RICHARD T. OEHRLE

TERM-LABELED CATEGORIAL TYPE SYSTEMS*

1. Generalized Compositionality

Through language, we are able to assign symbolic analyses to linguistic entities – physical objects and events – whose complexity has no intrinsic upper bound. Such symbolic analyses are abstract, since a single physical entity can support distinct analyses. Yet we have partial intuitive access to the properties of these analyses through their projections in different 'dimensions', including the widely recognized and studied dimensions of *phonology*, *syntax*, and *semantics/pragmatics*. Each of these dimensions gives rise to a dimension-specific problem of *compositionality*:

> given an analysis of a linguistic entity *e*, which has, for a specific dimension *d* the projection *d*(*e*), how do the global properties of *d*(*e*) depend on the correlative properties of the components of *e*, together with their mode of composition?

But an additional question – which we call the problem of *generalized compositionality* – arises as well:

> how does composition in one dimension depend on composition in other dimensions?

There are many possible answers to this question and existing grammatical architectures instantiate some of them. The question deserves to be stud-

* Versions of this paper were presented in a number of different contexts, including: the OTS Categorial Grammar Colloquium, Utrecht, May 1991; University of Arizona Linguistics Colloquium, September 1991; Institute for Research in Cognitive Science, University of Pennsylvania, March 1992; CSLI workshop on categorial grammar and quantification, June 1992; MOL3, Austin, November 1992; Cognitive Science Brown Bag Series, University of Arizona, January 1993; Peripatetic Workshop on HPSG and Categorial Grammar, Columbus, 1993; Symposium on logic- and unification-based grammars, ESSLLI, Lisbon, August 1993. The paper owes a great deal to discussions with the audiences at these presentations, and to conversations about this material with Tom Cornell, Mark Johnson, Aravind Joshi, Ed Keenan, Manfred Krifka, Glyn Morrill, Fernando Pereira, Mark Steedman, Susan Steele, and Frans Zwarts. I am especially grateful to Michael Moortgat for discussion and clarification of the issues this paper addresses. The constructive criticism of two anonymous referees led to significant improvements. Remaining errors are of course mine. Finally, I would like to thank Merrill Garrett and Terry Langendoen and the University of Arizona International Programs Office for material support.

ied more systematically, however, so that we may gain deeper insight into the properties of multidimensional grammatical systems.

The family of formal systems known generically as Categorial Grammar provides a useful framework in which to undertake investigations of this kind. Categorial grammars easily accommodate composition in multiple dimensions; moreover, a variety of grammatical architectures compatible with composition in multiple dimensions can be simulated within the general categorial framework, making comparison in a common framework a possibility [32, 34, 35]. Finally, the general tenets of the family of categorial grammars are not rigidly fixed: the general perspective is open to innovation.

In what follows, we begin with a characterization of the general form of categorial grammars that accommodates a broad range of grammatical systems. We then investigate how three of the more prominent grammatical systems in this family accommodate quantifier types and quantificational scope ambiguities, a problem prominent in the categorial literature since Ajdukiewicz's work [2]. Analysis of this problem leads to the study of term-labeled type-deduction. The interest of this study centers on the division of deductive labor between the types and the terms that label them: some familiar systems arise as special cases, but new solutions to the problems of generalized compositionality arise as well.

2. Categorial Grammars as Deductive Systems

A categorial grammar is standardly determined by the following information:

- a finite set $At(T)$ *of primitive types*
- a finite set Ω of *type-forming operators*
- a finitely axiomatizable *type-calculus* C
- a finite vocabulary V
- an *initial type-assignment* τ

Together, the primitive types and the type-forming operators determine a *type-language* T. The type-calculus characterizes a set of *valid type structures* of the form $\Sigma(t_1, \ldots, t_k) \vdash t_0$, (where $\Sigma(t_1, \ldots, t_k)$ is a bracketed sequence of types, or a concatenation of types, or some other structure which is definable over both elements of the type-language and elements of V). The initial type-assignment τ: $V \rightarrow Pow(T)$ associates each element $v \in V$ with a non-empty subset of types in T. When the type $t \in \tau(v)$, we write $v \vdash t$.

Now, these assumptions determine a relation between complex structured expressions and types in a natural way: we extend the initial type-assignment τ to a complex expression $\Sigma(v_1, \ldots, v_k)$ formed from the atomic elements of the vocabulary $v_1, \ldots, v_k$ by the rule:

> $\Sigma(v_1, \ldots, v_k) \vdash t_0$ iff there are types $t_1, \ldots, t_k$ with $v_1 \vdash t_1, \ldots, v_k \vdash t_k$ such that $\Sigma(t_1, \ldots, t_k) \vdash t_0$ is valid according to the type-calculus C.

Categorial systems defined in this fashion can be compared in three ways, according to

- the deducibility relation between types that they characterize;
- the relation between expressions and types they characterize;
- their Curry–Howard properties: the correspondence they determine between proofs and λ-terms.

The interaction of these properties is instructive. In the sequel, we will consider three well-known categorial systems: **AB**, **L**, and **LP**. All three examples have type-systems which are positive logics, with type-constructors corresponding to implication and (in the cases of **L** and **LP**) conjunction. The sequent perspective of Gentzen [13] provides an illuminating picture of the basic categorial landscape.[1]

3. Sequent Calculi

We take a sequent, here, to consist of a pair $\langle \Gamma, B \rangle$ of structured sets of types, characteristically written $\Gamma \vdash B$.[2] Γ is called the *antecedent* and B the *succedent*. The sequents we shall be concerned with require that the succedent consist of exactly one type, and that the antecedent consist of a nonempty sequence of types $A_1, \ldots, A_k$ $(1 \leqslant k)$.

In logical contexts, a sequent $\Gamma \vdash B$ counts as true on an appropriate interpretation v of the types in Γ and B if, when v makes every type in Γ true, it makes B true. In grammatical contexts, we interpret the sequent $\Gamma \vdash B$ as meaning that any expression corresponding to the structured set of types Γ is assigned the type B. On both interpretations, the identity axiom $A \vdash A$ and the *Cut rule* shown below are true under any interpretation v.

[1] For a sketch of this landscape as a whole, and the place in it of the three categorial systems discussed below, see [29, 30].

[2] We use upper-case Greek letters as variables for structured sets of types (possibly empty unless their being so conflicts with the appropriate definition of sequent) and upper-case Roman letters for individual types.

$$\frac{\Gamma \vdash B \quad \Delta, B, \Theta \vdash C}{\Delta, \Gamma, \Theta \vdash C}$$

All the systems we shall discuss contain this identity axiom and the Cut rule. What distinguishes these systems from each other are the other inference rules they contain, including the *logical rules* governing the behavior of type-constructors and *structural rules* governing the resources that the sequents themselves are assumed to provide.

4. Curry–Howard Morphisms: Proofs → Terms

For categorial type-languages containing only implicational and product type-constructors, a rule which associates each primitive type A of the categorial type language with a corresponding type $typ(A)$ in a typed λ-calculus induces an association between the full set of categorial types and λ-types:

- associate each implicational type with domain A and codomain B (such as $A \rightarrow B$, $A\backslash B$, or B/A) with the λ-type $typ(A) \rightarrow typ(B)$ – that is, the type of functions from $typ(A)$ to $typ(B)$;
- and associate the product type with first projection A and second projection B with the pairing of $typ(A)$ and $typ(B)$.

On the basis of this type-correspondence, it is possible to associate with each valid sequent a corresponding λ-term in a way defined by its proof. In the pure type-calculus itself, we are not concerned with the interpretation of any particular lexical element (since in the pure system there are no lexical assumptions), and in a given proof, we associate each atomic type A with a variable λ-term of the appropriate type. We indicate such an association by pairing the atomic type in question (for example, A) and the variable in question (for example u), by writing $\langle A, u \rangle$.

We may annotate the postulates of a type calculus to indicate how the λ-terms associated with the types of the sequent in the conclusion of each inference rule depend on the λ-terms associated with the types of its premise-sequent or premise-sequents.

The Curry–Howard isomorphism between natural deduction proofs in the positive intuitionistic propositional calculus and λ-terms [20, 15] reveals a deep underlying similarity between two formal systems that were constructed for different purposes. Adapting the details of the Curry–Howard isomorphism to the present context does not preserve the isomorphism: sequent calculus formulations can associate different proofs with the same λ-term; moreover, implicational logics without contraction and weakening – such as the ones discussed below – do not provide proofs for every λ-term. These mismatches provide fuel for current research efforts in a variety of directions: proof systems which combine the elegance of natural deduction and the clarity of sequent calculus [14, 27, 28, 42, 46]; subsystems of the λ-calculus which allow a Curry–Howard morphism to be defined for particular subsystems of the positive intuitionistic propositional calculus [6, 8].

The importance here of Curry–Howard morphisms lies in two related points. First, the fact that Curry–Howard morphisms associate *proofs* to terms means that they embody in the most direct way possible the Fregean principle of compositionality. Second, the Curry–Howard framework provides a simple account of the sources of ambiguity: one source of ambiguity involves non-uniqueness of syntactic or semantic lexical type-assignment; a second source of ambiguity involves multiplicity of proofs. These two related properties have immediate consequences for the assessment of systems of grammatical analysis, as we shall see in the examples of categorial systems discussed below.

5. Examples

5.1. AB

The classical type-system **AB** introduced by Ajdukiewicz [2] and formalized by Bar-Hillel [3] corresponds to the implicational logic which results from splitting the implicational type-constructor $\rightarrow$ into the two directionally sensitive variants \ and /, and introducing for each variant a rule of inference governing its behavior in sequent antecedents. This yields the following set of postulates:

identity $A \vdash A$

$$\text{cut}\quad \frac{\Gamma \vdash A \quad \Lambda, A, \Delta \vdash B}{\Lambda, \Gamma, \Delta \vdash B}$$

$$L/\quad \frac{\Gamma \vdash A \quad \Delta, B, \Lambda \vdash C}{\Delta, B/A, \Gamma, \Lambda \vdash C}$$

$$L\backslash\quad \frac{\Gamma \vdash A \quad \Delta, B, \Lambda \vdash C}{\Delta, \Gamma, A\backslash B, \Lambda \vdash C}$$

If we add λ-terms to types in accordance with the Curry–Howard principles, the postulates take the following form:

identity $\langle A, u\rangle \vdash \langle A, u\rangle$

$$\text{cut}\ \frac{\Gamma \vdash \langle A, u\rangle \quad \Lambda, \langle A, u\rangle, \Delta \vdash B}{\Lambda, \Gamma, \Delta \vdash B}$$

$$L/\ \frac{\Gamma \vdash \langle A, u\rangle \quad \Delta, \langle B, t(u)\rangle, \Lambda \vdash C}{\Delta, \langle B/A, t\rangle, \Gamma, \Lambda \vdash C}$$

$$L\backslash\ \frac{\Gamma \vdash \langle A, u\rangle \quad \Delta, \langle B, t(u)\rangle, \Lambda \vdash C}{\Delta, \Gamma, \langle A\backslash B, t\rangle, \Lambda \vdash C}$$

In the discussion to follow, we sometimes use the pure syntactic system illustrated first and sometimes use the system that incorporates the Curry–Howard assignment. As the discussion progresses, the latter will become increasingly prominent.

A sequent is *valid*, relative to this type calculus, if it is an axiom instance or is derivable from valid premises according to one of the inference rules Cut, $L/$, or $L\backslash$. A proof of the validity of a sequent is easily displayed in the form of a *proof-tree*, a tree whose root is labeled with the sequent proved, whose leaves consist of instances of the identity axiom, and whose interior nodes consist of conclusions (looking up) or premises (looking down) of inference rules. Below are proofs of the two syntactic sequents $B/A, A \vdash B$ and $A, A\backslash B \vdash B$, sometimes referred to as *forward application* and *backward application*, respectively.

$$\frac{A \vdash A \quad B \vdash B}{B/A, A \vdash B}L/ \qquad \frac{A \vdash A \quad B \vdash B}{A, A\backslash B \vdash B}L\backslash$$

The grammatical applications of these rules arise from the fact that we regard an expression of type B/A (respectively, type $A\backslash B$) as having the property that the result of concatenating it with an expression of type A to its right (respectively, its left) is an expression of type B. Consequently, if we adjoin to an **AB** type calculus whose type language consists of the primitive types *np* and *s*, the simple lexicon {Zim, Yim, caught} and the lexical type assignment below, we may prove (by the natural extension of the lexical type assignment function described above in §2) that the string Zim caught Yim can be assigned the type *s*. Moreover, if Zim is assigned the λ-term z, Yim the λ-term y, and caught the λ-term c, our proof assigns the type *s* associated with the string Zim caught Yim to the λ-term $(c(y))(z)$:

$$\begin{array}{r}\text{Zim} \vdash \langle np, z\rangle \\ \text{Yim} \vdash \langle np, y\rangle \\ \text{caught} \vdash \langle (np\backslash s)/np, c\rangle\end{array}$$

$$\dfrac{\dfrac{\langle np, y\rangle \vdash \langle np, y\rangle \quad \dfrac{\langle np, z\rangle \vdash \langle np, z\rangle \quad \langle s, (c(y))(z)\rangle \vdash \langle s, (c(y))(z)\rangle}{\langle np, z\rangle, \langle np\backslash s, c(y)\rangle \vdash \langle s, (c(y))(z)\rangle}}{\langle np, z\rangle, \langle (np\backslash s)/np, c\rangle, \langle np, y\rangle \vdash \langle s, (c(y))(z)\rangle}}{\text{Zim caught Yim} \vdash \langle s, (c(y))(z)\rangle}$$

5.1.1. *Remarks*

In any Gentzen-style sequent proof, the leaves of the proof tree pair up types. As with all the proof systems we shall consider here, there are **AB** sequents which can be proved in more than one way and among whose proofs we find different leaf-pairings of types. A simple example (due to van Benthem [5]) is the sequent $s/s,\ s, s\backslash s \vdash s$, which can be proved in two essentially different ways, depending on whether the second term of the antecedent – the type s – is paired with the right-hand sub-type of the type s/s on its left or with the left-hand sub-type of the type $s\backslash s$ on its right. We can indicate these two admissible pairings by attaching common subscripts to the subformulas which are ultimately paired, as illustrated below:

$$s_1/s_2, s_2, s_1\backslash s_0 \vdash s_0$$

$$s_0/s_1, s_2, s_2\backslash s_1 \vdash s_0$$

Every valid **AB** sequent proof induces a pairing in this way of subformulas. But not every pairing of subformulas corresponds with an **AB** proof. For example, the subformulas of the sequents below can be paired up uniquely in obvious ways:

$$a/b, b/c \vdash a/c$$

$$a/b \vdash (a/c)/(b/c)$$

$$a\backslash(b/c) \vdash (a\backslash b)/c$$

$$(a\backslash b)/c \vdash a\backslash(b/c)$$

None of these sequents is valid in **AB**. Nevertheless, they are all valid in the type calculus **L**, which we now turn to.

