\right\} k \text{ new constant}$$

Naturally the antecedent and succedent formulas of the target sequent are initially in separate pseudo-clauses, since they pertain to the same goal sequent. Now we note the following (for the implicational fragment): that all the clauses yielded by unfolding a positive (i.e. antecedent) formula contain exactly one positive literal (the head), and zero or more negative literals (the body), i.e. they are *definite clauses*. This can be seen from the fact that the antecedent occurrences start with one occurrence of $^+$ outermost, that the positive unfolding steps transfer a $^+$ to a single formula of its single output pseudo-clause, and that the negative unfolding steps leave intact any $^+$ on one output pseudo-clause, and introduces a $^+$ outermost on the other. We note further that

in all the definite clauses generated by antecedent (positive) formulas, all and only the variables occurring in the negative literals occur in the positive literal: the antecedent formulas are initially labelled by closed terms and so have no variables; the positive unfolding rules put the same variable in positive and negative subcomponents of the same pseudo-clause, and the negative unfolding rules introduce constants.

As for negative formulas, we see that an implicational succedent formula generates positive and negative output pseudo-clauses as it introduces constants. So the target sequent succedent will always unfold to yield one single-literal, i.e. *unit*, clause which is negative and which is ground. In so doing it may also generate antecedent-style definite clauses.

Now, given the restriction to interclausal linking (no incest) we can see that any successful linking must connect the unique negative unit clause generated from the succedent to the head of some definite clause. Since the label of the former is ground, and the label of the latter includes all the variables of its body, any corresponding negative literals in the body of the definite clause become ground on unification, and these in turn must resolve with positive heads, and so on. One or more ground negative literals define the agenda at each stage. Thus all clauses are *Horn clauses*, i.e. clauses with a maximum of one positive literal. We may choose to work breadth-first or depth-first; thus precisely the Prolog search strategy, (depth-first) "linear" input resolution (Chang and Lee 1973), SLD-resolution, is appropriate, but since we are also linear in the sense of occurrence logic, the database of clauses is consumed as we go.

Consider again then lifting:

(61) 1: A $\vdash$ 1: B/(A\B)

To minimize what need be written we write proofs as illustrated in (62). The labelled and signed sequent occurrences are listed down the page, starting with the succedent assignment. Unfolding is performed as shown in (60), which places the negative output pseudo-clause of a negative unfolding above the positive output pseudo-clause, so that working down the page the negative unit clause obtained by unfolding the succedent pseudo-clause is always the first to appear. Indenting shows the course of unfolding, and the fully unfolded clauses obtained are numbered in the left column. One such clause is marked on the right as being the one resolved with the succedent unit negative clause, and after a comma the unifying substitution is shown. The first line after the unfolding of the sequent formulas is the (ground) result of applying this unifier to the body of the asterisked clause. This is the successor agenda. By convention we choose to attempt further resolution with the leftmost literal of this agenda or *goal clause*: we choose a suitable input clause and unify, indicating the input clause and unifier on the right; the instantiated negative literals are added to the front of the agenda, which is written in full on the next line of the derivation, and so on. This gives a depth-first search. Adding the new negative literals to the back of the agenda would give a breadth-first search.

(62)
1: B/(A\B)$^-$
1. $1+2$: B$^-$
   2: A\B$^+$
2. $a+2$: B$^+$ ✠ a: A$^-$ *, $a=1$
3. 1: A$^+$
   1: A$^-$ 3

In (62), after resolving the clause 2 with the initial unit agenda 1 it only remains to match the new agenda with clause 3. For composition the story is a little longer:

(63) 1: A\B, 2: B\C $\vdash$ $1+2$: A\C

(64) $1{+}2$: $A\backslash C^-$
1. $3{+}(1{+}2)$: C^-
2. 3: A^+
   1: $A\backslash B^+$
3. $a{+}1$: $B^+ \Pisymbol{pzd}{58} a$: A^-
   2: $B\backslash C^+$
4. $b{+}2$: $C^+ \Pisymbol{pzd}{58} b$: B^- *, $b{=}3{+}1$
   $3{+}1$: B^- 3, $a{=}3$
   3: A^- 2

7 Linguistic Examples

Linguistic application cannot be explained in a few lines; for motivation see for example Moorgat (1988) and Morrill (1992a). Illustration should be instructive however given even a little familiarity. We consider linguistic examples starting with pure Lambek fragments. There is the following derivation of 'John walks' as a sentence in **L** or **NL**.

(65) 1. *John*+*walks*: S^-
2. *John*: N^+
   walks: $N\backslash S^+$
3. a+*walks*: $S^+ \Pisymbol{pzd}{58} a$: N^- *, $a{=}$*John*
   John: N^- 2

In the derivation of 'John likes Bill' the transitive verb gives rise to an agenda of length two. Assuming an associative context, parentheses are ommitted from the prosodic terms.

(66) 1. *John*+*likes*+*Bill*: S^-
2. *John*: N^+
   likes: $(N\backslash S)/N^+$
   likes+a: $N\backslash S^+ \Pisymbol{pzd}{58} a$: N^-
3. b+*likes*+a: $S^+ \Pisymbol{pzd}{58} b$: $N^- \Pisymbol{pzd}{58} a$: N^- *$b{=}$*John*, $a{=}$*Bill*
4. *Bill*: N^+
   John: $N^- \Pisymbol{pzd}{58}$ *Bill*: N^- 2
   Bill: N^- 4

The next example illustrates the effect for an auxiliary verb treated as a functor over a verb phrase, which is itself a functor. The auxiliary creates an antecedent literal at line 4 labelled by a Skolem constant, and this resolves with the subject literal of the embedded verb phrase.

