Generation as Deduction on Labelled Proof Nets

Josep M. Merenciano* and Glyn Morrill**

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord
Jordi Girona Salgado, 1–3
E–08034 Barcelona

Abstract. In the framework of labelled proof nets the task of *parsing* in categorial grammar can be reduced to the problem of first-order matching under theory. Here we shall show how to use the same method of labelled proof nets to reduce the task of *generating* to the problem of higher-order matching.

1 Introduction

Categorial grammar provides a mechanism for the analysis of linguistic expressions on the basis of lexicalism and the parsing as deduction paradigm ([17]).[3] In accordance with *lexicalism* each lexical entry of the language encapsulates all the information needed to analyse the lexical item, and the grammar itself only needs to know how to manage these resources. In the particular case of categorial grammar, a lexical categorisation is a formula, or type, constructed over some basic types by logical connectives; and the grammar constitutes the connectives' syntactic behaviour (i.e. the laws governing the connectives). Within the *parsing as deduction* paradigm the problem of analysing some linguistic expression is rendered as the problem of proving (i.e. deducing) theorems in a deductive system. In categorial grammar this means that to analyse a linguistic expression we have to construct a sequent and prove its validity in the logic of categorial connectives: the linguistic expression is well-formed if and only if the sequent is valid in the logic. Thus, the language accepted is defined not by a set of grammar rules (as in context free grammar) but by the meaning of the categorial types assigned to the lexical items.

In practise it is not parsing itself which is useful so much as the fact that parsing is tantamount to performing the process of *interpretation*: computing the semantics associated with a given concrete syntax (we shall say: prosodics). The opposite process is *generation*: computing the prosodics associated with a given semantics. For present purposes prosodics is limited to word order, and the semantics is understood as a logical form expressed as a term of higher-order logic. We offer here a uniform framework for the computational tasks of interpretation and generation.

* E-mail: `meren@lsi.upc.es`.
** E-mail: `morrill@lsi.upc.es`, HTTP: `//www-lsi.upc.es/~glyn/`.
[3] The work we report was partially supported by project KOALA: DGICYT PB95–0787.

Methods for categorial interpretation based on proof nets ([5], [1], [18]) and labelling of deductive systems ([3]) have been developed in [12], [14], [15] and [11]. The formalism of *proof nets* provides a representation of the fundamental structure of proofs, in the same way that parse trees do for context free grammar derivations. Using proof nets we avoid "spurious ambiguity": it is always the case that two distinct (Cut-free) proof nets represent distinct associations of prosodics and semantics. In *labelled deductive systems* we use labels to formalise metalanguage of connectives in *labelled formulas* which are pairs $\langle$label, formula$\rangle$. In the categorial application the label is split into the two linguistic dimensions: $\langle\langle$prosodics, semantics$\rangle$, categorial type$\rangle$.

Combining labelling and proof nets yields labelled proof nets for categorial grammar, a parsing framework that expresses the proof search restrictions in terms of (first-order) unifications. In fact, a clausal structuring of proof search allows one to deal with one-way unification, i.e. *matching*, in which one of the two terms to be unified contains no variables. Starting the search for proof nets with the prosodics of the goal instantiated to a ground term but its semantics expressed with a metavariable, the interpretation of a linguistic expression is computed by constructing a proof net: the prosodic labelling controls the proof search, and the semantic labelling allows us to retrieve the associated semantic form.

In this paper we invoke the same techniques used for parsing and interpreting linguistic expressions in order to generate from the logical form. The idea is that we can use labelled formulas in such a way that the semantic labelling controls the proof search while the prosodic labelling is used to retrieve the word order associated with the initial logical form. The main difficulty arises with the label unification: we must now unify typed λ-terms, for which even the second order problem is undecidable ([6]). In the special case of Second-Order Linear Unification (SOLU: where variables are first-order but constants may be first- or second-order, and each abstraction binds exactly one variable occurrence) unifiability is still undecidable. But if no free variable occurs more than twice, SOLU is decidable ([10]). Importantly, the labelled categorial proof nets adhere to this latter condition: each free variable appears exactly twice. Indeed, a clausal structuring again maintains a flow of information such that one term in each unification pair contains no free variables, i.e. we need only to deal with matching. In SOL matching there is a computable finite set of most general unifiers. Thus, we are able to present a terminating algorithm for categorial generation for the case of second order implicational categorial logic.[4]

Section 2 outlines proof nets for implicational linear logic; section 3 describes the methods involved in our categorial parsing as deduction on labelled proof nets; section 4 shows how to use these methods for the task of generation; finally, in the appendix we describe higher-order unification and the SOLU matching algorithm.

[4] It is a sufficient condition for termination of our method that every lexical logical form contain at least one constant. The method is complete for (second-order) logical forms without logical constants.

2 Calculus of linear implication

In this section we outline construction of proof nets for implicational linear logic. This serves two purposes. Firstly, the calculus of linear implication provides a point of reference for sublinear categorial calculi such as associative Lambek calculus to be used later. Secondly, and more importantly, the applications of the latter to linguistic processing will be seen as a refinement of the basic problem of linear implicational theorem proving considered here.

Let us assume sequents $\Gamma \Rightarrow A$ where Γ is a multiset (bag) of formulas, and A a single formula, built just out of the linear implication. The (intuitionistic) linear sequent calculus is as follows.

(1) a. $A \Rightarrow A$ id

b. $$\frac{\Gamma \Rightarrow A \qquad A, \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow B}\text{Cut}$$

c. $$\frac{\Gamma \Rightarrow A \qquad B, \Delta \Rightarrow C}{\Gamma, A{\multimap}B, \Delta \Rightarrow C}{\multimap}\text{L}$$

d. $$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A{\multimap}B}{\multimap}\text{R}$$

This calculus is also known in categorial contexts as Lambek-van Benthem calculus. It enjoys Cut-elimination, i.e. every sequent which is a theorem has a Cut-free proof, and is thus decidable since in the two logical rules the conclusion has one more connective than the premises. However, the sequent proofs copy contexts around in a cumbersome manner, and the partitionings required by binary rules are a costly form of non-determinism in proof search. A much deeper proof syntax is provided by proof nets.

Cut-elimination also entails the subformula property: all the formulas that can appear in a Cut-free proof already occur as subformulas of the sequent to be proved. In proof nets we work directly on the formation trees of formulas, in which all subformulas are already present as subtrees. We mark all subformulas with an explicit polarity, $^{\mathrm{a}}$ or $^{\mathrm{s}}$, to indicate antecedent or succedent occurrences, rather than using positioning with respect to the sequent arrow; these polar formulas are unfolded recursively into formation trees with atomic leaves as follows:

(2) $$\frac{A^{\overline{p}} \qquad B^{p}}{A{\multimap}B^{p}}$$

The polarities p and $\overline{p}$ are complementary. Polarities are propagated in such a way that it is indicated whether subformulas would have antecedent or succedent occurrences in sequent proofs.