5.2. L

L is Lambek's associative syntactic calculus [23]. It extends the type language of **AB** by the addition of a product type-constructor '·'; it retains the **AB** characterization of sequents and all the **AB** postulates; the major innovation is that it extends the set of postulates so that there are inference rules governing the behavior of each of the type-constructors with respect to both sequent antecedents and sequent succedents. The type calculus that results may be formulated as follows:

$$\text{identity} \quad \langle A, u\rangle \vdash \langle A, u\rangle$$

$$\text{cut} \ \frac{\Gamma \vdash \langle A, u\rangle \quad \Lambda, \langle A, u\rangle \Delta \vdash B}{\Lambda, \Gamma, \Delta \vdash B}$$

$$L/ \ \frac{\Gamma \vdash \langle A, u\rangle \quad \Delta, \langle B, t(u)\rangle, \Lambda \vdash C}{\Delta, \langle B/A, t\rangle, \Gamma, \Lambda \vdash C} \qquad \frac{\Gamma, \langle A, u\rangle \vdash \langle B, t\rangle}{\Gamma \vdash \langle B/A, \lambda u.t\rangle} \ u \notin \Gamma \quad R/$$

$$L\backslash \ \frac{\Gamma \vdash \langle A, u\rangle \quad \Delta, \langle B, t(u)\rangle, \Lambda \vdash C}{\Delta, \Gamma, \langle A\backslash B, t\rangle, \Lambda \vdash C} \qquad \frac{\langle A,u\rangle, \Gamma \vdash \langle B, t\rangle}{\Gamma \vdash \langle A\backslash B, \lambda u.t\rangle} \ u \notin \Gamma \quad R\backslash$$

$$L\cdot \ \frac{\Gamma, \langle A, \pi_L(t)\rangle, \langle B, \pi_R(t)\rangle, \Delta \vdash C}{\Gamma, \langle A \cdot B, t\rangle, \Delta \vdash C} \qquad \frac{\Gamma \vdash \langle A, \pi_L(t)\rangle \quad \Delta \vdash \langle B, \pi_R(t)\rangle}{\Gamma, \Delta \vdash \langle A \cdot B, t\rangle} \ R\cdot$$

Every valid **AB** sequent is valid in **L**, but there are many **L**-valid sequents which are not provable in **AB**. In the following representative list of **L**-valid sequents, only the Application rules hold in **AB**:

Some L-valid sequents

Application	$\langle A, u\rangle, \langle A\backslash B, t\rangle \vdash \langle B, t(u)\rangle$
	$\langle B/A, t\rangle, \langle A, u\rangle \vdash \langle B, t(u)\rangle$
Lifting	$\langle A, u\rangle \vdash \langle (B/A)\backslash B, \lambda u.t(u)\rangle$
	$\langle A, u\rangle \vdash \langle B/(A\backslash B), \lambda u.t(u)\rangle$
Co-division	$\langle A/B, t\rangle \vdash \langle (A/C)/(B/C), \lambda u.\lambda v.t(u(v))\rangle$
	$\langle B\backslash A, t\rangle \vdash \langle (C\backslash B)\backslash(C\backslash A), \lambda u.\lambda v.t(u(v))\rangle$
Contra-division	$\langle A/B, u\rangle \vdash \langle (C/A)\backslash(C/B), \lambda t.\lambda v.t(u(v))\rangle$
	$(B\backslash A, u) \vdash \langle (B\backslash C)/(A\backslash C), \lambda t.\lambda v.t(u(v))\rangle$
Swapping	$\langle (B\backslash A)/C, t\rangle \vdash \langle B\backslash(A/C), \lambda u.\lambda v.((t(v))(u))\rangle$
	$\langle B\backslash(A/C), w\rangle \vdash \langle (B\backslash A)/C, \lambda x.\lambda y.((w(y))(x))\rangle$
Currying	$\langle A/(B \cdot C), t\rangle \vdash \langle (A/C)/B, \lambda x.\lambda y.t(\langle x, y\rangle)\rangle$
	$\langle (A/C)/B, t\rangle \vdash \langle A/(B \cdot C), \lambda u.(t(\pi_L(u)))(\pi_R(u))\rangle$
	$\langle (B \cdot C)\backslash A, t\rangle \vdash \langle C\backslash(B\backslash A), \lambda x.\lambda y.t(\langle y, x\rangle)\rangle$
	$\langle C\backslash(B\backslash A), t\rangle \vdash \langle (B \cdot C)\backslash A, \lambda u.t(\pi_R(u), \pi_L(u))\rangle$

The richer type-calculus of **L** has consequences even for the analysis of simple sentences. For example, relative to the same 3-word lexicon and type-assignment used in the **AB** examples above, **L** provides more than one proof of the assignment of the string Zim caught Yim to the type s. The **AB**-proof exhibited earlier is an **L**-proof as well, but in addition, we have the proof (not valid in **AB**):

$$\dfrac{\dfrac{\dfrac{\dfrac{np \vdash np \qquad \dfrac{np \vdash np \qquad s \vdash s}{np, np\backslash s \vdash s}}{np, (np\backslash s)/np, np \vdash s}}{np, (np\backslash s)/np \vdash s/np}}{(np\backslash s)/np \vdash np\backslash(s/np)} \qquad \dfrac{np \vdash np \qquad \dfrac{np \vdash np \qquad s \vdash s}{s/np, np \vdash s}}{np, np\backslash(s/np), np \vdash s}}{np, (np\backslash s)/np, np \vdash s}$$

Together with the earlier proof, this proof suggests that **L** ignores differences among different binary bracketings of strings. In fact, **L** also disregards differences between 'functor' (that is, a type of the form A/B or $B\backslash A$) and 'argument' (such as type B in the case of the functor-types just mentioned), since the rules $R/$ and $R\backslash$ countenance the reversal of functor/argument relations: corresponding to the validity of the application sequent B, $B\backslash A \vdash A$ (where the 'functor' is the second element of the antecedent) is the validity of the sequent $A/(B\backslash A)$, $B\backslash A \vdash A$ (where the first element of the antecedent is the 'functor'). These observations concerning bracketing and functor/argument structure form the basis of Buszkowski's notion of the 'functional completeness' of **L**: see [7, 33]. for discussion.

The flexibility of **L** lies in its approach to constituency; all of the postulates of **L** are order-preserving. The postulates of **L** also depend on the adjacency of active types, so no action at a distance is countenanced. As a consequence, while **L** counts the Swapping rules $a\backslash(b/c) \vdash (a\backslash b)/c$ and $(a\backslash b)/c) \vdash a\backslash(b/c)$ as valid, there is no valid rule in **L** which relates $(a/b)/c$ and $(a/c)/b)$. This is reasonable from the point of view of preserving string order: the two types in question do not have the same properties with respect to the order of string combination. On the other hand, it is unreasonable if we wish to regard as equivalent two binary functors which differ only in the order in which they combine with their arguments. The contrast between these two cases reveals one of the characteristic properties of **L**: it cannot disentangle string conditions and type-structure.

One way to disentangle the two is to disregard string conditions completely. This is the difference between **L** and **LP**.

5.3. LP

The Lambek/van-Benthem calculus **LP** results from the addition of the structural rule Permutation to the postulates of **L**. In this system, the sequents $A/B \vdash B\backslash A$ and $B\backslash A \vdash A/B$ are valid, as the following proofs illustrate:

$$\begin{array}{lcc l}
 & \vdots & \vdots & \\
 & \underline{A/B, B \vdash A} & \underline{B, B\backslash A \vdash A} & \\
\text{Permutation} & \underline{B, A/B \vdash A} & \underline{B\backslash A, B \vdash A} & \text{Permutation} \\
R\backslash & A/B \vdash B\backslash A & B\backslash A \vdash A/B & R/
\end{array}$$

As a result, the types A/B and $B\backslash A$ fall together and it is convenient to represent them both as $B \rightarrow A$.[3]

Similarly, the addition of the structural rule of Permutation makes the product operator commutative, so that we have $A \cdot B \rightarrow B \cdot A$:

$$\begin{array}{ll}
 & \underline{B \vdash B \quad A \vdash A} \\
R\cdot & \underline{B, A \vdash B \cdot A} \\
\text{Permutation} & \underline{A, B \vdash B \cdot A} \\
L\cdot & A \cdot B \vdash B \cdot A
\end{array}$$

To distinguish the commutative product in **LP** from the noncommutative product in **L**, we write the commutative product as '$\otimes$' and the noncommutative product as '$\cdot$'.

One of the consequences of Permutation is that the symmetrical pairs of valid sequents found in **AB** and **L** fall together into a single valid **LP** sequent, in which $\alpha\backslash\beta$ is replaced by $\alpha \rightarrow \beta$ and β/α is replaced by $\alpha \rightarrow \beta$. In general, however, systematically undoing the results of such a transformation shows that such **LP** sequents represent a wider class of **L** sequents, some of which may be **L** valid and some of which may not be.

For example, consider the **LP** analogues of Swapping and Lifting. In **L**, there are two forms of Swapping:

$$(B\backslash A)/C \vdash B\backslash(A/C) \quad \text{and} \quad B\backslash(A/C) \vdash (B\backslash A)/C$$

By replacing both $\alpha\backslash\beta$ and β/α by $\alpha \rightarrow \beta$ in these two sequents, we have the representations:

$$C \rightarrow (B \rightarrow A) \vdash B \rightarrow (C \rightarrow A) \quad \text{and} \quad B \rightarrow (C \rightarrow A) \vdash C \rightarrow (B \rightarrow A)$$

[3] Technically, this changes the type language by collapsing two type-constructors into one. This is not important here, but it will be worth remembering later.

Unlike the two **L**-valid instances of Swapping, however, these two sequents are alphabetic variants.

But they also represent (by the same transformation) sequents which are not **L**-valid. In particular, they represent the fact that the sequent $(A/B)/C \vdash (A/C)/B$ (among others) is valid in **L + Permutation**, but not valid in **L**.

Similarly, the **LP** analogue of Lifting is $A \vdash (A \to B) \to B$. But this represents not just the two **L**-valid forms of Lifting $A \vdash (B/A)\backslash B$ and $A \vdash B/(A\backslash B)$ but also such **L**-invalid sequents as $A \vdash B/(B/A)$.

5.4. *Remarks*

There are many other examples of categorial logics based on products and implicational operators. For a survey, see [29, 30]. We turn shortly to the investigation of how these systems accommodate quantification and quantificational scope ambiguities. As a preliminary to this discussion, it is helpful to survey some properties of proofs in the three systems under consideration here.

6. Decidability of AB, L, LP

All three of the categorial type-calculi discussed above are *decidable*, in the following sense: for each calculus C, there exists an algorithm which determines, for an arbitrarily chosen C-sequent Σ, whether or not Σ is provable in C. The insight on which these algorithms are based originates with Gentzen [13]; its connection with categorial grammar is due to Lambek.

Gentzen observed that for certain logical systems,[4] the structural rule Cut is *eliminable*, in the sense that any sequent proof of a sequent Σ involving Cut can be transformed to a *Cut-free* proof of Σ – that is, a proof of Σ in which no step involves Cut.

Proofs in the Cut-free versions of **AB** and **L** have two interesting properties. First, they have the *subformula property*: all the types which occur in the proof of a sequent Σ are subformulas of Σ.[5] Second, in the absence of Cut, the inference rules of **AB** and **L** are *complexity increasing*: in every

[4] In particular, the calculi *LK* and *LJ* which he introduced and which correspond, respectively, to classical and intuitionistic first-order logic.

[5] In the obvious sense of *subformula*: the only subformula of an atomic type A is A itself; the subformulas of complex types of the form A/B (respectively, $B\backslash A$ or $A \cdot B$) consist of A/B (respectively, $B\backslash A$ or $A \cdot B$), together with the subformulas of A and the subformulas of B.

case, the conclusion of an inference rule contains one more occurrence of a type-forming operator than the premises.

For a type-calculus C to have the subformula property means that the search space involved in determining whether or not any C-sequent Σ is provable in C contains only types constructible from the atomic types contained in Σ. The complexity-increasing property of inference rules means that the search space is bounded and decreases as we move from conclusions to premises in the search for a valid proof. This situation lays the basis for a simple inductive argument, whose structure is clarified by the following two definitions.

The *degree of a type* is the number of type-forming operators it contains. For example, primitive types are of degree 0; the degree of types of the form A/B or $B\backslash A$ or $A \cdot B$ is the sum of three terms: degree(A) + degree(B) + 1.

The *degree of a sequent* is the sum of the degrees of the occurrences of antecedent types and the consequent type. For example, the degree of the sequent $A/B, B \vdash A$ is the sum of the three terms: degree(A/B) + degree(B) + degree(A). (If A and B are both atomic types, the degree of this sequent is obviously 1; otherwise, it is greater.)

It is a simple matter to tell whether or not a sequent is an instance of the identity axiom or not. Moreover, in the Cut-free variants of **AB** and **L**, all the inference rules are *degree-increasing*, in the sense that the degree of the conclusion strictly exceeds the sums of the degrees of the premises. Thus, the only provable sequents of degree 0 are axiom instances. And, since we can decide whether a sequent of degree 0 is an axiom-instance, we thus have a solution to the decidability question for degree-0 sequents.