(67) 1. *John*+*will*+*walk*: S^-
2. *John*: N^+
   will: $(N\backslash S)/(N\backslash S)^+$
   will+a: $N\backslash S^+ \Pisymbol{pzd}{58} a$: $N\backslash S^-$
   b+*will*+a: $S^+ \Pisymbol{pzd}{58} b$: $N^- \Pisymbol{pzd}{58} a$: $N\backslash S^-$
3. b+*will*+a: $S^+ \Pisymbol{pzd}{58} b$: $N^- \Pisymbol{pzd}{58} 1{+}a$: S^- *, $b{=}$*John*, $a{=}$*walk*
4. 1: N^+
   walk: $N\backslash S^+$
5. c+*walk*: $S^+ \Pisymbol{pzd}{58} c$: N^-
   John: $N^- \Pisymbol{pzd}{58} 1{+}$*walk*: S^- 2
   $1{+}$*walk*: S^- 5, $c{=}1$
   1: N^- 4

For the following minimal example of object relativisation associativity is essential. The relative pronoun is a higher order functor and the positive antecedent literal unfolded from its argument

corresponds to an "empty category" or "trace" of the extraction. But note that such a literal arose in the non-extraction example (67) also.

(68)

1.	*which+John+likes*: R^-	
	which: $R/(S/N)^+$	
	which+a: $R^+ \Cup a$: S/N^-	
2.	*which+a*: $R^+ \Cup a+1$: S^-	*, *a=John+likes*
3.	*1*: N^+	
4.	*John*: N^+	
	likes: $(N\backslash S)/N^+$	
	likes+b: $N\backslash S^+ \Cup b$: N^-	
5.	*c+likes+b*: $S^+ \Cup c$: $N^- \Cup b$: N^-	
	John+likes+1: S^-	5, *c=John*, *b=1*
	John: $N^- \Cup 1$: N^-	4
	1: N^-	3

7.1 Multimodal Unfolding

For multimodal calculi in general unfolding proceeds in the same way:

(69) a. $\dfrac{X(\gamma{+_i}a\colon B^+ \Cup a\colon A^-)}{X(\gamma\colon B/_iA^+)}$ $\quad$ $\dfrac{X(a{+_i}\gamma\colon B^+ \Cup a\colon A^-)}{X(\gamma\colon A\backslash_iB^+)}$ $\quad$ a new variable

b. $\dfrac{X(\gamma{+_i}k\colon B^-)(\otimes)k\colon A^+}{X(\gamma\colon B/_iA^-)}$ $\quad$ $\dfrac{X(k{+_i}\gamma\colon B^-)(\otimes)k\colon A^+}{X(\gamma\colon A\backslash_iB^-)}$ $\quad$ k new constant

7.2 Partially Associative Calculus

Assignment of a coordinator to (S>S)/S characterises a coordinate structure as a non-associative domain. For example in the following the complementised sentence 'that it rains and it shines' is analysed as containing a domain ['it rains and it shines'] composed of subconstituents 'it rains' and 'and it shines' (the latter is itself unstructured).

(70)

1.	*that+(it+rains, and+it+shines)*: CP^-	
	that: CP/S^+	
2.	*that+a*: $CP^+ \Cup a$: S^-	*, *a=(it+rains, and+it+shines)*
3.	*it+rains*: S^+	
	and: $(S>S)/S^+$	
	and+b: $S>S^+ \Cup b$: S^-	
4.	*(c, and+b)*: $S^+ \Cup c$: $S^- \Cup b$: S^-	
5.	*it+shines*: S^+	
	(it+rains, and+it+shines): S^-	4, *c=it+rains*, *b=it+shines*
	it+rains: $S^- \Cup$ *it+shines*: S^-	3
	it+shines: S^-	5

7.3 Regular Discontinuity

We have seen earlier that the regular discontinuity calculus is interpreted in a prosodic algebra $\langle L, +, (.,.), W, \epsilon\rangle$ (we shall neglect the issue whereby ϵ is not wanted in any type) for which in addition to the associativity of + we have $s{+}\epsilon = \epsilon{+}s = s$ and the split-wrap equation $(s_1, s_3)Ws_2 = s_1{+}s_2{+}s_3$.

The techniques given here show how proof net theorem-proving in implicational residuation calculi, and hence parsing in certain categorial logics, can be compiled into a form suitable for SLD-resolution in linear logic. This indicates a general strategy for checking the correctness of a proof net by unification in which one term is always ground, but leaves open the problem of computing unifiers in any particular case.

For the regular discontinuity calculus the task concerns us with the following equations:

(71) a. $s_1{+}(s_2{+}s_3) = (s_1{+}s_2){+}s_3$
b. $s{+}\epsilon = \epsilon{+}s = s$
c. $(s_1, s_3)Ws_2 = s_1{+}s_2{+}s_3$

The unification procedure is to compute, for a given ground term α, all the distinct assignments σ of ground terms to variables in another term α' such that $\alpha = \alpha'[\sigma]$. We treat the process in two stages; (71b) and (71c) are considered normalisation rules (with redex on the left and contractum on the right); associativity as in (71a) could be naturally treated by representing equivalence classes of associative bracketings as lists, though we shall not choose to do so.

In the first phase, the ground term α is normalised by transforming redexes to their contractums. The second stage proceeds by recursion on the structure of α'. There are the cases that α' is a variable, a constant, or has principle operator one of the three prosodic adjunctions. Finally there are the cases that α' is, or can be instantiated to, a redex.

If α' is a variable v then simply put $v = \alpha$. If α' is a constant k then if α is k succeed, otherwise fail. If α' is of the form (α'_1, α'_2) then if α is of the form (α_1, α_2) unify α_1 and α'_1 and α_2 and α'_2. If α' is of the form $\alpha'_1 W \alpha'_2$ then if α is of the form $\alpha_1 W \alpha_2$ unify α_1 and α'_1 and α_2 and α'_2. If α' is of the form $\alpha'_1{+}\alpha'_2$ then find representatives α_1 and α_2 satisfying $\alpha = \alpha_1{+}\alpha_2$ (using associativity) and unify α_1 and α'_1, and α_2 and α'_2.