To try to construct a proof net for a sequent we first mark each formula in the sequent with a polarity marker $^{\mathrm{a}}$ or $^{\mathrm{s}}$, to indicate antecedent or succedent occurrences, as shown in (3).

(3) $$\frac{A^{\mathrm{s}}\ A_1^{\mathrm{a}}\ \ldots\ A_n^{\mathrm{a}}}{A_1, \ldots, A_n \Rightarrow A}$$

The result is a bag of formulas with polarity. We then recursively unfold these polar formulas. The result of this is called a *proof frame*. A *proof structure* is the result of linking each literal to exactly one other, which must have the same atom with the opposite polarity. A proof structure is a *proof net*, i.e. is well-formed as a proof, if and only if it meets a global condition, the *long trip condition*, which can be expressed in various ways, and which ensures that the proof structure corresponds to a sequent proof. The links of proof nets are instances of the sequent axiom; but it must be assured that in $\multimap$R inferences the hypothetical A really is used in the proof of B and not in some other subproof.

Following on earlier work ([14], [15], [11]), we present the following linear clausal engine for the construction of proof nets, though without any proof of correctness here. It addresses the basic problem of partitioning for binary sequent rules by putting a list of goals in the consequent of a single sequent and using unary sequent rules together with checks that hypotheses have been used in the requisite subproofs. Formulas are labelled with constants and variables of an Associative and Commutative (AC) term algebra representing bags. A sequent $\Delta \Rightarrow \Sigma$ comprises a database Δ which is a bag of formulas labelled by distinct constants, and an agenda Σ which is a list of items each of which is either a formula labelled with a variable (all distinct), or an assignment := to a variable of an AC term formed by multiset addition $\oplus$ (a total operation), subtraction $\ominus$ (a partial operation) and the empty bag $\emptyset$. One attempts to prove a sequent $A_1, \ldots, A_n \Rightarrow A$ by proving $a_1\colon A_1, \ldots, a_n\colon A_n \Rightarrow [\alpha\colon A]$. The search terminates successfully with proof of the empty agenda from the empty database:

(4) $$\Rightarrow []$$

The variables labelling agenda formulas will in fact be assigned the labels of those database formulas which are used in their proof, i.e. in the overall case, α will be $a_1 \oplus \ldots \oplus a_n$. There are three rules. Reading from conclusion to premises, RES (resolution) states that to prove an atom A first on the agenda, choose a database clause with head A which through zero or more implications implies A; prove the antecedents of these implications, and the label for A is then the sum of those for the antecedent proofs plus that for the clause chosen (we now write the implications in the logic programming, right-to-left, direction):

(5) $$\frac{\Delta \Rightarrow [\alpha_1\colon A_1, \ldots, \alpha_n\colon A_n, \alpha := \alpha_1 \oplus \cdots \oplus \alpha_n \oplus k|\Sigma]}{\Delta, k\colon (\cdots(A \multimapinv A_n) \multimapinv \cdots) \multimapinv A_1 \Rightarrow [\alpha\colon A|\Sigma]}\text{RES, } \alpha_i \text{ new vars.}$$

The rule DT (deduction theorem) states that to prove $B \multimapinv A$ first on the agenda one assumes A and proves B, and then checks that A has been used to prove B:

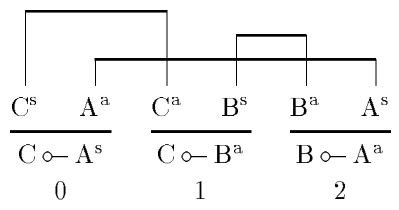
(6)
$$\frac{\Delta, k\colon A \Rightarrow [\beta\colon B, \gamma := \beta \ominus k|\Sigma]}{\Delta \Rightarrow [\gamma\colon B \multimapinv A|\Sigma]}\text{DT, } k \text{ new constant, } \beta \text{ new variable}$$

When the assignment condition in (6) is checked by (7) the evaluation succeeds if the hypothesis has been used but fails otherwise (since $\ominus$ is a partial operation):

(7)
$$\frac{\Delta \Rightarrow \Sigma[\alpha \leftarrow EVAL(\alpha')]}{\Delta \Rightarrow [\alpha := \alpha'|\Sigma]}\text{Assig}$$

Let us consider the construction of the proof net for $C \multimapinv B, B \multimapinv A \Rightarrow C \multimapinv A$:

(8)



C^{s} A^{a} C^{a} B^{s} B^{a} A^{s}
$\mathrm{C} \multimapinv \mathrm{A}^{\mathrm{s}}$
$\mathrm{C} \multimapinv \mathrm{B}^{\mathrm{a}}$
$\mathrm{B} \multimapinv \mathrm{A}^{\mathrm{a}}$
0
1
2

We shall reference subformulas by their tree, 0, 1, ... from left-to-right, and their node address within the tree given as a sequence of l(eft)s and r(ight)s starting at the root. Then, reading from the conclusion up to the axiom, the successive states in the construction of (8) are as follows.

(9)
$$\begin{array}{cl}
\Rightarrow [] & \\ \hline
\Rightarrow [\alpha_0 := (\mathrm{a}_3 \oplus \mathrm{a}_2 \oplus \mathrm{a}_1) \ominus \mathrm{a}_3] & \text{Assig} \\ \hline
\Rightarrow [\alpha_1 := \mathrm{a}_3 \oplus \mathrm{a}_2 \oplus \mathrm{a}_1, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{Assig} \\ \hline
\Rightarrow [\alpha_2 := \mathrm{a}_3 \oplus \mathrm{a}_2, \alpha_1 := \alpha_2 \oplus \mathrm{a}_1, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{Assig} \\ \hline
\Rightarrow [\alpha_3 := \mathrm{a}_3, \alpha_2 := \alpha_3 \oplus \mathrm{a}_2, \alpha_1 := \alpha_2 \oplus \mathrm{a}_1, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{Assig} \\ \hline
\mathrm{a}_3\colon 0r \Rightarrow [\alpha_3\colon 2r, \alpha_2 := \alpha_3 \oplus \mathrm{a}_2, \alpha_1 := \alpha_2 \oplus \mathrm{a}_1, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{RES} \\ \hline
\mathrm{a}_2\colon 2, \mathrm{a}_3\colon 0r \Rightarrow [\alpha_2\colon 1r, \alpha_1 := \alpha_2 \oplus \mathrm{a}_1, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{RES} \\ \hline
\mathrm{a}_1\colon 1, \mathrm{a}_2\colon 2, \mathrm{a}_3\colon 0r \Rightarrow [\alpha_1\colon 0l, \alpha_0 := \alpha_1 \ominus \mathrm{a}_3] & \text{RES} \\ \hline
\mathrm{a}_1\colon 1, \mathrm{a}_2\colon 2 \Rightarrow [\alpha_0\colon 0] & \text{DT}
\end{array}$$

To begin, one is trying to prove 0 from 1 and 2. Since 0 is implicational, in the first step $0r$ is added to the database and $0l$ is put on the agenda. This goal is attempted by resolution with clause 1 (highest link in the proof net). The new goal issued is attempted by resolution with clause 2 (middle link in the proof net). The next goal issued is resolved with the unit clause $0r$ put into the database at the first step (lowest link in the proof net).