Now make the inductive assumption that we have a solution to the decidability question for degree-n sequents. Any degree-$n + 1$ sequent is derivable from premises of strictly lesser degree in one of only finitely-many ways: the final rule of inference must introduce one of the principal type-forming operators in one of the antecedent-types or in the succedent type. Since there are only finitely many types in any sequent, there are only finitely many possible ways in which a given sequent of degree $n + 1$ can be derived, and each of these possibilities depends on premises of strictly lesser degree. In each case, then, it is decidable (by the inductive assumption) whether the premises required are provable. If they are, so is the sequent in question. On the other hand, if all of the finitely-many ways of introducing a principal type-forming operator among the elements of the antecedent or the succedent are considered and in no case is it found that the required premises are themselves provable, then the sequent in question has no proof. Thus, the Cut-free variants of **AB** and **L** are decidable. And, since every proof in **AB** or **L** (respectively) has a Cut-free variant, it follows that **AB** and **L** are decidable as well.

The case of **LP** is the same, except that at the inductive step, we have to take into account the possibility that the final step of the proof of the degree-$n + 1$ sequent in question may have been the structural rule Permutation. This introduces no fundamental difference, since there can be only finitely many permutations to examine (since sequents contain only finitely many types). Thus, **LP** is decidable as well.

As an example, consider the following sequents, none of which is an axiom instance:

Right Application:	$A/B,\ B \vdash A$
Left Lifting:	$B \vdash (A/B)\backslash A$
Backwards Application:	$B,\ A/B \vdash A$
Lowering:	$(A/B)\backslash A \vdash B$

The Right Application sequent is not an axiom instance and has only one occurrence of a principal type-forming operator; this occurrence could only be introduced by the rule $L/$, as follows:

$$\frac{B \vdash B \qquad A \vdash A}{A/B, B \vdash A}$$

Since this inference rule appears among the postulates of **AB**, of **L**, and of **LP**, this sequent is valid in all three systems.

The Left Lifting sequent $B \vdash (A/B)\backslash A$ is also not an axiom instance and contains only one occurrence of a principal type-forming operator, namely, the principal type-forming operator of the succedent type $(A/B)\backslash A$, which has the form $\beta\backslash\alpha$. This type-forming operator could only have been introduced by the rule $R\backslash$. This rule is lacking in the calculus **AB**, and thus this sequent is not provable in **AB**. On the other hand, $R\backslash$ is a postulate of both **L** and **LP**. And in these two systems, applying the inference rule $R\backslash$ to the premise sequent $A/B,\ B \vdash A$ – which, as we have just seen, is provable in both **L** and **LP** – yields the Left Lifting sequent as the end-sequent. Thus Left Lifting is provable in **L** and **LP**, but not provable in **AB**.

The sequent $B,\ A/B \vdash A$ is not provable in either **AB** or **L**, since the only possible Cut-free proof would introduce at the final step both the antecedent type A/B and a further non-empty sequence of antecedent types to its right, matching Γ in the left-hand premise of the inference rule $L/$. But no such Γ occurs in the antecedent. Thus, we have exhausted the search space for a proof in either system without finding one. In **LP**, however, there is another possible last proof-step: Permutation. That is, the final step can take the form:

$$\frac{A/B, B \vdash A}{B, A/B \vdash A}$$

And since the premise A/B, $B \vdash A$ is provable in **LP**, so is the conclusion. Thus, Backwards Application is provable in **LP**, but neither in **AB** nor in **L**.

Finally, the Lowering sequent $(A/B)\backslash A \vdash B$ is provable in none of the systems examined here. Since both the antecedent and succedent consist of single types, Permutation has no effect here. Thus, the last step in a proof of Lowering would have to be $L\backslash$, since the only principal type-forming operator in any sequent type is the occurrence of $\backslash$ in the antecedent type. But this cannot be the last step, since it would have to introduce additional antecedent types which are not present.

These simple examples show how it is possible to demonstrate that a given sequent is or is not provable in the systems under examination here. These proof-theoretic techniques are especially useful in the investigation of the Curry–Howard properties of these different systems when they contain types corresponding to generalized quantifiers.

7. Proof Structures and λ-Terms

For our purposes later on, there are two relations between proof structures and λ-terms which are important.

First, the Cut Elimination theorem respects the Curry–Howard assignment of λ-terms to proofs, in the sense that the transformation from a proof containing one or more Cut inferences to a Cut-free proof does not change (up to β-equivalence) the λ-term assigned to the endsequent. We will not prove this here, but refer the reader to [17, 27]. This result means that insofar as we are interested in the readings associated with a particular sequent in a given system, we need only consider the readings associated with it by Cut-free versions of the system, which in the cases under consideration here are decidable. Thus, if a sequent is provable in one of the systems investigated here, the range of readings associated with it is determinable as well. This fact is reassuring.

Second, in the Cut-free systems, the λ-term associated with a valid endsequent by the Curry–Howard correspondence is determined (up to alphabetic variance) by two factors:

- the bindings of types at axiom leaves;

- the form of the consequent term.

Proof: Note first that this property is trivial for axioms. For the inductive step, there are two cases to consider. In the inference rules $L/$, $L\backslash$, $L\cdot$ which introduce antecedent connectives, the Curry–Howard term is inherited directly from one of the premises. Thus, the inductive step in these cases is immediate. For the right rules $R/$, $R\backslash$, $R\cdot$, we make the inductive assumption that the claims to be shown hold for the premises and show that this determines the Curry–Howard term of the conclusion. For example, if we know the axiom bindings for the two sequents $\Gamma \vdash A$ and $\Delta \vdash B$, and if these bindings and the form of A and B determine the association of the λ-term u with the first of these and the association of the λ-term v with the second, then these same axiom bindings and the form of the type $A{\cdot}B$ determine the association of the λ-term $\langle u, v\rangle$ with the endsequent $\Gamma, \Delta \vdash A{\cdot}B$ derived from these two sequents by the rule $R\cdot$. A similar argument provides the inductive step in the case of the rules $R/$ and $R\backslash$.

This result is of interest in a variety of ways. First, the property of having the same axiom-leaf bindings (up to alphabetic variance) defines an equivalence relation among proofs. Defining the Curry–Howard mapping on the equivalence classes sharpens the correspondence between proofs and λ-terms. Second, the result places an upper bound on the number of distinct λ-terms that can be associated with a given endsequent by the Curry–Howard mapping. And these possibilities can be easily enumerated by examining the sub-formulas of the endsequent in question.

It is this last point which is of primary interest here, since it allows us to characterize the readings assigned to a given endsequent in a given calculus simply by stating the axiom bindings that it depends on. For example, van Benthem's example $s/s, s, s\backslash s \vdash s$ has two distinct readings (in any of the systems discussed above):

$$\frac{\langle s, u\rangle \vdash \langle s, u\rangle \qquad \dfrac{\langle s, v(u)\rangle \vdash \langle s, v(u)\rangle \qquad \langle s, t(v(u))\rangle \vdash \langle s, t(v(u))\rangle}{\langle s/s, t\rangle, \langle s, v(u)\rangle \vdash \langle s, t(v(u))\rangle}}{\langle s/s, t\rangle, \langle s, u\rangle, \langle s\backslash s, v\rangle \vdash \langle s, t(v(u))\rangle}$$

$$\frac{\langle s, u\rangle \vdash \langle s, u\rangle \qquad \dfrac{\langle s, t(u)\rangle \vdash \langle s, t(u)\rangle \qquad \langle s, v(t(u))\rangle \vdash \langle s, v(t(u))\rangle}{\langle s, t(u)\rangle, \langle s\backslash s, v\rangle \vdash \langle s, v(t(u))\rangle}}{\langle s/s, t\rangle, \langle s, u\rangle, \langle s\backslash s, v\rangle \vdash \langle s, v(t(u))\rangle}$$

We can characterize these readings more simply by assigning subscripts to the atomic subformulae and noting which subscripts are bound to one another at the axiom leaves:

Sequent	Bindings	λ-term
$\langle s_a/s_b, t\rangle, \langle s_c, u\rangle, \langle s_d\backslash s_e, v\rangle \vdash \langle s_f, term\rangle$	$b = c,\ a = d,\ e = f$	$term = v(t(u))$
$\langle s_a/s_b, t\rangle, \langle s_c, u\rangle, \langle s_d\backslash s_e, v\rangle \vdash \langle s_f, term\rangle$	$a = f,\ b = e,\ c = d$	$term = t(v(u))$

While this way of indicating the λ-terms that may be associated with a given endsequent is more succinct than actually exhibiting all the proofs that lead to these λ-terms, one must bear in mind that not all possible pairings of atomic subformulae correspond to possible proofs (in one system or another). In the next section, we turn to a detailed discussion of a class of examples bearing on this issue, namely the number of readings assignable (in different systems) to simple sentences containing quantifiers.

8. Quantifiers and Quantification

In the extensional fragment of Montague's *PTQ* [45], quantifiers (and terms generally) are assigned the syntactic type $t/(t/e)$ and are constrained to be interpreted as functions of type $\langle\langle e, t\rangle, t\rangle$. In the three systems under consideration here, this type assignment has a direct analog: quantifiers in subject position of declarative English sentences may be assigned the type $s/(np\backslash s)$, together with an interpretation on which quantifiers are functions from predicates (that is, functions from individuals to truth values) to truth values, in accordance with Generalized Quantifier Theory.

We may suppose, then, that if the interpretation of the noun detective is represented by the set D in a model $\mathcal{M}$, the expression every detective is interpreted in $\mathcal{M}$ by the generalized quantifier $\forall D$, which determines a truth value when combined with a 1-place predicate P, according to the rule: true iff $D \subseteq P$. And we may cast these suppositions in the form of the following lexical assumptions:

	lexical assumptions
every	$\vdash \langle (s/(np\backslash s))/n, \forall\rangle$
detective	$\vdash \langle n, D\rangle$
sneezed	$\vdash \langle np\backslash s, \lambda x.sneezed(x)\rangle$

And on these assumptions, we have the following proof, valid in **AB**, in **L**, and in **LP**, that every detective sneezed is assigned type s, with an acceptable interpretation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\langle np, x\rangle \vdash \langle np, x\rangle \qquad \langle s, sneezed(x)\rangle \vdash \langle s, sneezed(x)\rangle}{\langle np, x\rangle, \langle np\backslash s, \lambda x.sneezed(x)\rangle \vdash \langle s, sneezed(x)\rangle}}{\langle np\backslash s, \lambda x.sneezed(x)\rangle \vdash \langle np\backslash s, \lambda x.sneezed(x)\rangle \qquad \langle s, \forall D\lambda x.sneezed(x)\rangle \vdash \langle s, \forall D\lambda x.sneezed(x)\rangle}}{\langle n, D\rangle \vdash \langle n, D\rangle \qquad \langle s/(np\backslash s), \forall D\rangle, \langle np\backslash s, \lambda x.sneezed(x)\rangle \vdash \langle s, \forall D\lambda x.sneezed(x)\rangle}}{\langle (s/(np\backslash s))/n, \forall\rangle, \langle n, D\rangle, \langle np\backslash s, \lambda x.sneezed(x)\rangle \vdash \langle s, \forall D\lambda x.sneezed(x)\rangle}}{\text{every detective sneezed} \vdash \langle s, \forall D\lambda x.sneezed(x)\rangle}$$

Given that this proof exists, the information that it contains concerning axiom bindings can be displayed:

$$\cfrac{\langle (s^a/(np^b\backslash s^c))/n^d, \forall\rangle, \langle n^e, D\rangle, \langle np^f\backslash s^g, \lambda x.sneezed(x)\rangle \vdash \langle s^h, \forall D\lambda x.sneezed(x)\rangle}{\text{every detective sneezed} \vdash \langle s, \forall D\lambda x.sneezed(x)\rangle}$$

$$a = h,\ b = f,\ c = g,\ d = e$$

The bindings on the last line indicate the axiom pairings produced by the proof above.

This account of quantification is satisfactory for the monadic case in which a quantifier always bears the same syntactic position relative to its argument – satisfactory in the sense that it provides a type for quantifiers consistent with the principles of the Curry–Howard interpretation. But it does not extend easily to more complex cases, either in the extensional fragment of *PTQ* or in any of the three frameworks examined here.

8.1. *Quantification and 2-Place Predicates*

The interaction of quantifiers with 2-place predicates raises two basic questions. First, what types are to be assigned to the quantifiers and to the 2-place predicate, in such a way that syntactic composition is a consequence of general syntactic principles? Second, how are quantificational scope ambiguities to be accounted for?

The extensional fragment of Montague's *PTQ* contains a solution to both of these problems. He assigned transitive verbs a type – $(t/e)/(t/(t/e))$, in his syntactic type system – according to which transitive verbs are functions from (monadic) quantifiers to 1-place predicates. The corresponding type for English transitive verbs in the systems discussed here is

$$(np\backslash s)/(s/(np\backslash s)).$$

It is completely straightforward to see that the sequent

$$(np\backslash s)/(s/(np\backslash s)),\ s/(np\backslash s) \vdash np\backslash s$$

is valid, since this sequent has the form $A/B,\ B \vdash A$.

But what about interpretation? A simple transitive sentence such as every detective caught some thief has two interpretations, which we shall represent as a quantificational scope ambiguity:

$$\forall D\lambda d\exists T\lambda t.(d\ caught\ t)$$
$$\exists T\lambda t\forall D\lambda d.(d\ caught\ t)$$

Neither of these interpretations is assigned to the proof of the sequent corresponding to Montague's type assignment for every detective caught some thief. Instead, we have the interpretation

$$\forall D\lambda d.(d\ caught\ (\exists T))$$

This representation is not necessarily incorrect. Montague imposes the requirement that the interpretation of caught (and, *mutatis mutandis*, of every other extensional transitive verb) be equivalent, for some corresponding 2-place predicate $caught_*$ of type $(e, \langle e, t\rangle)$, to

$$\lambda\mathcal{Q}.\lambda d.\mathcal{Q}\lambda t.(d\ caught_*\ t)$$

This function, when applied to $\exists T$ reduces to

$$\lambda d.\exists T\lambda t.(d\ caught_*\ t),$$

which is the more standard representation exhibited earlier.

Quantifier scope ambiguities arise in Montague's system in an entirely different way, through the rule of Quantifying In, which involves combining a term-taking expression with a variable and then binding the variable later in the derivation.