It remains to consider the cases where α' has the form of, or can be instantiated to the form of, a redex (the redexes in α having been already removed in the first phase). If α' can be instantiated to $\beta{+}\epsilon$ unify α and β. If α' can be instantiated to $\epsilon{+}\beta$ unify α and β. If α' can be instantiated to $(\alpha'_1, \alpha'_3)W\alpha'_2$ then find representatives α_1, α_2 and α_3 satisfying $\alpha = \alpha_1{+}\alpha_2{+}\alpha_3$ and unify α_1 with α'_1, α_2 with α'_2, and α_3 with α'_3.

As seen earlier a simple instance of discontinuity is obtained by assigning the compound particle verb prosodic form $(rang, up)$ to a wrapping transitive verb type $(\mathrm{N}\backslash\mathrm{S})\uparrow\mathrm{N}$. At line 3 of the following derivation $Mary{+}rang{+}John{+}up$ is unified with $b{+}((rang, up)Wa)$ by the unifier $\{b = Mary, a = John\}$.

(72)
1. $Mary{+}rang{+}John{+}up$: S^-
2. $Mary$: N^+
   $(rang, up)$: $(\mathrm{N}\backslash\mathrm{S})\uparrow\mathrm{N}^+$
   $((rang, up)Wa)$: $\mathrm{N}\backslash\mathrm{S}^+ \pitchfork a$: N^-
3. $b{+}((rang, up)Wa)$: $\mathrm{S}^+ \pitchfork b$: $\mathrm{N}^- \pitchfork a$: N^- *. $b{=}Mary$, $a{=}John$
4. $John$: N^+
   $Mary$: $\mathrm{N}^- \pitchfork John$: N^- 2
   $John$: N^- 4

At line 4 of the following derivation $John{+}likes{+}everyone$ is unified with $cW everyone$ by $c = (John{+}likes, \epsilon)$, and $(John{+}likes, \epsilon)W1$ is subsequently unified with $b{+}likes{+}a$ by $\{b = John, a =$

1. *talked+to+Mary+about+herself*: VP^-
   talked: $(VP/PP)/PP^+$
      talked+a: $VP/PP^+ \Cross a: PP^-$
2. *talked+a+b*: $VP/PP^+ \Cross b: PP^- \Cross a: PP^-$
   to: PP/N^+
3. *to+c*: $PP^+ \Cross c: N^-$
4. *Mary*: N^+
   about: PP/N^+
5. *about+d*: $PP^+ \Cross d: N^-$
   herself: $(PP\backslash N)\backslash(((VP/PP)/N)>(VP\backslash N))^+$
      $(c\mathit{Wherself})$: $((VP/PP)/N)>(VP\backslash N)^+ \Cross e: PP\backslash N^-$
         $(f, (c\mathit{Wherself}))$: $VP\backslash N^+ \Cross f: (VP/PP)/N^- \Cross e: PP\backslash N^-$
            $((f, (c\mathit{Wherself}))\, Wg)$: $VP^+ \Cross g: N^- \Cross f: (VP/PP)/N^- \Cross e: PP\backslash N^-$
               $((f, (c\mathit{Wherself}))\, Wg)$: $VP^+ \Cross g: N^- \Cross f{+}1: VP/PP \Cross N^- \Cross e: PP\backslash N^-$
                  $((f, (c\mathit{Wherself}))\, Wg)$: $VP^+ \Cross g: N^- \Cross f{+}1{+}2: VP \Cross N^- \Cross e: PP\backslash N^-$
6. $((f, (c\mathit{Wherself}))\, Wg)$: $VP^+ \Cross g: N^- \Cross f{+}1{+}2: VP \Cross N^- \Cross (c\mathit{W3}): PP^-$
*, $f{=}talked{+}to$, $g{=}Mary$, $e{=}(about, \epsilon)$
7. *3*: N^+
8. *2*: PP^+
9. *1*: N^+

Mary: $N^- \Cross talked{+}to{+}1{+}2$: $VP^- \Cross ((about, \epsilon)\, W3)$: PP^-	4
talked+to+1+2: $VP^- \Cross ((about, \epsilon)\, W3)$: PP^-	2, $a{=}to{+}1$, $b{=}2$
2: $PP^- \Cross to{+}1$: $PP^- \Cross ((about, \epsilon)\, W3)$: PP^-	8
to+1: $PP^- \Cross ((about, \epsilon)\, W3)$: PP^-	3, $c{=}1$
1: $N^- \Cross ((about, \epsilon)\, W3)$: PP^-	9
$((about, \epsilon)\, W3)$: PP^-	5, $d{=}3$
3: N^-	7

Figure 1: Non-c-commanding object-antecedent reflexivisation

1}.

(73)
1. *John+likes+everyone*: S^-
2. *John*: N^+
   likes: $(N\backslash S)/N^+$
      likes+a: $N\backslash S^+ \Cross a: N^-$
3. *b+likes+a*: $S^+ \Cross b: N^- \Cross a: N^-$
   everyone: $(S\uparrow N)\downarrow S^+$
      $(c\mathit{Weveryone})$: $S^+ \Cross c: S\downarrow N^-$
4. $(c\mathit{Weveryone})$: $S^+ \Cross (c\mathit{W1})$: S^- — *, $c{=}(John{+}likes, \epsilon)$
5. *1*: N^+
   $(John{+}likes, \epsilon)\, W1$: S^- — 3, $b{=}John$, $a{=}1$
   John: $N^- \Cross 1$: N^- — 2
   1: N^- — 5

Our final example, in Figure 1, shows non-c-commanding object-antecedent reflexivisation.