3 Categorial parsing as deduction on labelled proof nets

3.1 Lambek Calculus

We shall deal with an implicational version **L** of associative Lambek Calculus ([9]) with formulas or (categorial) types defined by the connectives \ ('under') and / ('over') on the basis of atomic types $\mathcal{A}$, as shown in (10).

(10) $\mathcal{F} = \mathcal{A} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}/\mathcal{F}$

The two connectives are directional implications. By way of illustration of the notation, let us assume atomic types such as S (sentence), N (nominal), CN (common noun) and PP (prepositional phrase); then intransitive verbs, requiring a subject nominal on the left to form a sentence, have type N\S; transitive verbs, combining with an object on the right to form an intransitive verb phrase, have type (N\S)/N.

The interpretation of the categorial connectives is made prosodically in the field of a semigroup, i.e. a set L closed under an associative operation $+$, and semantically in a frame of function spaces, i.e. an indexed family $\{D_\tau\}_{\tau\in\mathcal{T}}, \mathcal{T} = \mathcal{D} \mid \mathcal{T} \rightarrow \mathcal{T}$ where $\{D_\tau\}_{\tau\in\mathcal{D}}$ are basic domains, and $D_{\tau_1\rightarrow\tau_2}$ is the set of functions from D_{τ_1} to D_{τ_2}. A mapping T which associates a semantic function space with each categorial type is such that $T(A\backslash B) = T(B/A) = T(A) \rightarrow T(B)$. Each categorial type A is interpreted as a subset of $L \times T(A)$. The signs of type $A\backslash B$ (B/A) are those which concatenate prosodically with signs of type A on the left (right), and apply semantically as functions, to yield signs of type B:

(11) $D(A\backslash B) = \{\langle s, m\rangle | \forall\langle s', m'\rangle \in D(A), \langle s'{+}s, m(m')\rangle \in D(B)\}$
$D(B/A) = \{\langle s, m\rangle | \forall\langle s', m'\rangle \in D(A), \langle s{+}s', m(m')\rangle \in D(B)\}$

In order to present calculi for reasoning about categorial types we use labelling to codify information from the interpretation clauses. Prosodic labels are terms over variables and constants constructed by the operator $+$; semantic labels are typed λ-terms. We define a sequent calculus as follows.[5] A *type assignment statement* is of the form $\alpha - \phi$: A where α is a prosodic term, ϕ a semantic term and A a categorial type. A *configuration* is a multiset (bag) of type assignment statements in which the terms are all variables, and are all distinct. A *sequent* $\Gamma \Rightarrow X$ comprises an antecedent Γ which is a configuration and a succedent X which is a type assignment statement. We read a sequent as stating that (for all interpretations), if the objects referred to in the antecedent are in

[5] The prosodic labelling is not essential for Lambek sequent calculus: the prosodic information can be left implicit in antecedents structured as sequences (ordered sequent calculus). The semantic information can also be recovered from a sequent proof: the associated lambda term is a notation for the proof as natural deduction, according to the Curry-Howard correspondence. But labelled sequent calculus is more general than ordered sequent calculus; both the prosodic and the semantic labelling are used in the subsequent development of proof nets, and the methods we describe apply not just to Lambek calculus but to a wider class of categorial logics which can be expressed in the general labelled format.

the types indicated, then the object referred to in the succedent is in the type indicated. The theorems of the calculus are generated by the following sequent rules.

(12) a. $a - x\colon A \Rightarrow a - x\colon A \quad \text{id}$

b. $$\frac{\Gamma \Rightarrow \alpha - \phi\colon A \qquad a - x\colon A, \Delta \Rightarrow \beta[a] - \psi[x]\colon B}{\Gamma, \Delta \Rightarrow \beta[\alpha] - \psi[\phi]\colon B}\text{Cut}$$

c. $$\frac{\Gamma \Rightarrow \alpha - \phi\colon A \qquad b - y\colon B, \Delta \Rightarrow \gamma[b] - \chi[y]\colon C}{\Gamma, d - w\colon A\backslash B, \Delta \Rightarrow \gamma[\alpha{+}d] - \chi[(w\ \phi)]\colon C}\backslash\text{L}$$

d. $$\frac{\Gamma, a - x\colon A \Rightarrow a{+}\gamma - \psi\colon B}{\Gamma \Rightarrow \gamma - \lambda x\psi\colon A\backslash B}\backslash\text{R}$$

e. $$\frac{\Gamma \Rightarrow \alpha - \phi\colon A \qquad b - y\colon B, \Delta \Rightarrow \gamma[b] - \psi[y]\colon C}{\Gamma, d - w\colon B/A, \Delta \Rightarrow \gamma[d{+}\alpha] - \psi[(w\ \phi)]\colon C}/\text{L}$$

f. $$\frac{\Gamma, a - x\colon A \Rightarrow \gamma{+}a - \psi\colon B}{\Gamma \Rightarrow \gamma - \lambda x\psi\colon B/A}/\text{R}$$

The notation $[\cdot]$ indicates distinguished suboccurrences of terms. By way of example, there is the following derivation of a case of "subject type raising":

(13) $$\frac{\dfrac{a - x\colon \text{N} \Rightarrow a - x\colon \text{N} \qquad c - z\colon \text{S} \Rightarrow c - z\colon \text{S}}{a - x\colon \text{N},\ b - y\colon \text{N}\backslash\text{S} \Rightarrow a{+}b - (y\ x)\colon \text{S}}\backslash\text{L}}{a - x\colon \text{N} \Rightarrow a - \lambda y(y\ x)\colon \text{S}/(\text{N}\backslash\text{S})}/\text{R}$$

Each lexical entry is a type assignment statement $\alpha - \phi\colon A$ where α and ϕ are closed (contain no free variables). Examples of lexical assignments are given in figure 1.[6] Consider the following derivation:

(14) $$\frac{\dfrac{a - x\colon \text{CN} \Rightarrow a - x\colon \text{CN} \qquad b - y\colon \text{N} \Rightarrow b - y\colon \text{N}}{d - w\colon \text{N/CN},\ a - x\colon \text{CN} \Rightarrow d{+}a - (w\ x)\colon \text{N}}/\text{L} \qquad c - z\colon \text{S} \Rightarrow c - z\colon \text{S}}{d - w\colon \text{N/CN},\ a - x\colon \text{CN},\ e - v\colon \text{N}\backslash\text{S} \Rightarrow d{+}a{+}e - (v\ (w\ x))\colon \text{S}}\backslash\text{L}$$

Substituting the prosodics and semantics for 'the', 'dog' and 'runs' we derive that **the+dog+runs** with semantics (**run** (**the dog**)) is a sentence. Of course the lexical semantics can be elaborated and more complex examples may invite λ-reduction in computational implementations, but we see here the essential features of analysis. We can also see here the essential computational problem

[6] We omit here details of inflection and morphology; see e.g. chapter 6 of [13].

John	–	**j**	: N
Mary	–	**m**	: N
runs	–	**run**	: N\S
likes	–	**like**	: (N\S)/N
votes	–	**vote**	: (N\S)/PP
talks	–	**talk**	: (N\S)/PP
for	–	**for**	: PP/N
about	–	**about**	: PP/N
the	–	**the**	: N/CN
dog	–	**dog**	: CN
who	–	$\lambda x \lambda y \lambda z[(y\ z) \wedge (x\ z)]$	: (CN\CN)/(S/N)
seeks	–	$\lambda x(\mathbf{try}\ (x\ \mathbf{find}))$	: (N\S)/(((N\S)/N)\(N\S))

Fig. 1. Lexical Assignments

with this proof syntax: although the Cut-elimination property renders decidability (since in the logical rules the conclusion has one more connective than the premises) distinct proofs may define the same analysis. For example (15), which is not the same proof as (14), nevertheless derives the same labelled conclusion.

(15)
$$\dfrac{a - x\colon \mathrm{CN} \Rightarrow a - x\colon \mathrm{CN} \qquad \dfrac{b - y\colon \mathrm{N} \Rightarrow b - y\colon \mathrm{N} \qquad c - z\colon \mathrm{S} \Rightarrow c - z\colon \mathrm{S}}{b - y\colon \mathrm{N},\ e - v\colon \mathrm{N}\backslash\mathrm{S} \Rightarrow b{+}e - (v\ y)\colon \mathrm{S}}\backslash L}{d - w\colon \mathrm{N}/\mathrm{CN},\ a - x\colon \mathrm{CN},\ e - v\colon \mathrm{N}\backslash\mathrm{S} \Rightarrow d{+}a{+}e - (v\ (w\ x))\colon \mathrm{S}}/L$$

This “spurious ambiguity” of the sequent proof syntax is remedied in the syntax of proof nets, to which we now turn.

3.2 Labelled Proof Nets

As before, in the proof nets we work directly on the formation trees of formulas, but now they are labelled. Marking labelled subformulas for polarity $^{\mathrm{a}}$ or $^{\mathrm{s}}$, polar type assignment statements are unfolded recursively into formation trees with atomic leaf types as follows:[7]

(16)
$$\dfrac{\alpha - \phi\colon A^{\overline{p}} \qquad \alpha{+}\gamma - (\chi\ \phi)\colon B^{p}}{\gamma - \chi\colon A\backslash B^{p}} \qquad \dfrac{\gamma{+}\alpha - (\chi\ \phi)\colon B^{p} \qquad \alpha - \phi\colon A^{\overline{p}}}{\gamma - \chi\colon B/A^{p}}$$
$$\alpha \text{ and } \phi \text{ new variable/constant as } p = \mathrm{a}/\mathrm{s}$$

Metavariables and Skolem constants correspond to the quantifers of the inter-

[7] The unfolding is a little different than that in [12], which instantiates λ-abstraction in succedent unfolding in such a way that extraction of semantics in parsing is immediate. The current version will be used uniformly for parsing and generating, serving to maximise a symmetry that would otherwise be less apparent.

pretation clauses and are introduced into labels in accordance with the polarity of context.

The definitions of proof frame and proof structure are just as before, but now we have labels. Any sublinear calculus must satisfy the linear long trip condition on proof nets, to ensure linear validity. But we have a further condition in view of sublinear structure, which is that the unification problem comprising the linked prosodic terms be solvable. So far as we are aware, the prosodic and semantic labelling actually subsume the linear labelling of section 2, in that unifiability of either ensures the relevant use of hypotheses; however for consistency we continue to include the linear labelling.[8]

This means, then, that in order to check that a proof structure is a proof net we have to determine the solvability of a *first-order unification system*, i.e. the solvability (under associativity) of the set of prosodic equations induced by the linking. However, from a processing point of view we do not want to construct whole proof structures and then test if they are proof nets, but rather propagate constraints and prune search in the course of conjecturing linking. Indeed, we do not want to take a given proof frame as our point of departure, since that presupposes a selection of lexical assignments: for n words each k-ways lexically ambiguous there are k^n such choices and we do not want to have to enumerate them all, but select them only when they must be brought into, and are compatible with, the search and proof construction.

A suitable method is obtained by generalising the clausal engine of the previous section, which in fact allows us to practice a top-down backtracking parse search restricted to one-way unification (matching), i.e. unification in which one term has no free variables. The generalisation includes resolution with lexical clauses, with control of the label tokens being introduced into the proof. Sequents now have the form $\varDelta \Rightarrow_{\delta} \varSigma$ where a control parameter δ is a multiset of constant tokens, the cardinalities which lexical insertion must meet as a necessary condition for successful proof construction. The lexical insertion resolution rule LRES requires the lexical assignment of type $(\cdots(A \multimap A_n) \multimap \cdots) \multimap A_1$ (ignoring directionality) to a sign ς and decrements $\delta \ominus \#(\varsigma)$ according to the count $\#(\varsigma)$ of the lexical sign ς on the label dimension controlling the proof search (prosodics for parsing; semantics for generation). The remaining rules are

[8] By retaining an appropriate ordering it is possible to restrict attention in **L** to *planar* linking in virtue of noncommutativity, which is certainly of crucial computational importance (though not enough to ensure the long trip condition). But our first concern here is with the generality of our methodology for generation, which does not need to rely on any noncommutativity and which extends to all manner of sublinear calculi through unification under theory as in [14]. Although introduced as long ago as [18], whether the prosodic unifiability alone assures the long trip condition has not been shown. Nor does it appear that unification under associativity (nondeterministic) is imperative for **L**: [15] and [16] propose formulations on the basis of just structural term unification (deterministic). Still, it seems unlikely that the present proposals would be irrelevant to such refinements as could be either necessary or advantageous. That the semantic unification is necessary but not sufficient is certain, since this checks validity as natural deduction, but does not check order.

unaltered except that they transmit the control parameter.