Montague's account of quantification has a number of interesting, even admirable, properties:

- it is based on a single type for quantifiers;
- the syntax is driven by type-theoretic considerations;
- there is a principled relation between syntactic type and semantic interpretation.

But these properties impose a cost: in order to characterize quantificational interpretations, external mechanisms – meaning postulates and quantifying-in rules – are introduced *ad hoc*. Are these costs inevitable consequences in any system with a single type for quantifiers, a type-driven syntax, and a principled relation between syntactic types and semantic interpretation? To explore this question, we shall look first at the treatment of quantification afforded by the type-calculi **AB**, **L**, and **LP**.

8.2. **AB** *with Quantifiers*

In adding quantifiers to **AB**, one finds that every fact about quantification must be treated as *sui generis*. To see why, consider the types that are necessary to allow a generic quantifier (which we would like to type simply as $\langle Q, \mathcal{Q}\rangle$) to occur in various syntactic positions under different scopings.

Context	Syntactic type	λ-term
Subject, wide scope	$s/(np\backslash s)$	$\mathcal{Q}$
Object of a 2-place predicate narrow scope	$((np\backslash s)/np)\backslash(np\backslash s)$	$\lambda P.\lambda x.\mathcal{Q}\lambda y.P(y)(x)$
Subject of a 2-place predicate narrow scope	$(s/np)/((np\backslash s)/np)$	$\lambda P.\lambda y.\mathcal{Q}\lambda x.P(y)(x)$
Object of a 2-place predicate wide scope	$(s/np)\backslash s$	$\mathcal{Q}$
⋮	⋮	⋮

With each new argument position, new types are necessary. This hardly accords with the fact that natural language quantificational terms do not standardly vary both according to their position and according to their scope.

8.3. **L** *with Quantifiers*

The situation in **L** is slightly better. If we associate a quantifier with two types, one for subject position, and one for object position, the ambiguity of scope for simple English transitive sentences falls out:

Context	Syntactic type	λ-term
Subject	$s/(np\backslash s)$	$\mathcal{Q}$
Object	$(s/np)\backslash s)$	$\mathcal{Q}$

Here are the final steps of two distinct proofs of the sequent corresponding to every detective caught some thief, proofs assigning interpretations with different quantifier scopes to the consequent of the endsequent:

$$\frac{s/(np\backslash s), (np\backslash s)/np \vdash \left\langle \begin{matrix} s/np \\ \lambda t\forall D\lambda d(d\ caught\ t)\end{matrix}\right\rangle \qquad s \vdash \left\langle \begin{matrix} s \\ \exists T\lambda t\forall D\lambda d(d\ caught\ t)\end{matrix}\right\rangle}{s/(np\backslash s), (np\backslash s)/np, (s/np)\backslash s \vdash s}$$

$$\frac{(np\backslash s)/np, (s/np)\backslash s \vdash \left\langle \begin{matrix} np\backslash s \\ \lambda d\exists T\lambda t(d\ caught\ t)\end{matrix}\right\rangle \qquad s \vdash \left\langle \begin{matrix} s \\ \forall D\lambda d\exists T\lambda t(d\ caught\ t)\end{matrix}\right\rangle}{s/(np\backslash s), (np\backslash s)/np, (s/np)\backslash s \vdash s}$$

We can represent these two proofs in our notation of indexed atomic types as follows:

$$\text{every detective caught some thief} \vdash \langle s, \exists T\lambda t\forall D\lambda d(d\ caught\ t)\rangle$$
$$s^a/(np^b\backslash s^c), (np^d\backslash s^e)np^f, (s^g/np^h)\backslash s^i \vdash s^j$$
$$a = g;\ b = d;\ c = e;\ f = h;\ i = j$$

$$\text{every detective caught some thief} \vdash \langle s, \forall D\lambda d\exists T\lambda t(d\ caught\ t)\rangle$$
$$s^a/(np^b\backslash s^c), (np^d\backslash s^e)/np^f, (s^g/np^h)\backslash s^i \vdash s^j$$
$$a = j;\ b = d;\ c = i;\ e = g;\ f = h$$

What is intriguing about these alternative proofs is the fact that the quantificational scope ambiguities are consequences of the combinatorial properties of the type calculus **L**. Thus, unlike **AB**, it is not necessary to code each scope possibility with a separate, *ad hoc* type assignment. The difference between **L** and **AB** lies in two related properties. First, in **L**, a transitive verb (in English) may combine with its arguments in either order, because of the validity of the Swapping rules:

$$(np\backslash s)/np \vdash np\backslash(s/np); \qquad np\backslash(s/np) \vdash (np\backslash s)/np$$

Second, and equally important, is the fact that **L** enjoys the Division rule, whose importance for quantifier types may be illustrated by the sequent:

$$s/(np\backslash s) \vdash (s/np)/((np\backslash s)/np)$$

Thus a monadic subject quantifier of type $s/(np\backslash s)$, which combines with a verb phrase of type $np\backslash s$ to form a sentence of type s, automatically combines as well, in virtue of the principles of **L**, with a transitive verb of type $(np\backslash s)/np$ to form a 1-place predicate of type s/np. Neither of these rules is valid in **AB**. Their applicability here suggests the possibility of finding a type calculus in which quantifiers can be assigned a single type and in which scope ambiguities are consequences of basic and simple syntactic principles, rather than being the consequence of specially-designed rules introduced expressly for the purpose of dealing with scope.

The calculus **L**, however, does not match these specifications. First, even in simple transitive sentences, two basic types for quantifiers, rather than a single type, are necessary. Moreover, the useful properties of the Swapping rule are available only when a binary predicate is flanked on either side by its arguments: in SOV or VOS orders, there is no corresponding rule, since the combinatory rules of **L** respect the principle of adjacency and there is no possibility of combining subject and transitive verb across an intervening object in either the SOV or the VOS case.[6]

[6] An interesting attempt to overcome this difficulty can be found in Keenan's 'Semantic Case Theory' [22], although this approach is not based on a general syntactic type calculus.

8.4. **LP** *with Quantifiers*

In **LP**, quantifiers can be assigned a single type:

$$(np \to s) \to s$$

On the basis of this type, quantifier scope ambiguities fall out naturally as a consequence of the properties of the **LP** type calculus. What makes this possible is the fact that in **LP**, the Division rule is still valid and the Swapping rule is generalized to the following form:

$$A \to (B \to C) \vdash B \to (A \to C)$$

As a consequence of Division, a quantifier can combine with a predicate of any arity greater than or equal to 1. This solves the type-assignment problem. As a consequence of Swapping, quantifiers can combine with a predicate in any order. This solves the scope-ambiguity problem.

This pleasing picture is marred only by the fact that in **LP**, word order is irrelevant and the distinct sentences every detective called some schurk, some schurk called every detective, some detective called every schurk, and every schurk called some detective are assigned the same range of interpretations. Some – but not all! – of these possibilities are indicated below, where the axiom-leaf bindings of atomic types are indicated with superscripts:

every detective called some schurk $\vdash \langle s, \mathcal{Q}_d \lambda d \mathcal{Q}_c \lambda c(d \textit{ called } c)\rangle$
$(np^j - s^b) \to s^c, np^i_{\text{obj}} \to (np^j_{\text{subj}} \to s^a), (np^i \to s^a) \to s^b) \vdash s^c$

every detective called some schurk $\vdash \langle s, \mathcal{Q}_d \lambda c \mathcal{Q}_c \lambda d(d \textit{ called } c)\rangle$
$(np^j - s^b) \to s^c, np^i_{\text{obj}} \to (np^j_{\text{subj}} \to s^a), (np^j \to s^a) \to s^b) \vdash s^c$

every detective called some schurk $\vdash \langle s, \mathcal{Q}_c \lambda c \mathcal{Q}_d \lambda d(d \textit{ called } c)\rangle$
$(np^j - s^a) \to s^b, np^i_{\text{obj}} \to (np^j_{\text{subj}} \to s^a), (np^i \to s^b) \to s^c) \vdash s^c$

every detective called some schurk $\vdash \langle s, \mathcal{Q}_c \lambda d \mathcal{Q}_d \lambda c(d \textit{ called } c)\rangle$
$(np^i - s^a) \to s^b, np^i_{\text{obj}} \to (np^j_{\text{subj}} \to s^a), (np^j \to s^b) \to s^c) \vdash s^c$

8.5. *Control Properties*

The inadequacies of these accounts derive from inadequate *control* of properties across dimensions. The common difficulty in each case involves an overly rigid correspondence between properties in the string dimension and properties in the interpretive dimension.

In **LP**, flexibility of quantifier scope is tied to flexibility of string order.

This system provides the key ingredients to a solution of quantifier-typing and quantifier scope, but it overshoots the mark: along with the desired ambiguities, it yields unwanted readings as well.

In **L**, flexibility of quantifier scope is tied to string positions flanking predicates. This is insufficiently general, since the solution to quantifier-scope ambiguities is only available in special cases.

In **AB**, there is no inherent quantifier scope flexibility at all, just as there is no inherent combinatorial flexibility syntactically.

One possible solution to the problem of cross-dimensional control is to continue to tie syntactic combination and string properties tightly together in a system (like either **AB** or **L**) based on concatenation, but to introduce at the same time independent rules in the semantic dimension which do not depend on the same principles of adjacency. A fundamental step forward in this direction can be found in the work of Hendriks [16], who proposed a calculus of semantic types to generate scope ambiguities involving quantifiers and Boolean operators. The proposed calculus operates in a type-driven way on semantic representations generated by a GPSG-style phrase-structure grammar.

Moortgat [25, 26] shows that the semantic type-shifting principles proposed to generate the full range of interpretations have basic affinities with type-shifting principles found in **L** and **LP**. Moortgat proposes to treat quantificational scope ambiguities by the addition of a new type-constructor whose associated inference rules incorporate the effects of the **LP** structural rule Permutation in the semantic dimension, leaving the string dimension invariant. The resulting system is an elegant one, with many attractive properties [10].

Nevertheless, on both Hendriks's account and Moortgat's, scope ambiguities and the other forms of discontinuous dependency treated with them are still regarded as special cases whose accommodation calls for special means. But with the aid of these achievements, it is possible to find a type-theoretical framework in which quantificational scope ambiguities and related forms of discontinuous dependency are an *emergent property*, in the sense that a single theoretical framework which can be justified on the basis of simple phenomena where the property is not observable accounts automatically for the emergence of the property in question under more complex conditions.

There is in fact an attractive formal setting (whose instances will be referred to here as *term-labeled categorial type systems*) which is consistent with the approach of categorial grammar to the problem of generalized compositionality and in special cases of which quantificational scope ambiguities are an emergent property. Certain aspects of the resource-sensitivity of term-labeled type systems are expressed by the terms of a system of *string-term labels* and the role of term operations in term-labeled deduction. This leads to different possibilities for the the division of grammatical labor. And while it is possible to construct term-labeled categorial type systems of the kind defined below which are equivalent to standard categorial systems, it is of linguistic interest to note that the system which exhibits freedom of quantifier scope is simpler than many of the more restrictive systems.

9. Types and Terms

We wish to label *syntactic types* with pairs of *terms* representing *interpretive* and *phonological* (here: orthographical) properties. The interpretive terms will be drawn from the extensional fragment of Montague's *IL*, augmented by pairing and projection operations and additional constants as needed. We call the phonological terms ϕ-terms and the interpretive terms *IL*-terms. Like the *IL*-terms, the ϕ-terms will be terms of a system of typed λ-terms. Labeling of syntactic types with terms is regulated by compatibility of the syntactic type to be labeled and the types of the labeling terms, as described below.

Syntactic types. Let At be a set of atomic types (including s, np, n). The set Σ of syntactic types is the smallest set containing At and closed under the binary operations $\otimes$ (product) and $\rightarrow$ (residual). We drop parentheses of nested residual types according to the convention of right-associativity, so that $a \rightarrow b \rightarrow c$ represents $(a \rightarrow (b \rightarrow c))$.

ϕ-types. There is a single atomic ϕ-type: the type s (for 'string'). If A and B are ϕ-types, so are $A \rightarrow B$ and $\langle A, B\rangle$.

ϕ-terms. Let V be a set of string atoms. For each ϕ-type α, we adjoin denumerably many variable terms $x_1^\alpha, \ldots$. (We typically use x, y, z as variables of type s and P, Q, R as variables of type $s \rightarrow s$, suppressing subscripts and superscripts.) The point of the definition below is to characterize a set of linear higher-order terms over a simple algebraic structure, where *linear* means that any variable occurs in a term at most once and vacuous abstraction is forbidden. Accordingly, the set of ϕ-terms is the least set satisfying the conditions below, which also define for each ϕ-term t, the multiset $FA(t)$ of *free atoms* of t and the multiset $FV(t)$ of *free variables* of t, which will be useful later:

1. Each element v of V is a ϕ-term of type s; $FA(v) = v$, $FV(v) = \emptyset$;
2. Each variable x^α is a ϕ-term of type α; $FA(x) = x = FV(x)$;
3. If A and B are two ϕ-terms of type s with $FV(A) \sqcap FV(B) =$

$\emptyset$, then A · B is a ϕ-term of type s; $FA(A \cdot B) = FA(A) \sqcup FA(B)$, $FV(A \cdot B) = FV(A) \sqcup FV(B)$;

4. For all ϕ-terms A, B, C of type s, $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ whenever either side (hence, both sides) is defined;
5. There is a designated ϕ-term constant 1 of type s, with $1 \cdot A = A = A \cdot 1$ for all ϕ-terms A of type s; $FA(1) = FV(1) = \emptyset$;
6. If A is a ϕ-term of type $\alpha \rightarrow \beta$ and B is a ϕ-term of type α, and $FV(A) \sqcap FV(B) = \emptyset$, then (AB) is a ϕ-term of type β; $FA((AB)) = FA(A) \sqcup FA(B)$; $FV((AB)) = FV(A) \sqcup FV(B)$;
7. If x is a variable ϕ-term of type α and A is a ϕ-term of type β with $x \in FV(A)$, then $(\lambda x.A)$ is a ϕ-term of type $\alpha \rightarrow \beta$; $FA(\lambda x.A) = FA(A) \backslash \{x\}$, $FV(\lambda x.A) = FV(A) \backslash \{x\}$;
8. If A and B are two ϕ-terms of types α and β, respectively, with $FV(A) \sqcap FV(B) = \emptyset$, then $\langle A, B\rangle$ is a ϕ-term of type $\langle \alpha, \beta\rangle$; $FA(\langle A, B\rangle) = FA(A) \sqcup FA(B)$; $FV(\langle A, B\rangle) = FV(A) \sqcup FV(B)$;
9. If $A = \langle B, C\rangle$ is a ϕ-term of type $\langle \alpha, \beta\rangle$, then $\pi_1 A = B$ and $\pi_2 A = C$ are ϕ-terms of types α and β, respectively; $FA(\pi_1 A) = FA(B)$, $FA(\pi_2 A) = FA(C)$, $FV(\pi_1 A) = FV(B)$, $FV(\pi_2 A) = FV(C)$.