7.4 Head-Oriented Discontinuity

The same methods are applicable to head-oriented discontinuity, for which we also require bracket unfolding:

(74) $$\frac{[\alpha]^{-1}: A^+}{\alpha: [\,]A^+} \qquad \frac{[\alpha]: A^+}{\alpha: [\,]^{-1}A^+}$$

(75) $$\frac{[\alpha]^{-1}: A^-}{\alpha: [\,]A^-} \qquad \frac{[\alpha]: A^-}{\alpha: [\,]^{-1}A^-}$$

Recall the appropriate label equations:

(76) $(\alpha +_r \beta) +_h \gamma = \alpha +_r (\beta +_h \gamma)$
$(\alpha +_l \beta) +_h \gamma = (\alpha +_h \beta) +_l \gamma$

(77) $[_v \alpha] +_h \beta = [_v \beta +_l \alpha]$

Then the earlier verb-infixing example is derived as follows by clausal resolution:

(78)
1. *boeken*$+_r[_v$*kan*$+_l$*lezen*$]$: VP^-
2. *boeken*: N^+
   lezen: $[v]^{-1}(N\backslash_r IVi)^+$
   $[_v$*lezen*$]$: $N\backslash_r IVi^+$
3. $a +_r [_v$*lezen*$]$: $IVi^+ \Cap a$: N^-
   kan: $IVi \downarrow VP^+$
4. $b +_h$ *kan*: $VP^+ \Cap b$: IVi^- — *, $b =$ *boeken*$+_r[_v$*lezen*$]$
   boeken$+_r[_v$*lezen*$]$: IVi^- — 3, $a =$ *boeken*
   boeken: N^-

The only non-trivial step is that resolving the unit goal clause with the head of clause 4, for which the unification is explicated by the following:

(79) *boeken*$+_r[_v$*kan*$+_l$*lezen*$]$ = *boeken*$+_r([_v$*lezen*$]+_h$*kan*$)$ = $(boeken +_r [_v lezen]) +_h kan$

Bibliography

Bibal, W.: 1981, 'On matrices with connections', *Journal of the Association for Computing Machinery* **28**, 633–645.

Buszkowski, K.: 1986, 'Competeness results for Lambek syntactic calculus', *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **32**, 13–28.

Chang, C-L. and R.C-T. Lee: 1973, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.

Došen, Kosta: 1992, 'A Brief Survey of Frames for the Lambek Calculus', *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **38**, 179–187.

Došen, Kosta and Peter Schroeder-Heister (eds.): 1993, *Substructural Logics*, Oxford University Press, Oxford.

Gabbay, D.: 1991, *Labelled Deductive Systems*, to appear, Oxford University Press, Oxford.

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

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

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.

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

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

Morrill, Glyn: 1990, 'Rules and Derivations: Binding Phenomena and Coordination in Categorial Logic', in Deliverable R1.2.D of DYANA Dynamic Interpretation of Natural Language, ESPRIT Basic Research Action BR3175.

Morrill, Glyn: 1992a, *Type Logical Grammar*, Report de Recerca LSI–92–5–R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, and OTS Working Paper OTS–WP–CL–92–002, Rijksuniversiteit Utrecht.

Morrill, Glyn: 1992b, '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.

Oehrle, Richard T. and Shi Zhang: 1989, 'Lambek Calculus and Preposing of Embedded Subjects', *Chicago Linguistic Society* **25**, Chicago.

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

Wallen, L.A.: 1990, *Automated proof search in non-classical logics: efficient matrix proof methods for modal and intuitionistic logics*, MIT Press.

Appendix: Implementation of Discontinuity Calculi

Program Listing

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

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

:- op(450, xfx, -).

/*
Operators over, under, to, from, wrap and infix
*/

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

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

:- op(400, xfx, >). % also right-headed prosodic adjunction
:- op(400, xfx, <). % also left-headed prosodic adjunction

:- op(400, xfx, h). % head adjunction

% Headed operators

:- op(400, xfx, ol). % over left
:- op(400, xfx, ul). % under left

:- op(400, xfx, or). % over right
:- op(400, xfx, ur). % under right

:- op(400, xfx, hw). % head wrap
:- op(400, xfx, hi). % head infix
```

```
% Bracket operators

:- op(300, fx, v).
:- op(300, fx, a).

/*
Prosodic surface adjunctions + and wrapping adjuction 'W'; splitting adjunction will be represented [., .]
*/

:- op(300, yfx, +).
:- op(300, xfx, 'W').

% Lexical assignments

about   - about
        := pp/n.
and     - [lmd, X, [lmd, Y, [and, Y, X]]]
        := (s>s)/s.
believes- believe
        := (n\s)/s.
bill    - b
        := n.
book    - book
        := 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.
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.
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.

boeken  -  books
        := n.
gooien  -  throw
        := a(pp ur (n ur ivi)).
gooien  -  throw
        := pp ur a(n ur ivi).
het     -  it
        := n.
jan     -  j
        := n.
kan     -  can
        := ivi hi (n ur s).
kan     -  [lmd, X, [app, X, can]]
        := q ol (s hw (ivi hi(n ur s))).
kunnen  -  can
        := ivi hi ivi.
leest   -  [lmd, X, [app, X, read]]
        := q ol (s hw a(n ur (n ur s))).
lezen   -  read
        := a (n ur ivi).
weg     -  away
        := pp.
willen  -  want
        := ivi hi ivi.
wil     -  want
        := ivi hi (n ur s).
zal     -  shall
        := ivi hi (n ur s).
zal     -  [lmd, X, [app, X, shall]]
        := q ol (s hw (ivi hi(n ur s))).

% reset resets the gensymb record to 1

reset :-
        clear,
        assert(rec(1)), !.