(17) $$\frac{}{\Rightarrow_{\emptyset}[]}$$

(18) $$\frac{\Delta \Rightarrow_{\delta}[\alpha_1\colon A_1,\ldots,\alpha_n\colon A_n, \alpha := \alpha_1 \oplus\cdots\oplus\alpha_n\oplus k|\Sigma]}{\Delta, k\colon (\cdots(A \circ\!\!- A_n)\circ\!\!- \cdots)\circ\!\!- A_1 \Rightarrow_{\delta}[\alpha\colon A|\Sigma]}\text{RES, } \alpha_i \text{ new vars.}$$

(19) $$\frac{\Delta \Rightarrow_{\delta}[\alpha_1\colon A_1,\ldots,\alpha_n\colon A_n, \alpha := \alpha_1 \oplus\cdots\oplus\alpha_n|\Sigma]}{\Delta \Rightarrow_{\delta\oplus\#(\varsigma)}[\alpha\colon A|\Sigma]}\text{LRES, } \alpha_i \text{ new vars.}$$

(20) $$\frac{\Delta, k\colon A \Rightarrow_{\delta}[\beta\colon B, \gamma := \beta\ominus k|\Sigma]}{\Delta \Rightarrow_{\delta}[\gamma\colon B \circ\!\!- A|\Sigma]}\text{DT, } k \text{ new constant, } \beta \text{ new variable}$$

(21) $$\frac{\Delta \Rightarrow_{\delta}\Sigma[\alpha \leftarrow EVAL(\alpha')]}{\Delta \Rightarrow_{\delta}[\alpha := \alpha'|\Sigma]}\text{Assig}$$

Let us consider first just parsing our example 'the dog runs' as S; see figure 2. The trace of search states is as follows:

(22) $$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Rightarrow_{\emptyset}[]}{\Rightarrow_{\emptyset}[\alpha_0 := \emptyset]}\text{Assig}}{\Rightarrow_{\emptyset}[\alpha_1 := \emptyset, \alpha_0 := \alpha_1]}\text{Assig}}{\Rightarrow_{\emptyset}[\alpha_2 := \emptyset, \alpha_1 := \alpha_2, \alpha_0 := \alpha_1]}\text{Assig}}{\Rightarrow_{\mathrm{dog}}[\alpha_2\colon 2r, \alpha_1 := \alpha_2, \alpha_0 := \alpha_1]}\text{LRES}}{\Rightarrow_{\mathrm{the}\oplus\mathrm{dog}}[\alpha_1\colon 1l, \alpha_0 := \alpha_1]}\text{LRES}}{\Rightarrow_{\mathrm{the}\oplus\mathrm{dog}\oplus\mathrm{runs}}[\alpha_0\colon 0]}\text{LRES}$$

Initially, the agenda comprises the type S unit clause at 0 with prosodic term **the+dog+runs**. There are no clauses in the database, so we must resolve by LRES with a lexical clause projecting a type S head. We do not need to attempt resolving with any lexical entry unless the prosodic constant(s) occurring in the entry are contained in the bag of tokens controlling the proof search.[9] Since 'the' and 'dog' do not project S, we can only resolve with the lexical clause for 'runs', the prosodic label of which must be unified with **the+dog+runs**. This instantiates a to **the+dog** (in this case there is no other unifier) and the new agenda comprises the type N unit clause at $1l$ with prosodic term **the+dog**. For the same reasons again, this must resolve with the lexical clause for 'the', in-

[9] In practice, we can precompute a *working set* of lexical entries, being those the constants of which are contained in the target assignment. Skolem constants and metavariables need to be refreshed on each invocation, but, the rest of the lexicon can be ignored.

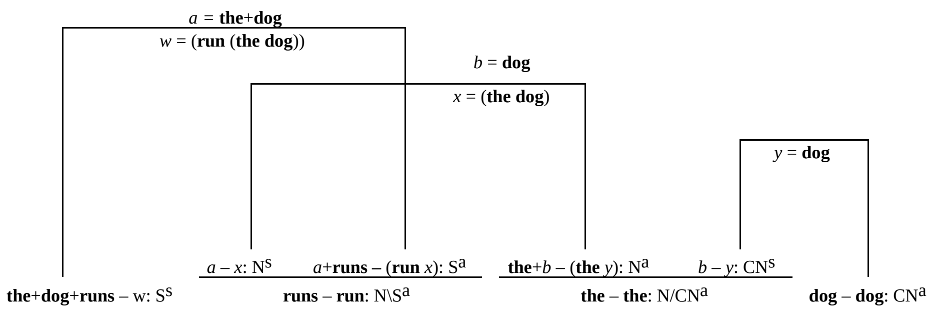


Fig. 2. Proof net for parsing of 'the dog runs'

stantiating b to **dog**, and the subgoal issued resolves with the unit lexical clause for 'dog'. Tracing back, the three successive steps yield semantically $y = \mathbf{dog}$, $x = (\mathbf{the\ dog})$ and $w = (\mathbf{run}\ (\mathbf{the\ dog}))$, which last represents the semantic interpretation. Note that this is the only proof net for this (unambiguous) expression: the problem of spurious ambiguity that is encountered in the syntax of sequent calculus is not present in that of proof nets.

By way of a slightly more involved example we consider parsing 'John talks about' as S/N (as in the analysis of a relative clause 'who John talks about' given the second-order assignment to the relative pronoun); see figure 3. First, the unit N clause in the body of the initial agenda is put into the database and the S head is linked to the head of the lexical clause for 'talks' which projects an S. The prosodic unifier is $\{b = \mathbf{John}, a = \mathbf{about}{+}\mathbf{d}\}$. The two (unit) clauses in the body of the lexical clause go into the agenda; the N is resolved with the unit lexical clause for 'John', and the PP with the lexical clause for 'about', when $c = \mathbf{d}$. Now the agenda comprises an N unit clause with prosodic label **d** and we terminate by resolving with the corresponding clause added to the database at the first step. Tracing back the semantics we end up with $(v\ \mathbf{w}) = ((\mathbf{talk}\ (\mathbf{about}\ \mathbf{w}))\ \mathbf{j})$ which clearly has solution $v = \lambda w((\mathbf{talk}\ (\mathbf{about}\ w))\ \mathbf{j})$.