Remarks. The set of ϕ-terms just defined contains the free monoid $\langle V^*, \cdot, 1\rangle$ generated by V. Because the underlying algebra of ϕ-terms is a **mon**oid, we call the resulting system **mon:LP**. On the other hand, if V is empty, the above definitions extend the usual definition of a term algebra [21] to a certain set of higher-order terms – namely, those terms which contain at most one occurrence of any free variable and which disallow vacuous abstraction. Thus, if a term of the form $\lambda x.A$ is a ϕ-term, then A contains exactly one occurrence of the variable x.

For all the terms employed, substitution and β-conversion are defined in the standard way [19]. We write $\alpha \rhd \alpha'$ when α' is the result of removing one or more β-redexes from α by β-conversion, or by replacing an occurrence in α of the form $\pi_1\langle u, v\rangle$ (respectively, $\pi_2\langle u, v\rangle$) by u (respectively, v).

Labeled types. A labeled type is an element of *Typ* associated with a ϕ-term and an *IL*-term. This association is regulated by type-compatibility functions τ_ϕ: $Typ \rightarrow \phi$-types and τ_{IL}: $Typ \rightarrow IL$-types. For each atom $A \in Typ$, we have $\tau_\phi(A) = s$; and we assume that $\tau_{IL}(A)$ is defined. In particular, we shall assume that τ_{IL} makes the assignments:

$$s \mapsto t,$$
$$np \mapsto e,$$
$$n \mapsto (e \rightarrow t).$$

These assumptions are obviously too extensional, but the simplifications involved do not affect the properties of interest here. Inductively, we have the clauses:

$$\begin{aligned}
\tau_\phi(A \to B) &= \tau_\phi(A) \to \tau_\phi(B) \\
\tau_{IL}(A \to B) &= \tau_{IL}(A) \to \tau_{IL}(B) \\
\tau_\phi(A \cdot B) &= \langle \tau_\phi(A), \tau_\phi(B) \rangle \\
\tau_{IL}(A \cdot B) &= \langle \tau_{IL}(A), \tau_{IL}(B) \rangle
\end{aligned}$$

We display labeled types, in the notation of Pereira [40], in the form $t\colon T \rightsquigarrow t'$, with $T \in Typ$, t a ϕ-term of type $\tau_\phi(T)$ and t' an IL-term of type $\tau_{IL}(T)$.

9.1. *Labeled Deduction*

We give below a sequent presentation of a system of labeled deduction for the labeled types just defined. A sequent $\Gamma \vdash t\colon A \rightsquigarrow t'$ is a pair $\langle \Gamma, t\colon A \rightsquigarrow t' \rangle$, whose antecedent Γ is a nonempty multiset of labeled types with no free variables in common, and whose succedent $t\colon A \rightsquigarrow t'$ is a single labeled type whose terms are in λ-normal form – that is, they contain no β-redexes and no terms of the form $\pi_1\langle t, t' \rangle$ or $\pi_2\langle t, t' \rangle$. Identifying individual labeled types with the corrseponding singleton multiset, we display antecedent multisets as lists of multisets. For a labeled type $\mathrm{t}\colon A \rightsquigarrow t$, we write $FA(A)$ for $FA(\mathrm{t}) \sqcup FA(t)$; similarly for $FV(A)$. If $\Gamma = \mathrm{t}_1\colon A_1 \rightsquigarrow t_1, \ldots, \mathrm{t}_k\colon A_k \rightsquigarrow t_k$ is a sequent antecedent, then $FA(\Gamma) = FA(A_1) \sqcup \ldots \sqcup FA(A_k)$; $FV(\Gamma)$ is defined similarly. For a sequent of the form $\Gamma \vdash \mathrm{t}\colon A \rightsquigarrow t$, we indicate that $FA(\Gamma) = X$ by writing $\Gamma \vdash_X \mathrm{t}\colon A \rightsquigarrow t$. To denote the multiset union of two *disjoint multisets* X and Y, we write $X \dot{\sqcup} Y$.

$X \sqcup Y$. In the presentation below, postulates appear on the left and accompanying term conditions appear on the right.

identity axiom: $\mathrm{t}\colon A \rightsquigarrow t' \vdash \mathrm{u}\colon A \rightsquigarrow u'$ $\qquad A \in At, \mathrm{t} \rhd \mathbf{u}, t' \rhd t'$

$L\to$ $$\frac{\Gamma \vdash_X \mathrm{u}\colon A \rightsquigarrow u' \quad (\mathrm{tu})\colon B \rightsquigarrow (t'u'), \Delta \vdash_{Y \dot{\sqcup} FA(A)} \mathrm{v}\colon C \rightsquigarrow v'}{\Gamma, \mathrm{t}\colon A \to B \rightsquigarrow t', \Delta \vdash_{X \dot{\sqcup} Y} \mathrm{v}\colon C \rightsquigarrow v'}$$

$R\to$ $$\frac{\Gamma, \mathrm{x}\colon A \rightsquigarrow x' \vdash_{X \dot{\sqcup} \mathrm{x} \dot{\sqcup} x'} \mathrm{t}\colon B \rightsquigarrow t'}{\Gamma \vdash_X \lambda \mathrm{x}.\, \mathrm{t}\colon A \to B \rightsquigarrow \lambda x'.t'}$$

$L\otimes$ $$\frac{\pi_1(\mathrm{t})\colon A \rightsquigarrow \pi_1(t'), \pi_2(\mathrm{t})\colon B \rightsquigarrow \pi_2(t'), \Gamma \vdash \mathrm{v}\colon C \rightsquigarrow v'}{\mathrm{t}\colon A \otimes B \rightsquigarrow t', \Gamma \vdash \mathbf{v}\colon C \rightsquigarrow v'}$$

$R\otimes$ $$\frac{\Gamma \vdash_X \mathrm{u}\colon A \rightsquigarrow u' \quad \Delta \vdash_Y \mathrm{v}\colon B \rightsquigarrow v'}{\Gamma, \Delta \vdash_{X \dot{\sqcup} Y} \langle \mathrm{u}, \mathrm{v} \rangle\colon A \otimes B \rightsquigarrow \langle u', v' \rangle}$$

cut:
$$\frac{\Gamma \vdash \mathrm{u}: A \rightsquigarrow u' \quad \mathrm{v}: A \rightsquigarrow v', \Delta \vdash \mathrm{t}: C \rightsquigarrow t'}{\Gamma, \Delta \vdash \mathrm{t}: C \rightsquigarrow t'} \qquad \mathrm{v} \triangleright \mathrm{u}, v' \triangleright v'$$

Remark 1: normalization. We have required that the terms of succedent types be in λ-normal form. For typed λ-terms, such normal forms always exist and are unique. The requirement that the terms of the antecedent type in axiom instances convert by β-reduction to the corresponding succedent terms arises in connection with the rule $L \rightarrow$, as may be observed in the proof below:

$$\frac{\mathrm{j}: np \rightsquigarrow j \vdash \mathrm{j}: np \rightsquigarrow j \quad (\lambda\, \mathrm{u.\, u\, walks})\mathrm{j}: s \rightsquigarrow (\lambda u.walk'(u))j \vdash \mathrm{j\ walks}: s \rightsquigarrow walk'(j)}{\mathrm{j}: np \rightsquigarrow j, \lambda\, \mathrm{u.\, u\ walks}: np \rightarrow s \rightsquigarrow \lambda u.walk'(u) \vdash \mathrm{j\ walks}: s \rightsquigarrow walk'(j)}$$

In general, we must allow the φ-term of the residual type in the conclusion of the rule to be an abstraction, since abstractions play an essential role in introducing the directionality of combination of phonological structure. But then we need a step of normalization to convert redexes to λ-normal form. In the formulation above, this step is enforced at the level of axiom-leaves. Alternatively, normalization could be enforced in the premises of the rules $L \rightarrow$ and $L\otimes$.[7]

Normalization also plays a role in the term conditions on Cut.

Remark 2: parameters. In the proof step which introduces the principal type- and term-constructors in succedents of the form $\lambda\mathrm{x.t}: A \rightarrow B \rightsquigarrow \lambda x'.t'$, it is required that the term variables x and x' not occur in the antecedent. This is the analogue in the present context of the eigenvariable condition on the rule $R\forall$, in the following sense: if the variables x and x' are of types α and α', respectively, then the intuitive interpretation of the φ- and *IL*-terms λ x.t and $\lambda x'.t'$ is as functions defined for all entities of types α and α'. Thus, we countenance the deduction of a sequent of the form

$$\Gamma \vdash \lambda x.\phi: A \rightarrow B \rightsquigarrow \lambda x'.\phi'$$

from a premise sequent of the form

[7] In an earlier version of this paper, I attempted to compile normalization into the statement of the $L \rightarrow$ rule, by stating it along the following lines, in a way connected to the practice in Prolog programming called 'pseudoabstraction' by Pereira [39, 40]:

$$\frac{\Gamma \vdash \mathrm{u}: A \rightsquigarrow u \quad [\mathrm{u/x}]\mathrm{t}: B \rightsquigarrow [u/x]t', \Delta \vdash \mathrm{v}: C \rightsquigarrow v'}{\lambda\mathrm{x.t}: A \rightarrow B \rightsquigarrow \lambda x.t', \Gamma, \Delta \vdash \mathrm{v}: C \rightsquigarrow v'}$$

As Pereira stresses, and as a referee for this paper pointed out, this is in general not adequate, as may be seen when the the terms u or u in the above formulation stand themselves for higher order terms: in this case, substitution eliminates a β-redex at the top level, but can lead to new β-redexes internal to $[u/x]t$, as for example in $[\lambda P.P(a)/\mathcal{Q}](\mathcal{Q}(\lambda y.y))$.

$$\Gamma, x\colon A \rightsquigarrow x' \vdash \phi\colon B \rightsquigarrow \phi'$$

because the term variables represent arbitrary objects of the appropriate type. We call x and x' *parameters* or *parametric terms*: they are always single variables – that is, terms with no internal structure – and each one occurs only once among the variables of any sequent antecedent. (For discussion, see [24] and [40].)

Remark 3: free atoms. The following theorem is the analog here of a result first observed by van Benthem [5].

DEFINITION. A sequent $\Gamma \vdash \mathsf{t}\colon A \rightsquigarrow t$ for which $FA(\Gamma) = FA(A)$ is said to be *term-linear*.

THEOREM. *If a sequent* Σ *is valid in* **mon:LP**, *then* Σ *is term-linear.*

Proof. Induction. If $\mathsf{t} \rhd \mathsf{u}$, then $FA(\mathsf{t}) = FA(\mathsf{u})$. So the claim holds of axiom instances. Checking the inference rules is straightforward: the term conditions on inference rules are specifically designed so that this theorem will hold. □

9.1.1. *Cut Elimination*

Gentzen's Cut elimination proof can be adapted to show that any theorem of the above sequent system can be proved without using the Cut inference rule. The proof can be carried out in a way that closely follows Lambek's adaptation of Gentzen's idea to the associative syntactic calculus **L**. The only difference is that we need to ensure that the term conditions are respected. We omit the proofs here.

9.1.2. *Decidability*

In the calculus **LP** which forms the basis of the type system for the term-labeled types used here, the proof of Cut Elimination opens the way to a straightforward demonstration that the calculus is *decidable*: there is an effective method for determining whether a sequent is deducible from the postulates of **LP** or not. Does the proof carry over directly to the system of labeled deduction considered here? There is one sticking point: the label of the left-hand premise of the rule $L\to$ is not determined by the sub-terms of its conclusion. Nevertheless, decidability follows from the proof of a slightly stronger result.

THEOREM. *Given an antecedent multiset* Γ *of term-labeled types and a* *succedent type A, it is possible to determine the set of pairs* (Term, *Term*) *for which the sequent* $\Gamma \vdash \mathsf{Term}: A \rightsquigarrow \mathit{Term}$ *is provable.*

Proof. The proof is by induction on sequent-degree. In the base case of the induction, we take the sequent-degree to be 0. A sequent of degree zero contains only atomic types and the only provable sequents of degree 0 constitute axiom instances of the form:

$$\mathsf{u}: A \rightsquigarrow u \vdash \mathsf{Term}: A \rightsquigarrow \mathit{Term}$$

Since it is decidable whether or not we have $\mathsf{u} \rhd \mathsf{Term}$ and $u \rhd \mathit{Term}$, it is decidable whether sequents of degree 0 are provable. In the inductive step, we assume that the Theorem holds for sequents of degree k and examine sequents of degree $k + 1$. In the absence of Cut, any provable sequent of degree $k + 1$ must be derived by an application of one of the inference rules $R \rightarrow$, $L \rightarrow$, $R \otimes$, $L \otimes$ in a way that introduces the principal type-constructor of an antecedent labeled type or the succedent labeled type. This can be done in only finitely many ways. For each possibility, the inductive hypothesis ensures that it is possible to determine both the provability and term-labeling of candidate premises; and when a candidate set of premises is provable, the conclusion is provable as well, with labeling determined by the labeling of the premises. □

9.2. *Grammatical Applications*

In grammatical applications, we are interested in proofs from lexical assumptions. The ϕ-terms of lexical assumptions have a special character: if functional lexical ϕ-terms are applied to arguments of the required type, they should in fact determine (by normalization) elements of the underlying algebra V^+. The following examples illustrate lexical and non-lexical ϕ-terms of various simple types, with the non-lexical terms prefixed by a †:

Type	Term
$s \rightarrow s$	$\lambda x.x \cdot c, \lambda x.c \cdot x, \lambda x.c \cdot x \cdot d$
	$\dagger \lambda x.x$
$(s \rightarrow s) \rightarrow s$	$\lambda P.P(c), \lambda P.c \cdot P(1), \lambda P.(P(1)) \cdot c$
	$\dagger \lambda P.P(1)$
$((s \rightarrow s) \rightarrow s) \rightarrow s$	$\lambda \mathcal{Q}.c \cdot \mathcal{Q}(\lambda x.x)$

As in these examples, we require that the ϕ-term associated with a lexical assumption must:

- contain a subterm drawn from V^+;

- contain no free variables;

We list below some lexical assumptions which satisfy these requirements:

λ x.λ y.(y · questioned · x: $np \rightarrow (np \rightarrow s) \rightsquigarrow$
$\lambda x \lambda y.question'(x)(y)$
smith: $np \rightsquigarrow s$
jones: $np \rightsquigarrow j$
λ x. λ Q. (Q (every x)): $n \rightarrow ((np \rightarrow s) \rightarrow s) \rightsquigarrow \lambda P \lambda Q \forall(P)(Q)$
λ x. λ Q. (Q (a x)): $n \rightarrow ((np \rightarrow s) \rightarrow n) \rightsquigarrow \lambda P \lambda Q \exists(P)(Q)$
dean: $n \rightsquigarrow dean'$
student: $n \rightarrow student'$

9.2.1. *Grammatical Composition in* **mon:LP**

Grammatical composition in **mon:LP** depends both on the underlying type system – identical with the system of **LP** – and on the properties of the labeling algebras and the method of linking between terms and types. With regard to the ϕ-terms, four properties stand out.