% clear removes any gensymb records
```

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

clear :- !.

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

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

/*
lex(+PformIn, -Xs, -PformOut) means that Xs is a list of lexical clauses obtained by lexical lookup
on PformIn. PformOut is the result of tokenising PformIn according to the tokens used in Xs.
*/

lex(PformIn, Xs, PformOut) :-
        copy(PformIn, K, PformOut1, N), !,
        lex1(PformOut1, K, N, Xs, PformOut).

lex(Pform, [], Pform).

/*
copy(+Pform, ?K, -Pform1, ?N) copies prosodic form Pform to Pform1, replacing the first word
(atom) unifiable with K by N; not resatisfiable; fails if there is no such word
*/

copy(Alpha+Beta, K, Alpha1+Beta, N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(Alpha+Beta, K, Alpha+Beta1, N) :-
        copy(Beta, K, Beta1, N), !.

copy(Alpha'W'Beta, K, Alpha1'W'Beta, N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(Alpha'W'Beta, K, Alpha'W'Beta1, N) :-
        copy(Beta, K, Beta1, N), !.

copy([Alpha,Beta], K, [Alpha1,Beta], N) :-
        copy(Alpha, K, Alpha1, N), !.

copy([Alpha,Beta], K, [Alpha,Beta1], N) :-
        copy(Beta, K, Beta1, N), !.

copy(K, K, N, N) :-
        atom(K), !.

copy(Alpha>Beta, K, Alpha1>Beta, N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(Alpha>Beta, K, Alpha>Beta1, N) :-
        copy(Beta, K, Beta1, N), !.

copy(Alpha<Beta, K, Alpha1<Beta, N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(Alpha<Beta, K, Alpha<Beta1, N) :-
        copy(Beta, K, Beta1, N), !.

copy(Alpha h Beta, K, Alpha1 h Beta, N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(Alpha h Beta, K, Alpha h Beta1, N) :-
        copy(Beta, K, Beta1, N), !.
```

```
copy(b(Alpha), K, b(Alpha1), N) :-
        copy(Alpha, K, Alpha1, N), !.

copy(u(Alpha), K, u(Alpha1), N) :-
        copy(Alpha, K, Alpha1, N), !.

/*
lex1(+PformOut1, +K, +N, -XXs, -PformOut) looks up lexical assignments containing an occurrence of
the word K, and checks if the other words in the lexical entry also occur in PformOut1; lexical lookup
is then continued; XXs is the list of tokenised lexical clauses resulting obtained from lexical assignments;
PformOut is the result of tokenising PformOut1 correspondingly, with the first occurrence of word K
having been replaced by the result of tokenising N.
*/

lex1(PformOut1, K, N, [cls([a(AlphaT-Phi:A)])|Xs], PformOut) :-
        Alpha-Phi := A,
        copy(Alpha, K, Alpha1, N),
        gensymb(N),
        do_all(Alpha1, PformOut1, AlphaT, PformOutT),
        lex(PformOutT, Xs, PformOut).

/*
do_all(+Alpha1, +PformOut1, -AlphaT, -PformOutT) checks that each word occurrence in the prosodic form
Alpha1 has a corresponding occurrence in PformOut1, tokenising in the process to AlphaT and PformOutT
*/

do_all(Alpha1, PformOut1, AlphaT, PformOutT) :-
        copy(Alpha1, K, Alpha2, N),
        copy(PformOut1, K, PformOut2, N),
        gensymb(N),
        do_all(Alpha2, PformOut2, AlphaT, PformOutT), !.

do_all(Alpha, Pform, Alpha, Pform) :- !.

/*
unfold(+InCls, -OutClss) makes one unfolding step of the (pseudo-)
clause InCls, to give the list of output (pseudo-) clauses OutClss
*/

unfold(cls([a(Gamma-Chi:B/A)|L]),
    [cls([a(Gamma+Alpha-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A\B)|L]),
    [cls([a(Alpha+Gamma-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B/A)|L2], L),
        append(L1, [s(Gamma+K-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A\B)|L2], L),
        append(L1, [s(K+Gamma-Psi:B)|L2], Ln),
        gensymb(K).

unfold(cls([a(Gamma-Chi:B<A)|L]),
    [cls([a([Gamma, Alpha]-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A>B)|L]),
    [cls([a([Alpha, Gamma]-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B<A)|L2], L),
        append(L1, [s([Gamma, K]-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A>B)|L2], L),
        append(L1, [s([K, Gamma]-Psi:B)|L2], Ln),
        gensymb(K).
```

```
unfold(cls([a(Gamma-Chi:B wr A)|L]),
    [cls([a(Gamma'W'Alpha-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A in B)|L]),
    [cls([a(Alpha'W'Gamma-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B wr A)|L2], L),
        append(L1, [s(Gamma'W'K-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A in B)|L2], L),
        append(L1, [s(K'W'Gamma-Psi:B)|L2], Ln),
        gensymb(K).

unfold(cls([a(Gamma-Chi:B or A)|L]),
    [cls([a(Gamma>Alpha-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A ur B)|L]),
    [cls([a(Alpha>Gamma-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B or A)|L2], L),
        append(L1, [s(Gamma>K-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A ur B)|L2], L),
        append(L1, [s(K>Gamma-Psi:B)|L2], Ln),
        gensymb(K).

unfold(cls([a(Gamma-Chi:B ol A)|L]),
    [cls([a(Gamma<Alpha-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A ul B)|L]),
    [cls([a(Alpha<Gamma-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B ol A)|L2], L),
        append(L1, [s(Gamma<K-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A ul B)|L2], L),
        append(L1, [s(K<Gamma-Psi:B)|L2], Ln),
        gensymb(K).

unfold(cls([a(Gamma-Chi:B hw A)|L]),
    [cls([a(Gamma h Alpha-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).
unfold(cls([a(Gamma-Chi:A hi B)|L]),
    [cls([a(Alpha h Gamma-[app,Chi,Phi]:B), s(Alpha-Phi:A)|L])]).

unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:B hw A)|L2], L),
        append(L1, [s(Gamma h K-Psi:B)|L2], Ln),
        gensymb(K).