This last example shows that recovery of semantics can involve a mild degree of higher-order unification. In the continuation we address the task of generation and this will require a direct confrontation with higher-order unification.

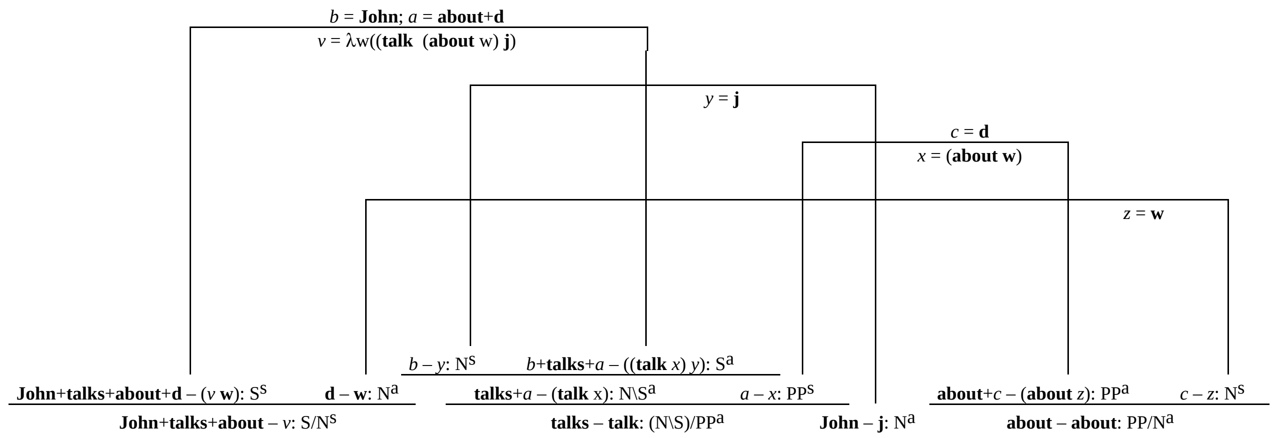

b = John; a = about+d
v = λw((talk (about w) j)
y = j
c = d
x = (about w)
z = w
b – y: N^s
b+talks+a – ((talk x) y): S^a
John+talks+about+d – (v w): S^s
d – w: N^a
talks+a – (talk x): N\S^a
a – x: PP^s
about+c – (about z): PP^a
c – z: N^s
John+talks+about – v: S/N^s
talks – talk: (N\S)/PP^a
John – j: N^a
about – about: PP/N^a

Fig. 3. Proof net for parsing of 'John talks about'

4 Categorial generation as deduction on labelled proof nets.

Now we will use the same techniques to *generate* from the semantic form. The basic idea is that we can use bidimensional labelled formulas $\alpha - \phi$: A in such a way that the semantic labelling controls the proof search while the prosodic labelling is used to retrieve the word order associated with the semantic form. To do this it is enough to express in the semantic label of the goal a closed λ-term representing the source semantics, leave a metavariable in the prosodic label, determine solvability of the unification system expressed by (semantic) linking, and confirm solvability of the unification system induced over prosodic labels, recovering the prosodic unknown.

Using the goal-driven strategy for proof search above we have to resolve, for each linking, a higher-order matching problem. Like matching under associativity, higher-order matching is non-deterministic in that (already considering only normal forms) in general there is not a unique most general unifier. For example, the unification problem $\{(x\ \mathbf{f}) = (\mathbf{f}\ (\mathbf{f}\ \mathbf{a}))\}$ has solucions $x = \lambda z(z\ (\mathbf{f}\ \mathbf{a}))$ and $x = \lambda z(\mathbf{f}\ (z\ \mathbf{a}))$ neither of which is more general than the other. It's known that third-order matching is decidable; though it's not known whether the problem in general is decidable, the conjecture is that it is ([7]); see also [8], [6], [4] and [2].

Decidability is not enough however: in processing we are interested in effectively computing all possible unifiers, and in general this is not possible because with higher-order terms we can have an infinite number of unifiers for a given equation that cannot be expressed with a finite number of most general unifiers. If we restrict the linguistic fragment to expressions with a second-order semantic form the problem reduces to second-order linear matching because the semantic unfolding maintains linearity. This particular problem is semi-decidable, but if no free variable occurs more than twice we obtain decidability; furthermore, in this case we have a finite number of (second-order linear) most general unifiers and an algorithm to compute them ([10]). Critically, our semantic unfolding definition is such that we have exactly two occurrences of each free variable. Note that restriction to second-order types is not so bad: Montague's grammar is at most third order.

Once a proof net has been built we can confirm ordering well-formedness and retrieve the word order associated with the semantics by solving the equations on prosodic labels generated by the linking. Thus, if the matching problem for a given fragment of λ-calculus is computable, so also is the task, within that fragment, of generating linguistic expressions from their semantics, if each lexical semantics contains at least one constant. This last condition is not necessary to ensure a finite search space, but it is sufficient, since it bounds the number of LRES inferences that can be made. It is the analogue of an assumption that lexical items contain at least one prosodic constant (i.e. that no type is lexically assigned to the empty string) which is sufficient (though not necessary) to ensure termination of certain parsing methods.

Figure 4 shows generation of 'John talks about' from $\lambda w((\mathbf{talk}\ (\mathbf{about}\ w))\ \mathbf{j})$. As with the prosodic control of search in parsing, we do not need to attempt resolving with any lexical entry unless the bag of (semantic) constants occurring lexically is contained in the bag of such constants controlling the proof search.[10] The entries for 'about' and 'John' do not project S: after adding the conditionalised N to the database, the head of the literal initially on the agenda is matched to the head of the lexical clause for 'talks'. A unifier performing the matching is $\{x = (\mathbf{about\ w}), y = \mathbf{j}\}$. The N goal resolves with 'John', the PP goal with the clausal head for 'about' under $z = \mathbf{w}$, and the N goal issued is resolved at the last step with the N put into the database at the first step.

The structure is just like that for parsing, but with known and unknown information inverted. Since matching under associativity is non-deterministic one must in general be prepared to backtrack and try different unifiers in parsing (though this does not arise in our examples); likewise for higher-order matching in generation, where indeed the non-determinism of matching is more severe. The confirmation and recovery of prosodic form is also invoked by matching, as with parsing. Working backwards we have at the last step $c = \mathbf{d}$, then $a = \mathbf{about}{+}\mathbf{d}$ and $b = \mathbf{John}$. Finally, we solve $e{+}\mathbf{d} = \mathbf{John}{+}\mathbf{talks}{+}\mathbf{about}{+}\mathbf{d}$, generating the prosodic form $\mathbf{John}{+}\mathbf{talks}{+}\mathbf{about}$.