First, since the underlying phonological algebra is a free monoid, the algebraic product of any two distinct normal ϕ-terms of type s distinct from 1 is never idempotent and never commutative. That is, the following *inequalities* hold:

$$s_1 \cdot s_2 \neq s_2 \cdot s_1; \quad s \cdot s \neq s$$

Second, because a λ-operator need not bind only string-peripheral variables, phonological composition is not restricted to concatenation, but is generalized to substitution. This means that phonologically discontinuous constituents are possible, as illustrated by the perfectly well-formed labeled type λ x. take x to task: $np \rightarrow vp \rightsquigarrow \lambda x.(take(to\ task))(x)$.

Third, although the underlying algebraic structure on which ϕ-terms are constructed and the underlying algebraic structure on which IL-terms are constructed are distinct, the two systems of terms share common properties of pairing, projection, abstraction, and application. It is through this common system that terms are linked to types, in a way that systematically enforces corresponding links between subtypes and subterms.

Fourth, since phonological order is expressed in the structure of the ϕ-terms, there is no need to impose linear order on the structure of term-labeled types. It is this division of labor which makes it possible to assume that sequent antecedents are formed by multisets of labeled types, rather than sequences of labeled types.

To illustrate these properties, we present a proof below of the analog in **mon:LP** of the **LP**-valid Permutation rule $A \to B \to C \vdash B \to A \to C$, which illustrates this rule's independence of the details of basic word order. Let Term range over linear monoidal products of two string variables x and y and a string constant c. Thus, Term ranges over the string c · x · y and its permutations. Think of c as representing the phonological structure of a transitive verb and any value of Term as representing one possible convention governing the phonological properties of the mode of combination of the transitive verb c and its *np* arguments x and y. Consider now the following proof, where np_{subj} stands for the type of a subject *np*, and np_{obj} stand for the type of an object *np*. We give the proof schematically, with each type labeled by a single schematic term (appearing directly below it), which exhibits all the relevant information governing the relation between types and labels. Thus, the schematic term labeling a type does not present the internal structure of either ϕ-term label or the *IL*-term label, but only the structure of application, abstraction, pairing, and projection which connects deduction in the type system with the corresponding term operations. Thus internal structure of the ϕ-term is consistent with any of the basic words orders sov, svo, ovs, osv, vso, vos.

$$\dfrac{\dfrac{\dfrac{\underset{\mathrm{x}}{np_{\text{subj}}} \vdash \underset{\mathrm{x}}{np_{\text{subj}}} \qquad \dfrac{\underset{y}{np_{\text{obj}}} \vdash \underset{y}{np_{\text{obj}}} \qquad \underset{((\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{T})\mathrm{x})\mathrm{y}}{s} \vdash \underset{\mathrm{T}}{s}}{\underset{(\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{T})\mathrm{x}}{np_{\text{obj}} \to s},\ \underset{y}{np_{\text{obj}}} \vdash \underset{\mathrm{T}}{s}}}{\underset{\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{T}}{np_{\text{subj}} \to np_{\text{obj}} \to s},\ \underset{\mathrm{y}}{np_{\text{obj}}},\ \underset{\mathrm{x}}{np_{\text{subj}}} \vdash \underset{\mathrm{T}}{s}}}{\underset{\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{T}}{np_{\text{subj}} \to np_{\text{obj}} \to s},\ \underset{\mathrm{y}}{np_{\text{obj}}} \vdash \underset{\lambda \mathrm{x}.\mathrm{T}}{np_{\text{subj}} \to s}}}{\underset{\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{T}}{np_{\text{subj}} \to np_{\text{obj}} \to s} \vdash \underset{\lambda \mathrm{y}.\lambda \mathrm{x}.\mathrm{T}}{np_{\text{obj}} \to np_{\text{subj}} \to s}}$$

9.2.2. *Applications to Quantification*

Quantifiers in **mon:LP** can be associated with a single type, as illustrated below,

$$\lambda \mathrm{P}.\mathrm{P}(\text{every doctor}): (np \to s) \to s \rightsquigarrow \lambda P.\forall(doctor')(P)$$
$$\lambda \mathrm{P}.\mathrm{P}(\text{a problem})(np \to s) \to s \rightsquigarrow \lambda P.\exists(problem')(P)$$

Because of the proof-induced polymorphism of **mon:LP**, a labeled type with this structure can combine not only with a 1-place predicate of type $np \to s$, but with any k-place predicate whose k arguments include an argument of type np. One way to see why this is possible is to recall that **mon:LP** allows both Permutation (so the k-place predicate may be assigned a type of the form $\ldots \to np \to s$, with the np argument of interest represented as the last argument that the predicate combines with) and Division (so the quantifier type $(np \to s) \to s$ is shiftable to type $(\ldots np \to s) \to (\cdots \to s)$).

If a k-place predicate ($k \geqslant 2$) takes two or more np-arguments, then it can combine with two or more quantifiers. Fixing the argument position associated with each quantifier, there are as many orders of combination as there are ways of assigning a linear order to the set of quantifiers. Different orders of combination give rise intrinsically to different scopes. As a result, the system contains no special rule for introducing quantifier scope ambiguities: rather, such ambiguities arise in **mon:LP** as a form of *proof indeterminacy* – the inability of the proof system to fix a particular order of combination.

To illustrate these ideas concretely, we examine the set of IL-terms assignable to the succedent term-labeled type every detective caught some thief: $s \rightsquigarrow Term$ on the basis of the lexical assumptions below:

$$\begin{array}{l}
\lambda \mathrm{P.P(every\ detective)}: (np \to s) \to s \rightsquigarrow \lambda P.\forall(detective')(P) \\
\lambda \mathrm{P.P(some\ thief)}: (np \to s) \to s \rightsquigarrow \lambda P.\exists(thief')(P) \\
\lambda \mathrm{x}.\ \lambda \mathrm{y.(y\ caught\ x)}: n \to np \to s \rightsquigarrow \lambda x.\lambda y.caught'(x)(y)
\end{array}$$

In **mon:LP**, there are two essentially-different cut-free proofs of this kind. One of them is equivalent (in the sense of having the same axiom-leaf bindings) to the schematic proof below:

$$\begin{array}{c}
\underset{\mathrm{r}}{np} \vdash \underset{\mathrm{r}}{np} \qquad \underset{\mathrm{(TV(P))(r)}}{s} \vdash \underset{\mathrm{E}}{s} \\
\hline
\underset{\mathrm{p}}{np} \vdash \underset{\mathrm{p}}{np} \quad \underset{\mathrm{TV(p)}}{np \to s}, \underset{\mathrm{r}}{np} \vdash \underset{\mathrm{E}}{s} \\
\hline
\underset{\mathrm{TV}}{np \to np \to s}, \underset{\mathrm{r}}{np}, \underset{\mathrm{p}}{np} \vdash \underset{\mathrm{E}}{s} \\
\hline
\underset{\mathrm{TV}}{np \to np \to s}, \underset{\mathrm{r}}{np} \vdash \underset{\lambda \mathrm{p.E}}{np \to s} \qquad \underset{\mathrm{B(\lambda p.E)}}{s} \vdash \underset{\mathrm{D}}{s} \\
\hline
\end{array}$$

$$\cfrac{\cfrac{\underset{\mathrm{TV}}{np\to np\to s},\ \underset{\mathrm{B}}{(np\to s)\to s},\ \underset{\mathrm{r}}{np} \vdash \underset{\mathrm{D}}{s}}{\underset{\mathrm{TV}}{np\to np\to s},\ \underset{\mathrm{B}}{(np\to s)\to s} \vdash \underset{\lambda\,\mathrm{r.D}}{np\to s} \qquad \underset{\mathrm{A}(\lambda\,\mathrm{r.D})}{s} \vdash \underset{\mathrm{C}}{s}}}{\underset{\mathrm{A}}{(np\to s)\to s},\ \underset{\mathrm{TV}}{np\to np\to s},\ \underset{\mathrm{B}}{(np\to s)\to s} \vdash \underset{\mathrm{C}}{s}}$$

The schematic terms in this proof have the values displayed below:

A := λ P. P(every detective), $\lambda P.\forall(\mathit{detective}')(P)$
TV:= λ x.λ y.y caught x, $\lambda x.\lambda y.\mathit{caught}'(x)(y)$
B := λ P. P(some thief), $\lambda P.\exists(\mathit{thief}')(P)$
C := every detective caught some thief,
$\forall(\mathit{detective}')\ (\lambda r.\exists(\mathit{thief}')(\lambda p.\mathit{caught}'(p)(r)))$
D := r caught some thief, $\exists(\mathit{thief}')(\lambda p.\mathit{caught}'(p)(r))$
E := r caught p, $\mathit{caught}'(p)(r)$

The reader may check that all the required normalizations of terms, such as that represented schematically as A(λ r. D) ⊳ C, in fact hold.

There is a second proof whose endsequent differs from that of the proof above only in the *IL*-term assigned to the succedent type. We present the last two lines of a schematic proof below.

$$\cfrac{\underset{\mathrm{A}}{(np\to s)\to s},\ \underset{\mathrm{TV}}{np\to np\to s} \vdash \underset{\lambda \mathrm{u.G}}{np\to s} \qquad \underset{\mathrm{B}(\lambda\mathrm{u.G})}{s} \vdash \underset{\mathrm{F}}{s}}{\underset{\mathrm{A}}{(np\to s)\to s},\ \underset{\mathrm{TV}}{np\to np\to s},\ \underset{\mathrm{B}}{(np\to s)\to s} \vdash \underset{\mathrm{F}}{s}}$$

The schematic terms A, TV, and B have the same values here as in the previous proof; the newly introduced schematic terms F and G have values:

F := every detective caught some thief,
$\exists(\mathit{thief}')\ (\lambda u.\forall(\mathit{detective}')\ (\lambda v.\mathit{caught}'(u)(v)))$
G := every detective caught u,
$\forall(\mathit{detective}')\ (\lambda v.\mathit{caught}'(u)(v))$

The two proofs above illustrate precisely the two *IL*-terms that can be paired with the ϕ-term every detective caught some thief relative to the assumed lexical assumptions. In particular, in contrast with **LP**, the succedent types below cannot be proved from these lexical assumptions:

every doctor caught some thief: $s \rightsquigarrow \forall(\mathit{detective'})$
$(\lambda p.\exists(\mathit{thief'})(\lambda r.\mathit{caught'}(p)(r))$
every doctor caught some thief: $s \rightsquigarrow \exists(\mathit{thief'})$
$(\lambda v.\forall(\mathit{detective'})(\lambda u.\mathit{caught'}(u)(v))$

And the reason for this is a simple one: there is an implicit regulation of variables between ϕ-terms and IL-terms.

9.3. *Explicit Cross-Dimensional Control*

In the presentation above of **mon:LP**, there is a correspondence across dimensions that is governed by the design properties of the type language. For example, we can represent any type in these systems by a schema of the form (for $k \geqslant 0$):

$$\lambda \mathsf{x}_k \ldots \lambda \mathsf{x}_1.\mathsf{T}: \tau_k \rightarrow \cdots \tau_1 \rightarrow \tau_0 \rightsquigarrow \lambda x_k \ldots \lambda x_1 T$$

In this schema, the type τ_i and the variables x_i, and x_i $(1 \leqslant i \leqslant k)$ are implicitly linked, and in proofs this triple corresponds to one of the atomic labeled types of an axiom leaf. Note that there is no intrinsic property of type or label which connects the corresponding elements of different dimensions: rather, in the schema above, it is the common sequential ordering of types, on the one hand, and the interpretive and phonological abstraction operators, on the other. In general, it is possible to permute the elements of these sequences as long as the correspondence across them is maintained. This suggests that the sequences are merely a particular way of keeping track of the correspondence. In fact, the common sequencing of the elements represented in the schema above merely means that these elements are indexed by a single linearly-ordered set. Although the assumption that this set is linearly-ordered has some advantages – in particular, the advantage that the correspondence can be expressed implicitly by the design features of the type language –, it has some disadvantages as well. To gain some insight into these properties, we shall describe two families of systems in which the correspondence of elements across dimensions is expressed explicitly.