unfold(cls(L), [cls(Ln), cls([a(K-X:A)])]) :-
        append(L1, [s(Gamma-[lmd,X,Psi]:A hi B)|L2], L),
        append(L1, [s(K h Gamma-Psi:B)|L2], Ln),
        gensymb(K).

unfold(cls(L), [cls(Ln)]) :-
        append(L1, [a(Alpha-Phi: a A)|L2], L),
        append(L1, [a(b(Alpha)-Phi: A)|L2], Ln).
unfold(cls(L), [cls(Ln)]) :-
        append(L1, [s(Alpha-Phi: a A)|L2], L),
        append(L1, [s(b(Alpha)-Phi: A)|L2], Ln).
unfold(cls(L), [cls(Ln)]) :-
        append(L1, [a(Alpha-Phi: v A)|L2], L),
        append(L1, [a(u(Alpha)-Phi: A)|L2], Ln).
unfold(cls(L), [cls(Ln)]) :-
```

```
        append(L1, [s(Alpha-Phi: v A)|L2], L),
        append(L1, [s(u(Alpha)-Phi: A)|L2], Ln).

/*
unf(+X, -U) means that U is the list of clauses resulting from
completely unfolding the pseudo-clause X
*/

unf(X, U) :-
        unfold(X, XL), !,
        unf_l(XL, U).

unf(X, [X]).

/*
unf_l(Xs, U) means that U is the list of clauses resulting from
completely unfolding the list of pseudo-clauses Xs
*/

unf_l([], []).

unf_l([X|Xs], YsZs) :-
        unf(X, Ys),
        unf_l(Xs, Zs),
        append(Ys, Zs, YsZs).

/*
match(+SClss, +AClss) means that the succedent clauses SClss can
be matched off against the antecedent clauses AClss
*/

match([], []).

match([s(Alpha-Phi:A)|Gs], L) :-
        append(L1, [cls([a(Alpha1-Phi:A)|Ds])|L2], L),
        eq1(Alpha, Alpha1),
        append(Ds, Gs, DsGs),
        append(L1, L2, L1L2),
        match(DsGs, L1L2).

/*
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).

/*
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, Gamma]'W'Beta, Alpha+Beta+Gamma).

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(Alpha'W'Beta, AlphaN'W'Beta) :-
        pcontract(Alpha, AlphaN).

pcontract(Alpha'W'Beta, Alpha'W'BetaN) :-
        pcontract(Beta, BetaN).

pcontract([Alpha, Beta], [AlphaN, Beta]) :-
        pcontract(Alpha, AlphaN).

pcontract([Alpha, Beta], [Alpha, BetaN]) :-
        pcontract(Beta, BetaN).

pcontract(Alpha>0, Alpha).

pcontract(0>Alpha, Alpha).

pcontract(Alpha<0, Alpha).

pcontract(0<Alpha, Alpha).

pcontract((Alpha>Beta) h Gamma, Alpha>(Beta h Gamma)).

pcontract((Alpha<Beta) h Gamma, (Alpha h Gamma)<Beta).

pcontract(b(Alpha) h Beta, b(Beta<Alpha)).

pcontract(b(u(Alpha)), Alpha).

pcontract(u(b(Alpha)), Alpha).

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

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

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

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

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

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

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

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

/*
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([Alpha, Beta], [Gamma, Delta]) :-
        eq(Alpha, Gamma),
        eq(Beta, Delta).

eq(Alpha'W'Beta, Gamma'W'Delta) :-
        eq(Alpha, Gamma),
        eq(Beta, Delta).

eq(Delta, [Alpha,Beta]'W'Gamma) :-
        eq(Delta,Alpha1+Gamma1+Beta1),
        eq(Alpha1, Alpha),
        eq(Beta1, Beta),
        eq(Gamma1, Gamma).

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).

eq(Alpha>Beta, Gamma>Delta) :-
        eq(Alpha, Gamma),
        eq(Beta, Delta).

eq(Alpha<Beta, Gamma<Delta) :-
        eq(Alpha, Gamma),
        eq(Beta, Delta).

eq(Alpha h Beta, Gamma h Delta) :-
        eq(Alpha, Gamma),
        eq(Beta, Delta).

eq(b(Alpha), b(Alpha1)) :-
        eq(Alpha, Alpha1).

eq(u(Alpha), u(Alpha1)) :-
        eq(Alpha, Alpha1).

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

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

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

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

eq(Alpha1>Delta, (Alpha>Beta)h Gamma) :-
        eq(Delta, Beta1 h Gamma1),
        eq(Alpha1, Alpha),
        eq(Beta1, Beta),
        eq(Gamma1, Gamma).
```

```
eq(Delta<Beta1, (Alpha<Beta)h Gamma) :-
        eq(Delta, Alpha1 h Gamma1),
        eq(Alpha1, Alpha),
        eq(Beta1, Beta),
        eq(Gamma1, Gamma).

eq(Delta, b(Alpha) h Beta) :-
        eq(Delta, b(Beta1<Alpha1)),
        eq(Alpha1, Alpha),
        eq(Beta1, Beta).

eq(Delta, b(u(Alpha))) :-
        eq(Delta, Alpha).
eq(Delta, u(b(Alpha))) :-
        eq(Delta, Alpha).

/*
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).

/*
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([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 prosodic forms

% Simple sentences

pf(1, john+walks, s).
pf(2, john+likes+mary, s).
pf(3, john+seeks+mary, s).
pf(4, mary+shows+the+woman+the+book, s).

% Discontinuous functors

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

% Relativisation

pf(9, the+man+that+mary+likes+walks, s).
pf(10, the+man+that+john+thinks+mary+likes+walks, s).
pf(11, the+man+that+john+thinks+bill+believes+mary+likes+walks, s).

% Coordinate Structure Constraint

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

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

pf(14, john+thinks+bill+or+mary+walks, s).

% Subject-antecedent reflexivisation

pf(15, john+likes+himself, s).
pf(16, john+votes+for+himself, s).
pf(17, john+shows+himself+the+book, s).

% Object antecedent reflexivisation

pf(18, john+shows+mary+herself, s).
pf(19, john+shows+herself+mary, s).

% Pied-Piping object antecedent reflexivisation

pf(20, john+shows+mary+the+picture+of+herself, s).

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

pf(21, john+talks+to+mary+about+herself, s).

% Quantification
```