One last example is given in figure 5.[11] The main interest is the semantic matching at the first step, which requires x to be mapped to a second-order λ-abstracted term. In the appendix we discuss an algorithm for the case of second-order matching. Although that is not enough for the current example, which is third-order, it does enable us to compute in the manner we have presented the task of generation not only for the associative Lambek calculus as elucidated, but also for a much wider range of sublinear labelled calculi. This is because the only adjustment necessary is accommodation of prosodic matching under the relevant theory. Until now no such general method has been available. The principle issues arising are: how to perform matching for a wider class of λ-terms, and how to extend treatment to include *logical* constants. We hope to be able to address these questions in future work.

[10] Again, we can in practice precompute a working set of lexical entries and ignore the rest of the lexicon.

[11] For explanation of the categorial assignment to 'seeks' see chapter 5 of [13].

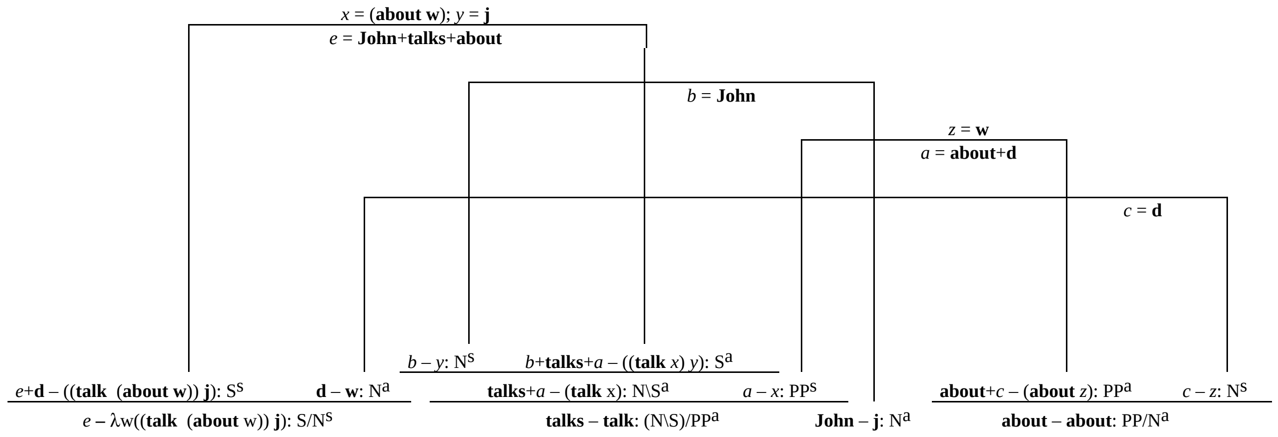


Fig. 4. Proof net for generation of 'John talks about'

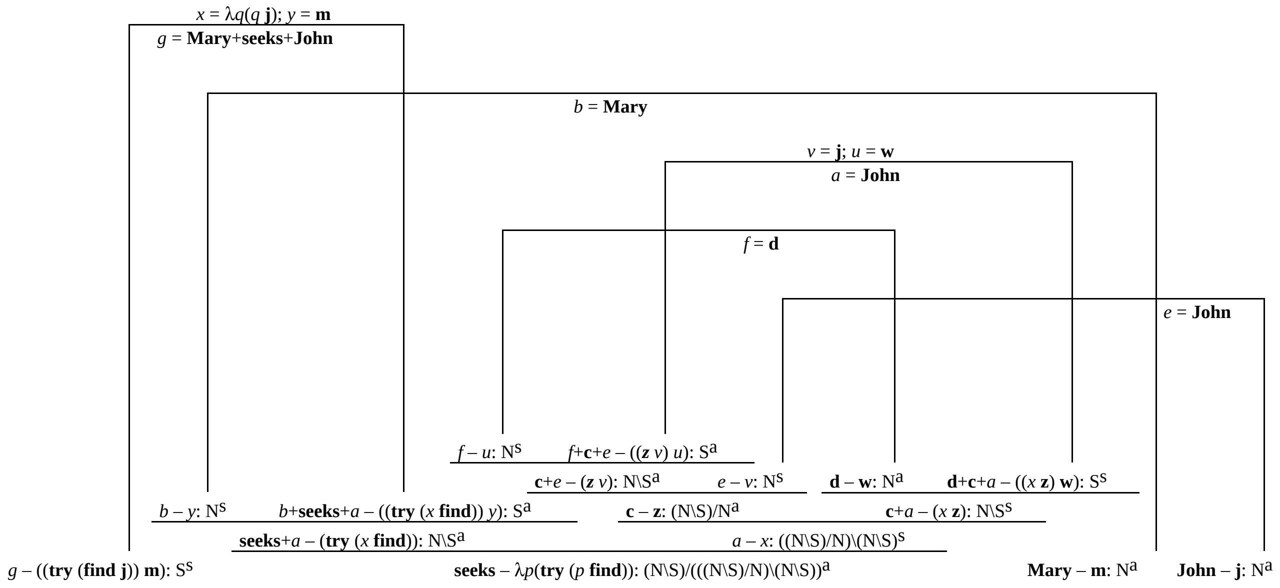

x = λq(q **j**); y = **m**
g = **Mary**+**seeks**+**John**
b = **Mary**
v = **j**; u = **w**
a = **John**
f = **d**
e = **John**
f – u: N^s
f+**c**+e – ((**z** v) u): S^a
c+e – (**z** v): N\S^a
e – v: N^s
d – **w**: N^a
d+**c**+a – ((x **z**) **w**): S^s
b – y: N^s
b+**seeks**+a – ((**try** (x **find**)) y): S^a
c – **z**: (N\S)/N^a
c+a – (x **z**): N\S^s
seeks+a – (**try** (x **find**)): N\S^a
a – x: ((N\S)/N)\(N\S)^s
g – ((**try** (**find** **j**)) **m**): S^s
seeks – λp(**try** (p **find**)): (N\S)/(((N\S)/N)\(N\S))^a
Mary – **m**: N^a
John – **j**: N^a

Fig. 5. Proof net for generation of 'Mary seeks John'

Appendix: Higher-order unification

Higher-order unification is the task of unifying typed λ-terms. We assume single-bind, i.e. linear, λ-terms. All λ-terms can be expressed in $\beta\eta$-long normal form as shown in (23a), where each ψ_i is in $\beta\eta$-long normal form; ϕ is a constant or a variable (free or bound), called the *head* of the term. In (23b) we show an abbreviated "flattened" form that we use for the nested functional application in (23a); and in (23c), vector-style notations for iterated functional application and λ-abstraction.