9.3.1. *Decomposition of Signed Types to Signed Atomic Types*

A fundamental idea in the sequent systems is that axioms take the form $A \vdash A$, with the turnstile flanked by two occurrences of a single type. Axioms thus encode a kind of matching of occurrences of types, a matching of an antecedent occurrence and a succedent occurrence. It is useful

[43, 46] to indicate this matching in another way by assigning a *polarity* to each type. If we assign a succedent type A the polarity 0 and an antecedent type A the polarity 1, then we can replace the identity sequent $A \vdash A$ with the *multiset* of *signed types* $[\langle A, 1\rangle, \langle A, 0\rangle]$. If we inspect the standard rules for products and residuals, we might hope that signed products and residuals obey the following rules:

$$\begin{aligned}
&[\Gamma, \langle A \rightarrow B, 0\rangle] \Rightarrow [\Gamma, \langle A, 1\rangle, \langle B, 0\rangle]\\
&[\Gamma, \langle A \rightarrow B, 1\rangle] \Rightarrow [\Gamma, \langle A, 0\rangle, \langle B, 1\rangle]\\
&[\Gamma, \langle A \otimes B, 0\rangle] \Rightarrow [\Gamma, \langle A, 0\rangle, \langle B, 0\rangle]\\
&[\Gamma, \langle A \otimes B, 1\rangle] \Rightarrow [\Gamma, \langle A, 1\rangle, \langle B, 1\rangle]
\end{aligned}$$

If we decompose complex types in the conclusion of the inference rules for products and residuals according to these rules, we observe that the polarity of a type is a global proof invariant. This is shown for the rules below, where sequent antecedents are multisets of types and sequent succedents constitute a single type.

	Inference figure	Polar types	Decomposed polar types
$R\rightarrow$	$\dfrac{\Theta, A \vdash B}{\Theta \vdash A \rightarrow B}$	$\dfrac{[\Theta', \langle A, 1\rangle, \langle B, 0\rangle]}{[\Theta', \langle A \rightarrow B, 0\rangle]}$	$\dfrac{[\Theta', \langle A, 1\rangle, \langle B, 0\rangle]}{[\Theta', \langle A, 1\rangle, \langle B, 0\rangle}$
$L\rightarrow$	$\dfrac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \rightarrow B \vdash C}$	$\dfrac{[\Gamma', \langle A, 0\rangle] \quad [\Delta', \langle B, 1\rangle, \langle C, 0\rangle]}{[\Gamma', \langle A \rightarrow B, 1\rangle, \Delta', \langle C, 0\rangle]}$	$\dfrac{[\Gamma', \langle A, 0\rangle] \quad [\Delta', \langle B, 1\rangle, \langle C, 0\rangle]}{[\Gamma', \langle A, 0\rangle, \langle B, 1\rangle, \Delta', \langle C, 0\rangle]}$
$L\otimes$	$\dfrac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$	$\dfrac{[\Gamma', \langle A, 1\rangle, \langle B, 1\rangle, \langle C, 0\rangle]}{[\Gamma', \langle A \otimes B, 1\rangle, \langle C, 0\rangle]}$	$\dfrac{[\Gamma', \langle A, 1\rangle, \langle B, 1\rangle, \langle C, 0\rangle]}{[\Gamma', \langle A, 1\rangle, \langle B, 1\rangle, \langle C, 0\rangle]}$
$R\otimes$	$\dfrac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$	$\dfrac{[\Gamma', \langle A, 0\rangle] \quad [\Delta, \langle B, 0\rangle]}{[\Gamma', \Delta'(A \otimes B, 0\rangle]}$	$\dfrac{[\Gamma', \langle A, 0\rangle] \quad [\Delta, \langle B, 0\rangle]}{[\Gamma', \Delta'\langle A, 0\rangle, \langle B, 0\rangle]}$

On this basis, then, we may translate any sequent $A_1, \ldots, A_k \vdash B$ into a multiset of polar types $[\langle, A, 1\rangle, \ldots, \langle A_k, 1\rangle, \langle B, 0\rangle]$, and then by performing the unfolding transformations on any complex types, convert this multiset of polar types into a multiset of *atomic*, *polar* types. If the original sequent is valid, the axiom leaves of the proof tree correspond precisely to pairs of atomic types of opposite polarity. This is a proof invariant similar to van Benthem's notion x-count [5, 6] or Roorda's notion *balance* [42].

Although such a pairing exists for every valid sequent, such pairings can exist as well for sequents which are invalid in all of the systems discussed here, such as the Lowering sequent (see §6) $(B \rightarrow A) \rightarrow A \vdash B$, whose atomic polar normal form is $\{\langle B, 1\rangle, \langle A, 0\rangle, \langle A, 1\rangle, \langle B, 0\rangle\}$. In converting a sequent to its atomic polar normal form, a critical property of proof structures is lost: namely, information concerning the scope of each type-constructor. The two-premise rules $L \rightarrow$ and $R \otimes$ not only introduce a new occurrence of $\rightarrow$ or $\otimes$, they also merge the endsequents of their two premises into a single sequent. In trying to construct a proof bottom-up from such an endsequent, one must take apart the antecedent occurrence of $\rightarrow$ or the succedent occurrence of $\otimes$ and *in addition*, decide how to split the sequent context of the complex type into two independent contexts for the two components of the complex type. It is this aspect of proof structure which is ignored in the transition from a sequent to its atomic polar normal form. One way of building in information about the scope of a connective is to refer to the corresponding properties of the associated λ-term. Roorda [41, 42] uses this method to extend to **L** and **LP** the proof-nets – *les substantifiques moelles* – of Girard [14, 15]. Moortgat [27, 28] uses string-term conditions for the same purposes. Here we discuss the extension of these methods involving term-relations to **mon:LP**.

If we take an atomic labeled type in **mon:LP**, such as jones: $np \rightsquigarrow j$, we can assign it an atomic polar normal form by simply adding a polarity, 0 or 1. To extend the assignment to complex types, it is necessary to add conditions on the terms involved. In certain cases, it is necessary to introduce fresh variables or parameters, adding appropriate information concerning their interpretation. We write $x \sqsubset \text{Term}$ to mean that $x \in FA(Term)$.

Complex polar type	Decomposed polar form
Term: $A \otimes B^0 \rightsquigarrow Term$	[Term $\mapsto \langle \pi_1$ Term, π_2Term,$\rangle$ $Term \mapsto \langle \pi_1 Term,\ \pi_2 Term,\rangle$, π_1Term: $A^0 \rightsquigarrow \pi_1 Term$, π_2Term: $B^0 \rightsquigarrow \pi_2 Term$]
Term: $\mathrm{A} \otimes B^1 \rightsquigarrow Term$	[Term $\mapsto \langle \pi_1$Term, π_2Term,$\rangle$ $Term \mapsto \langle \pi_1 Term,\ \pi_2 Term,\rangle$, π_1Term: $A^1 \rightsquigarrow \pi_1 Term$, π_2Term: $B^1 \rightsquigarrow \pi_2 Term$]
Term: $A \rightarrow B^0 \rightsquigarrow Term$	[x, x fresh, Term = λx.Term′, x $\sqsubset$ Term′, $Term = \lambda x.Term', x \sqsubset Term'$, x: $A^1 \rightsquigarrow x$, Term: $B^0 \rightsquigarrow Term$]
Term: $A \rightarrow B^1 \rightsquigarrow Term$	[x, x fresh, Term = λx.Term′, x $\sqsubset$ Term′, $Term = \lambda x.Term', x \sqsubset Term'$, x: $A^1 \rightsquigarrow x$, Term′: $B^0 \rightsquigarrow Term'$]

Iteration leads to atomic polar normal forms, in which every type-forming operator is replaced by its polar decomposition.

Applying these methods to some of the lexical assumptions introduced earlier, we have:

λx. λy. y caught x: $np \to np \to s^1 \leadsto \lambda x.\lambda y.caught'(x)(y)$
  $\Rightarrow$ [x: $np^0 \leadsto x$, y: $np^0 \leadsto y$, y caught x: $s^1 \leadsto caught'(x)(y)$]
λP. P[every student): $(np \to s) \to s^1 \leadsto \lambda P.\forall(student')(P)$
  $\Rightarrow$ [P: $np \to s^0 \leadsto P$, P(every student): $s^1 \leadsto \forall(student')(P)$]
  $\Rightarrow$ [x, Q, x, Q fresh, P = λx.Q, x $\sqsubset$ Q, $P = \lambda x.Q$, $x \sqsubset Q$,
    x: $np^1 \leadsto x$, Q: $s^0 \leadsto Q$, P(every student): $s^1 \leadsto \forall(student')(P)$]
λP. P(some thief)) $(np \to s) \to s^1 \leadsto \lambda P.\exists(thief')(P)$
  $\Rightarrow$ [z, N, z, N fresh, M = λz.N, z $\sqsubset$ N, $M = \lambda z.N.$, z $\sqsubset N$,
    z: $np^1 \leadsto z$, N: $s^0 \leadsto N$, M(some thief): $s^1 \leadsto \exists(thief')(M)$]

Consider now the following examples of the decomposition of sequents:

Example 1.

sequent:α, β, $\gamma \vdash$ jay chased kay: $s \leadsto chased'(k)(j)$
  a = jay: $np \leadsto j$
  β = λx. λy. y chased x: $np \to np \to s \leadsto$
    $\lambda x.\lambda y.chased'(x)(y)$
  γ = kay: $np \leadsto k$

atomic polar decomposition

[jay: $np^1 \leadsto j$,
  x: $np^0 \leadsto x$,
  y: $np^0 \leadsto y$,
  y chased x: $s^1 \leadsto chased'(x)(y)$
  kay: $np^1 \leadsto k$
  jay chased kay: $s^0 \leadsto chased'(k)(j)$]

θ:
  x $\mapsto$ kay, $x \mapsto k$,
  y $\mapsto$ jay,y $\mapsto j$

μ:
  jay: $np^1 \leadsto j \mapsto$ y: $np^0 \leadsto y$
  kay: $np^1 \leadsto k \mapsto$ x: $np^0 \leadsto x$
  y chased x: $s^1 \leadsto chased'(x)(y) \mapsto$ jay chased kay:
    $s^0 \leadsto chased'(k)(j)$

In this example, θ is a function which anchors term-variables to terms; μ is function from labeled polar types with polarity 1 to labeled polar types with polarity 0 which satisfies the requirements that if μ: t: $T^1 \leadsto t \mapsto$ u: $U^0 \leadsto u$, then $T = U$, θ(t) = θ(u), and $\theta(t) = \theta(u)$. These requirements are related to sequent-style proofs in the following way: atomic types of opposite polarity paired by μ correspond to axiom leaves of a sequent proof; the term conditions imposed on axiom bindings are specified by θ; the scope of each connective in the endsequent is expressed by the type- and term-conditions of its decomposition, and the inclusion relations on terms.

A simple example involving multiple quantification is exhibited below:

Example 2.

sequent: $\alpha, \beta, \gamma \vdash$ every detective caught some thief:
$s \rightsquigarrow \forall(detective')(\lambda w.\exists(thief')(\lambda z.caught'(z)(w)))$
$\alpha = \lambda$ P. P(every detective): $(np \rightarrow s) \rightarrow s \rightsquigarrow \lambda P.\forall(detective')(P)$
$\beta = \lambda$ x. λ y. y caught x: $np \rightarrow np \rightarrow s \rightsquigarrow \lambda x.\lambda y.caught'(x)(y)$
$\gamma = \lambda$ R. R(some thief); $(np \rightarrow s) \rightarrow s \rightsquigarrow \lambda R.\exists(thief')(R)$

atomic polar decomposition:

[P: $np \rightarrow s^0 \rightsquigarrow P \mapsto \lambda$ w. Q: $np \rightarrow s^0 \rightsquigarrow \lambda w.Q$, w $\sqsubset$ Q, $w \sqsubset Q$,
R: $np \rightarrow s^0 \rightsquigarrow R \mapsto \lambda$ z. N: $np \rightarrow s^0 \rightsquigarrow \lambda z.N$, z $\sqsubset$ N, $z \sqsubset N$,
w: $np^1 \rightsquigarrow w$,
Q: $s^0 \rightsquigarrow Q$,
(λ w.Q)(every detective): $s^1 \rightsquigarrow \forall(detective')(\lambda w.Q)$
x: $np^0 \rightsquigarrow x$,
y: $np^0 \rightsquigarrow y$,
y caught x: $s^1 \rightsquigarrow caught'(x)(y)$
z: $np^1 \rightsquigarrow z$,
N: $s^0 \rightsquigarrow N$,
(λ z.N)(some thief): $s^1 \rightsquigarrow \exists(thief')(\lambda z.N)$
every detective caught some thief: $s^0 \rightsquigarrow \forall(detective')$
$(\lambda w.\exists(thief')(\lambda z.caught'(z)(w)))$
]

For this set of atomic polar types, where the global interpretation is fixed, there is just one solution that satisfies the inclusions relating terms:

θ:
y $\mapsto$ w, $y \mapsto w$
x $\mapsto$ z, $x \mapsto z$
N $\mapsto$ w caught z, $N \mapsto caught'(z)(w)$,
Q $\mapsto$ w caught some thief, $Q \mapsto \exists(thief')(\lambda z.N)$

μ:

w: $np^1 \rightsquigarrow w \mapsto$ y: $np^0 \rightsquigarrow y$
z: $np^1 \rightsquigarrow z \mapsto$ x: $np^0 \rightsquigarrow x$
y caught x: $s^1 \rightsquigarrow \mathit{caught}'(x)(y) \mapsto$ N: $s^0 \rightsquigarrow N$
(λ z.N)(some thief): $s^1 \rightsquigarrow \exists(\mathit{thief}')(\lambda z.N) \mapsto$ Q: $s^0 \rightsquigarrow Q$
(λ w.Q)(every detective): $s^1 \rightsquigarrow \forall(\mathit{detective}')(\lambda w.Q) \mapsto$
every detective caught some thief: $s^0 \rightsquigarrow \forall(\mathit{detective}')$
$(\lambda w.\exists(\mathit{thief}')(\lambda z.\mathit{caught}'(z)(w)))$

If we change the scope of quantifiers in the interpretation of the endsequent, the atomic polar decomposition differs only in that respect as well, but we have a different, but still unique, solution set θ' and μ':

Example 3.