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

% Pied-Piping

pf(30, the+man+whom+john+likes+the+picture+of, n).
pf(31, the+man+the+picture+of+whom+john+likes, n).
pf(32, the+man+whose+book+john+likes+the+picture+of, n).
pf(33, the+man+the+picture+of+whose+book+john+likes, n).

pf(d(1), boeken>b(kan<lezen), n ur s).
pf(d(2), boeken>b(wil<(kunnen<lezen)), n ur s).
pf(d(3), het>(weg>b(zal<gooien)), n ur s).
pf(d(4), het>b(zal<(weg>gooien)), n ur s).
pf(d(5), het>(weg>b(zal<(willen<gooien))), n ur s).
pf(d(6), het>b(zal<(willen<(weg>gooien))), n ur s).
pf(d(7), het>b(weg>(willen<gooien)), n ur s).
pf(d(8), het>(weg>b(willen<gooien)), n ur s).
pf(d(9), het>(weg>(willen<b(gooien))), n ur s).
pf(d(10), leest<(jan>(boeken>b(0))), q).
pf(d(11), kan<(jan>(boeken>b(lezen))), q).
pf(d(12), zal<(jan>(boeken>b(kunnen<lezen))), q).
pf(d(13), zal<(jan>(het>b(willen<(weg>gooien)))), q).

% test(?N) tests parsing from prosodic form N

test(N) :-
        pf(N, Alpha, A),
        reset,
        nl, nl, write(N), write('. '), write(Alpha: A),
        lex(Alpha, M, TokenAlpha),
        unf_l([cls([s(TokenAlpha-Phi:A)])|M], [cls(0)|01]),
        match(0, 01),
        eval(Phi, NF), nl, nl, write(NF), fail.
```

Log

```
?- test(_).


1. john+walks:s

[app,walk,j]

2. john+likes+mary:s

[app,[app,like,m],j]

3. john+seeks+mary:s

[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

[app,[app,[app,show,[iota,$VAR(2),[app,woman,$VAR(2)]]],[iota,$VAR(0),[app,book,$VAR(0)
]]],m]

5. john+rings+mary+up:s
```

```
[app,[app,phone,m],j]

6. john+gives+mary+the+cold+shoulder:s

[app,[app,shun,m],j]

7. either+john+walks+or+mary+sings:s

[or,[app,walk,j],[app,sing,m]]

8. john+neither+walks+nor+sings:s

[not,[or,[app,walk,j],[app,sing,j]]]

9. the+man+that+mary+likes+walks:s

[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

[app,walk,[iota,$VAR(4),[app,[app,and,[app,man,$VAR(4)]],[app,[app,think,[app,[app,like,$VA
R(4)],m]],j]]]]

11. the+man+that+john+thinks+bill+believes+mary+likes+walks:s

[app,walk,[iota,$VAR(4),[app,[app,and,[app,man,$VAR(4)]],[app,[app,think,[app,[app,believe,[a
pp,[app,like,$VAR(4)],m]],b]],j]]]]

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

[and,[app,walk,j],[app,sing,m]]

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

14. john+thinks+bill+or+mary+walks:s

[app,[app,think,[or,[app,walk,b],[app,walk,m]]],j]

[or,[app,[app,think,[app,walk,b]],j],[app,[app,think,[app,walk,m]],j]]

15. john+likes+himself:s

[app,[app,like,j],j]

16. john+votes+for+himself:s

[app,[app,vote,j],j]

17. john+shows+himself+the+book:s

[app,[app,[app,show,j],[iota,$VAR(1),[app,book,$VAR(1)]]],j]

18. john+shows+mary+herself:s

[app,[app,[app,show,m],m],j]

19. john+shows+herself+mary:s

20. john+shows+mary+the+picture+of+herself:s

[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

[app,[app,[app,talk,[app,to,m]],[app,about,m]],j]

22. someone+walks:s
```

```
[xst,$VAR(1),[app,walk,$VAR(1)]]

23. some+man+walks:s

[xst,$VAR(1),[and,[app,man,$VAR(1)],[app,walk,$VAR(1)]]]

24. everyone+likes+someone:s

[all,$VAR(4),[xst,$VAR(2),[app,[app,like,$VAR(2)],$VAR(4)]]]

[xst,$VAR(4),[all,$VAR(2),[app,[app,like,$VAR(4)],$VAR(2)]]]

25. john+seeks+someone:s

[app,[app,seek,[lmd,$VAR(2),[lmd,$VAR(0),[xst,$VAR(3),[app,[app,$VAR(2),$VAR(3)],$VAR
(0)]]]]],j]

[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

[all,$VAR(6),[app,[app,seek,[lmd,$VAR(3),[lmd,$VAR(1),[xst,$VAR(4),[app,[app,$VAR(3),$V
AR(4)],$VAR(1)]]]]],$VAR(6)]]