(23) a. $\lambda x_1 \lambda x_2 \ldots \lambda x_n (\ldots ((\phi\ \psi_1)\ \psi_2)\ \ldots \psi_m)\ m, n \geq 0$
b. $\lambda x_1 \lambda x_2 \ldots \lambda x_n . \phi(\psi_1, \psi_2, \ldots, \psi_m)$
c. $\lambda \overline{x_n} \phi(\overline{\psi_{\{1,\ldots,m\}}})$

We say a term is *rigid* when its head is a constant or a bound variable, and *flexible* when its head is a free variable.

A higher-order unification equation is a pair of terms to be unified, $\phi = \psi$, and a Higher-Order Unification System (HOUS) is a set of such equations, that must be satisfied simultaneously. A common method for solution of a HOUS is that of transformations ([19], [4]): there are a set of transformation rules that transform a system S to a system S' with the same set of unifiers. The transformation rules we will use not only transform the HOUS but will also compute a unifier; any state of the process is represented by the unifier σ computed up to the current moment and the current unification system S (to which σ has been applied): $\langle S, \sigma \rangle$; the end of the process is marked by transformation into the empty system.

Application of transformation rules has the form (24): some (*active*) unification equation $\phi = \psi$ is removed and replaced by subproblems R, and a substitution ρ is applied to the system and composed with the substitution to date.

(24) $\langle S \cup \{\phi = \psi\}, \sigma \rangle \Longrightarrow \langle \rho(S \cup R), \rho \circ \sigma \rangle$

At each stage $\beta\eta$-long normal form is to be restored. There are three kinds of transformation depending on the nature of the terms in the active equation: flexible/flexible, flexible/rigid, and rigid/rigid. The algorithm for the particular case of linear second-order matching is adapted from the unification algorithm of [10] (in fact, we only have to drop from it the flexible/flexible case).

If two terms to be unified have different length lambda prefixes the unification fails (it means they are not of the same type). Otherwise, we assume α-conversion making the λ-prefixes identical. The rigid/rigid case checks the identity of the two heads and tries to unify the parameters:

(25) $\lambda \overline{x_n} . \phi(\overline{\psi_{\{1,\ldots,m\}}}) = \lambda \overline{x_n} . \phi(\overline{\chi_{\{1,\ldots,m\}}})$
$R = \{\lambda \overline{x_n} . \psi_1 = \lambda \overline{x_n} . \chi_1, \ldots, \lambda \overline{x_n} . \psi_m = \lambda \overline{x_n} . \chi_m\}$
$\rho = \{\}$

For the flexible/rigid case we choose in a don't know non-deterministic way to *project* or to *imitate*. The projection rule instantiates the head of the flexible term, a free variable x, with the identity function (it must, then, be of a first-order endocentric type $\tau \rightarrow \tau, \tau \in \mathcal{D}$):

(26) $\lambda\overline{x_n}.\phi(\overline{\psi_{\{1,\ldots,m\}}}) = \lambda\overline{x_n}.x(\chi)$
$R = \{\lambda\overline{x_n}.\phi(\overline{\psi_{\{1,\ldots,m\}}}) = \lambda\overline{x_n}.\chi\}$
$\rho = \{x = \lambda y.y\}$

The imitation rule decomposes (in a don't know non-deterministic way) the flexible term into a set of m new flexible terms, arranging a linear allocation of the arguments $\chi_1, \ldots, \chi_p$ as the arguments of m new variables $z_1, \ldots, z_m$ to be unified with the m terms $\psi_1, \ldots, \psi_m$ This allocation is given by a partition $R_1, \ldots, R_m$ of $1, \ldots, p$.

(27) $\lambda\overline{x_n}.\phi(\overline{\psi_{\{1,\ldots,m\}}}) = \lambda\overline{x_n}.x(\overline{\chi_{\{1,\ldots,p\}}})$
$R = \{\lambda\overline{x_n}.\psi_1 = \lambda\overline{x_n}.z_1(\overline{\chi_{R_1}}), \ldots, \lambda\overline{x_n}.\psi_m = \lambda\overline{x_n}.z_m(\overline{\chi_{R_m}})\}$
$\rho = \{x = \lambda\overline{y_p}.\phi(z_1(\overline{y_{R_1}}), \ldots, z_m(\overline{y_{R_m}}))\}$

References

1. V. Danos and L. Regnier. The structure of multiplicatives. *Archive for mathematical logic*, (28):181–203, 1989.
2. Gilles Dowek. Third order matching is decidable. In *7th Annual IEEE Symposium of Logic in Computer Science*, pages 2–10, 1992.
3. Dov Gabbay. *Labelled Deductive Systems.* Oxford University Press, 1996.
4. Jean Gallier and Wayne Snyder. Designing unification procedures using transformations: a survey. In *Workshop Logic For Computer Science.* MSRI Berkeley, 1989.
5. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
6. W. Goldfard. The undecibility of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.
7. G. Huet. *Constrained Resolution: A Complete Method for Higher-Order Logic.* PhD thesis, Case Western Reserve University, 1972.
8. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.
9. J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, (65):154–170, 1958.
10. Jordi Levy. Linear second-order unification. *Rewriting Techniques and Applications*, 1996.
11. Xavier Lloré and Glyn Morrill. Difference lists and difference bags for logic programming of categorial deduction. In *Proceedings of the Sociedad Española para el Procesamiento del Lenguaje Natural*, pages 115–129, 1995.
12. Michael Moortgat. Labelled deductive systems for categorial theorem proving. In *Proceedings Eight Amsterdam Colloquium*, pages 403–424, Amsterdam, 1992. Institute of Language, Logic and Information, Universiteit van Amsterdam.
13. Glyn Morrill. *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Publishers, 1994.
14. Glyn Morrill. Clausal proofs and discontinuity. *Bulletin of the Interest Group in Pure and Applied Logics*, 3(2–3):403–427, 1995.

15. Glyn Morrill. Higher-order linear logic programming of categorial deduction. In *Proceedings ACL*, pages 133–140, 1995.
16. Glyn Morrill. Memoisation of categorial proof nets: parallelism in categorial processing. In Michele Abrusci and Claudia Casadio, editors, *Proof and Linguistic Categories: Proceedings 1996 Roma Workshop*, pages 157–169. Università di Bologna, 1996. To appear in S. Manandhar, W. Nutt and G. Lopez (eds.), Springer-Verlag.
17. F. Pereira and D. Warren. Parsing as deduction. In *Proceedings ACL*, pages 132–144, 1983.
18. Dirk Roorda. *Resource Logics: Proof-theoretical Investigations.* PhD thesis, Universiteit van Amsterdam, 1991.
19. Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, (8):101–140, 1989.