[all,$VAR(6),[xst,$VAR(4),[app,[app,seek,[lmd,$VAR(3),[lmd,$VAR(1),[app,[app,$VAR(3),$V
AR(4)],$VAR(1)]]]],$VAR(6)]]]

[xst,$VAR(6),[all,$VAR(4),[app,[app,seek,[lmd,$VAR(3),[lmd,$VAR(1),[app,[app,$VAR(3),$V
AR(6)],$VAR(1)]]]],$VAR(4)]]]

27. bill+thinks+someone+walks:s

[app,[app,think,[xst,$VAR(1),[app,walk,$VAR(1)]]],b]

[xst,$VAR(1),[app,[app,think,[app,walk,$VAR(1)]],b]]

28. bill+thinks+some+man+shows+everyone+john:s

[app,[app,think,[xst,$VAR(4),[and,[app,man,$VAR(4)],[all,$VAR(2),[app,[app,[app,show,$VAR
(2)],j],$VAR(4)]]]]],b]

[app,[app,think,[all,$VAR(5),[xst,$VAR(2),[and,[app,man,$VAR(2)],[app,[app,[app,show,$VAR
(5)],j],$VAR(2)]]]]],b]

[xst,$VAR(4),[and,[app,man,$VAR(4)],[app,[app,think,[all,$VAR(2),[app,[app,[app,show,$VAR
(2)],j],$VAR(4)]]],b]]]

[xst,$VAR(4),[and,[app,man,$VAR(4)],[all,$VAR(2),[app,[app,think,[app,[app,[app,show,$VAR
(2)],j],$VAR(4)]],b]]]]

[all,$VAR(5),[app,[app,think,[xst,$VAR(2),[and,[app,man,$VAR(2)],[app,[app,[app,show,$VAR
(5)],j],$VAR(2)]]]],b]]

[all,$VAR(5),[xst,$VAR(2),[and,[app,man,$VAR(2)],[app,[app,think,[app,[app,[app,show,$VAR
(5)],j],$VAR(2)]],b]]]]

29. the+book+that+john+shows+everyone:n

[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

[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

[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

[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

[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]]]

d(1). boeken>b(kan<lezen):n ur s

[lmd,$VAR(0),[app,[app,can,[app,read,books]],$VAR(0)]]

d(2). boeken>b(wil< (kunnen<lezen)):n ur s

[lmd,$VAR(0),[app,[app,want,[app,can,[app,read,books]]],$VAR(0)]]

d(3). het> (weg>b(zal<gooien)):n ur s

[lmd,$VAR(0),[app,[app,shall,[app,[app,throw,away],it]],$VAR(0)]]

d(4). het>b(zal< (weg>gooien)):n ur s

[lmd,$VAR(0),[app,[app,shall,[app,[app,throw,away],it]],$VAR(0)]]

d(5). het> (weg>b(zal< (willen<gooien))):n ur s

[lmd,$VAR(0),[app,[app,shall,[app,want,[app,[app,throw,away],it]]],$VAR(0)]]

d(6). het>b(zal< (willen< (weg>gooien))):n ur s

[lmd,$VAR(0),[app,[app,shall,[app,want,[app,[app,throw,away],it]]],$VAR(0)]]

d(7). het>b(weg> (willen<gooien)):n ur s

d(8). het> (weg>b(willen<gooien)):n ur s

d(9). het> (weg> (willen<b(gooien))):n ur s

d(10). leest< (jan> (boeken>b(0))):q

[app,[app,read,books],j]

d(11). kan< (jan> (boeken>b(lezen))):q

[app,[app,can,[app,read,books]],j]

d(12). zal< (jan> (boeken>b(kunnen<lezen))):q

[app,[app,shall,[app,can,[app,read,books]]],j]

d(13). zal< (jan> (het>b(willen< (weg>gooien)))):q

[app,[app,shall,[app,want,[app,[app,throw,away],it]]],j]
    no
?-
```

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1994

LSI–94–1–R "Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.

LSI–94–2–R "Degrees and reducibilities of easy tally sets", Montserrat Hermo.

LSI–94–3–R "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI–94–4–R "Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.

LSI–94–5–R "A multiple shooting vectorial algorithm for progressive radiosity", Blanca Garcia and Xavier Pueyo.

LSI–94–6–R "Construction of the Face Octree model", Núria Pla-Garcia.

LSI–94–7–R "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI–94–8–R "A transformation scheme for double recursion", José L. Balcázar.

LSI–94–9–R "On architectures for federated DB systems", Fèlix Saltor, Benet Campderrich, and Manuel García-Solaco.

LSI–94–10–R "Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.

LSI–94–11–R "A top-down design of a parallel dictionary using skip lists", Joaquim Gabarró, Conrado Martínez, and Xavier Messeguer.

LSI–94–12–R "Analysis of an optimized search algorithm for skip lists", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.

LSI–94–13–R "Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.

LSI–94–14–R "A volume visualization algorithm using a coherent extended weight matrix", Daniela Tost, Anna Puig, and Isabel Navazo.

LSI–94–15–R "Deriving transaction specifications from deductive conceptual models of information systems", María Ribera Sancho and Antoni Olivé.

LSI–94–16–R "Some remarks on the approximability of graph layout problems", Josep Díaz, María J. Serna, and Paul Spirakis.

LSI–94–17–R "SAREL: An assistance system for writing software specifications in natural language", Núria Castell and Àngels Hernández.

LSI–94–18–R "Medición del factor modificabilidad en el proyecto LESD" (written in Spanish), Núria Castell and Olga Slávkova

LSI–94–19–R "Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI–94–20–R "Una paral·lelització SIMD de la conversió d'objectes codificats segons el model de fronteres al mdel d'octrees clàssics" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI–94–21–R "Clausal proof nets and discontinuity", Glyn Morrill.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es