sequent: $\alpha, \beta, \gamma \vdash$ every detective caught some thief:
$s \rightsquigarrow \exists(\mathit{thief}')(\lambda z.\forall(\mathit{detective}')(\lambda w.\mathit{caught}'(z)(w)))$
$\alpha = \lambda$ P. P(every detective): $(np \rightarrow s) \rightarrow s \rightsquigarrow \lambda P.\forall(\mathit{detective}')(P)$
$\beta = \lambda$ x. λ y. y caught x: $np \rightarrow np \rightarrow s \rightsquigarrow \lambda x.\lambda y.\mathit{caught}'(x)(y)$
$\gamma = \lambda$ R. R(some thief: $(np \rightarrow s) \rightarrow s \rightsquigarrow \lambda R.\exists(\mathit{thief}')(R)$

atomic polar decomposition:

[P: $np \rightarrow s^0 \rightsquigarrow P \mapsto \lambda$ w. Q: $np \rightarrow s^0 \rightsquigarrow \lambda w.Q$,
w $\sqsubset$ Q, $w \sqsubset Q$,
R: $np \rightarrow s^0 \rightsquigarrow R \mapsto \lambda$ z. N: $np \rightarrow s^0 \rightsquigarrow \lambda z.N$,
z $\sqsubset$ N, $z \sqsubset N$,
w: $np^1 \rightsquigarrow w$,
Q: $s^0 \rightsquigarrow Q$,
(λ w.Q)(every detective): $s^1 \rightsquigarrow \forall(\mathit{detective}')(\lambda w.Q)$
x: $np^0 \rightsquigarrow x$,
y: $np^0 \rightsquigarrow y$,
y caught x: $s^1 \rightsquigarrow \mathit{caught}'(x)(y)$
z: $np^1 \rightsquigarrow z$,
N: $s^0 \rightsquigarrow N$,
(λ z.N)(some thief): $s^1 \rightsquigarrow \exists(\mathit{thief}')(\lambda z.N)$
every detective caught some thief:
$s^0 \rightsquigarrow \exists(\mathit{thief}')(\lambda z.\forall(\mathit{detective}')(\lambda w.\mathit{caught}'(z)(w)))$]

In this case, we have the solution:

θ:
y $\mapsto$ w, $y \mapsto w$
x $\mapsto$ z, $x \mapsto z$
Q $\mapsto$ w caught z, $Q \mapsto \mathit{caught}'(w)(z)$
N $\mapsto$ every detective caught z, $N \mapsto \forall(\mathit{detective}')(\lambda w.Q)$

μ:
w: $np^1 \rightsquigarrow w \mapsto$ y: $np^0 \rightsquigarrow y$
z: $np^1 \rightsquigarrow z \mapsto$ x: $np^0 \rightsquigarrow x$
y caught x: $s^1 \rightsquigarrow \mathit{caught}'(x)(y) \mapsto$ Q: $s^0 \rightsquigarrow Q$
(λ z.N)(some thief): $s^1 \rightsquigarrow \exists(\mathit{thief}')(\lambda z.N)$
$\mapsto$ every detective caught some thief:
$s^0 \rightsquigarrow \exists(\mathit{thief}')(\lambda z.\forall(\mathit{detective}')(\lambda w.\mathit{caught}'(z)(w)))$
(λ w.Q)(every detective): $s^1 \rightsquigarrow \forall(\mathit{detective}')(\lambda w.Q)$
$\mapsto$ N : s^0 : N

Call a multiset of atomic polar types (and an associated set of inclusions relating the terms of the those types) *paired and anchored* if there is a matching substitution θ of term variables associated with types of polarity 0 with ground terms or parameters which is consistent with term inclusion relations and β-conversion (and is extended to a map from terms to terms in the usual way), and a bijection μ from labeled types of polarity 1 to polarity 0 which satisfies the condition: if μ: Term: $A^1 \rightsquigarrow \mathit{Term}$ = Term$'$: $B^0 \rightsquigarrow \mathit{Term}'$, then $A = B$ and θ(Term) $\rhd$ θ(Term$'$) and $\theta(\mathit{Term}) \rhd \theta(\mathit{Term}')$.

THEOREM. *If a sequent* Σ *is provable in* **mon:LP**, *then the atomic polar decomposition of* Σ *is paired and anchored.*

Proof. If a sequent is provable, then the set of axiom-leaf bindings provide μ; if the decomposition procedure introduces a new term variables v at any point that is distinct from the associated term u of type corresponding to the type labeled by v, impose the requirement $\theta(v) = u$. Since introduced variables are always fresh, this can be done in a globally consistent way. This yields the required substitution θ. □

We know already that any valid sequent in **mon:LP** is term-linear. We can use term-linearity to attain a result that goes in the other direction.

THEOREM. *If the atomic polar decomposition of a term-linear sequent*

Σ is paired and anchored relative to θ and μ, then $\theta(\Sigma)$ is provable in **mon:LP**.

Proof. By induction on the degree of the sequent Σ. If the degree of Σ is 0, then every labeled type in the sequent is atomic. Moreover, there is only one succedent type t': $T' \rightsquigarrow t'$, by the definition of sequent, and there can be only a single labeled antecedent type, t: $T \rightsquigarrow t$, since μ is a bijection. Thus, we have $\mu(\mathsf{t}\colon T^1 \rightsquigarrow t) = \mathsf{t}'\colon T'^0 \rightsquigarrow t'$ with $T = T'$, $\theta(t) \rhd_\beta \theta(t')$. Therefore, $\theta(\Sigma)$ has the form $\theta(\mathsf{t})$: $T \rightsquigarrow \theta(t) \vdash \theta(\mathsf{t}')$: $T \rightsquigarrow \theta(t')$. Thus, $\theta(\Sigma)$ is an instance of the identity axiom.

Now suppose that the theorem holds for sequents of degree k and consider a sequent Σ of degree $k+1$ whose atomic decomposition is paired and anchored. If Σ has the form $\Gamma \vdash \mathsf{t}\colon A \to B \rightsquigarrow t$, then we may choose terms x, r and x, r according to the decomposition of Σ so that we have a sequent $\Sigma' = \Gamma, \mathsf{x}\colon A \rightsquigarrow x \vdash \mathsf{r}\colon B \rightsquigarrow r$. But the decompositions of Σ and Σ' can be carried out so that they are identical; thus, the decomposition of Σ' is paired and anchored; by the inductive hypothesis, $\theta(\Sigma')$ is provable in **mon:LP**. But since $\theta(\Sigma)$ is derivable from $\theta(\Sigma')$ in **mon:LP** by an application of $R\to$, $\theta(\Sigma)$ is derivable in **mon:LP**. A similar argument holds for sequents derivable by the other 1-premise rule $L\otimes$: namely, if Σ is of the form $\Delta, \mathsf{t}\colon A \otimes B \rightsquigarrow t \vdash \mathsf{u}$: $C \rightsquigarrow u$, then $\theta(\Sigma)$ is derivable in **mon:LP**. Thus, we may assume that the succedent type of Σ is either an atomic type or a product, and that no antecedent type of Σ is a product.

Suppose, then, that the succedent type is an atom of form u: $C \rightsquigarrow u$; then μ pairs this atom with an atom of the form u': $C \rightsquigarrow u'$ derived by decomposition from the final type of a type of the form term: $D_k \to \cdots D_1 \to C \rightsquigarrow \mathit{term}$ ($k \geqslant 0$, with the understanding that if $k = 0$, then $D_k \to \cdots \to D_1 \to C = C$). (Note that u: $C \rightsquigarrow u$ cannot be paired with any non-leading type C' of an implicational type $C' \to E$, since the ϕ-term of C' is properly included in the ϕ-term of E (by the decomposition procedure) and the ϕ-term of E is included in $FA(\mathsf{u})$ (by term-linearity), so the ϕ-term of C' would have to be properly included in the ϕ-term of the atomic type it is paired with. Impossible!) If $k = 0$, we're done: the ϕ-term of the atomic type corresponding to the antecedent type C coincides under θ with $\theta(\mathsf{u})$ and is disjoint from the ϕ-term associated with any other antecdent type; but this is impossible if there is any other antecedent type, since (by term-linearity) its free atoms must be contained in $FA(u)$; hence, if $k = 0$, the sequent is an axiom instance, and we're done. If $k \geqslant 1$, let D_k^* be the multiset union of the atomic polar types derived from D_k and their pairs under μ; none of these atomic polar types is paired with any atomic polar type derived from the decomposition of $D_{k-1} \to \cdots \to C$, since this is prevented by the term conditions of the decomposition. Now note that the final type B of any implicational antecedent type with form $A_k \rightarrow \cdots \rightarrow A_1 \rightarrow B$ ($k \geqslant 0$) belongs to D_k^* if and only if all the atomic types derived from A_i ($0 \leqslant i \leqslant k$) belong to D_k. This partitions the multiset of antecedent types into two disjoint multisets of types: a multiset Δ, whose atoms belong to D_k^* and a multiset Θ, whose atoms do not. The atoms of the latter class must be paired with the atoms derivable from $D_{k-1}^* \sqcup \cdots D_1^*$. Thus, we have the following situation (with term-labels suppressed):

$$\frac{\Delta \vdash D_k \qquad \Theta, D_{k-1} \rightarrow \cdots \rightarrow D_1 \rightarrow C \vdash C}{\Theta, \Delta, D_k \rightarrow D_{k-1} \rightarrow \cdots \rightarrow D_1 \rightarrow C \vdash C}$$

The endsequent, which is identical to Σ, is derivable from the premises if they are provable. Since each of the premises inherits an atomic decomposition from the conclusion relative to which it is paired and anchored, the induction hypotheses assures us that they are derivable. Thus, so is the conclusion Σ.

A similar argument – using the linearity of terms and sequents to partition Σ into two smaller sets of paired and anchored terms corresponding to appropriate premises for the rule $R\otimes$ – applies in the case that the succedent type is a product.

This concludes the proof. □

Atomic polar decompositions, then, provide an alternative presentation of **mon:LP**. What is attractive about this presentation is the fact that proofs no longer display excessive and redundant bookkeeping: passive assumptions need not be carried along with each inference step; different orders of inference steps need not be addressed. These issues are compiled inside the unification algorithm.

9.3.2. *Indexed Terms*

There is another way to explicitly manage proof resources which is related to the atomic polar normal forms, but which is technically more complicated. Instead of decomposing types into atomic polar types, one associates each occurrence of an atomic type with an index and assigns the same index to the associated component of each dimension. Earlier, we translated the lexical type declaration for caught as:

$$\lambda \mathsf{x}.\ \lambda \mathsf{y}.\ \mathsf{y\ caught\ x}\colon np \rightarrow np \rightarrow s \rightsquigarrow \lambda x.\lambda y.caught'(x)(y)$$

This determines a multiset of atomic polar types which are linked by term inclusions:

$$[\mathsf{x}\colon np^0 \rightsquigarrow x, \mathsf{y}\colon np^0 \rightsquigarrow y, \mathsf{y\ caught\ x}\colon s^1 \rightsquigarrow \lambda x.\lambda y.caught'(x)(y)]$$

In the indexed format, this declaration takes the form below, with the links explicitly represented by a common set of indices:

$$[\mathsf{u}^i, \mathsf{v}^j].\ \mathsf{u}^i\ \mathsf{caught}\ \mathsf{v}^j\colon [np^i, np^j] \rightarrow s^k \rightsquigarrow [x^j, y^i].caught'(x^j)(y^i)$$

A system along these lines is midway between the system of implicit control found in type languages that rely on λ-terms and sequentially-residuated types for cross-dimensional control and the atomic polar normal forms just examined. It is like the implicit, sequential control systems in that residuated types and terms display a collection of argument types or argument variables initially. It is distinct from the sequentially-organized systems in that it is not required that the arguments be linearly ordered. Instead, like the atomic polar normal forms, there is a registration – here, using a common index set – of corresponding arguments in different dimensions, correlating exactly those components in a complex type which belong to individual atomic polar types in the atomic polar normal form. Moreover, within this framework, the rules for the type-constructors corresponding to products and residuals can be constructed in a non-deterministic way that combines (partial) application with composition in the way schematically indicated by the inference figure below (corresponding to the standard rule $L\rightarrow$):

$$\frac{\Gamma \overset{x}{\vdash} \{Z \sqcup np^a\} \qquad \{V \sqcup Z\} \rightarrow A \sqcup \Delta \overset{\xi}{\vdash} B}{\Gamma \sqcup \{V \sqcup np^b\} \rightarrow A, \Delta \overset{x,\xi,a=b}{\vdash} B}$$

In this rule, the index a is bound to b (as indicated in the unification bindings of indices above the turnstile), one of the arguments – np^b – disappears in the transition from the conclusion to the right-hand premise, but other possible arguments – indicated by V are carried along, and the arguments represented by Z appear in both premises. An elaboration of this schema allows a functional type to combine simultaneously with a number of arguments. Like the proof net systems discussed above, type systems constructed along these lines allow simpler proofs than the sequential systems. But their construction involves technical complexities which go beyond the bounds of the present paper.[8]

[8] After this paper was written, it came to my attention that there may be interesting connections between the system of indexed terms just sketched and the systems of application

10. Variant Systems

The system of labeled deduction **mon:LP** depends on taking the types of **LP** and labeling them with terms from a higher-order term algebra based on a free monoid. The resulting system allows relations to be defined between ϕ-terms and IL-terms that are not definable in a natural way in the substructural systems **AB**, **L**, or **LP**. A natural question to ask is what other systems arise as variants of this form of labeled deduction. We mention two possible modes of variation here, deferring discussion to another occasion.

The first possibility is to consider other algebraic possibilities as the basis on which higher-order ϕ-terms are constructed. For example, if we replace the *free monoid* V^* by the *free commutative monoid* **com**V^*, we have a system **common:LP** in which the ϕ-terms of atomic type are simply multisets of elements of V. This system is the analog within the landscape of ϕ-term labeled deduction of **LP**.

A second possibility is to consider variations on the set of admissible terms. One limitation already observed is to the set of *linear* λ-terms: terms in which each abstraction operator λx in a term of the form $\lambda x.\phi$ binds exactly one free occurrence of the variable x. But other forms of limitation are possible. For example, we may require that in $\lambda x.\phi$, either $\phi = (xt)$ or $\phi = (tx)$, limiting the depth of the free occurrence. The interesting consequences of this restriction – it disallows Division and the general form of Permutation, for example – and the question of how the standard categorial type systems can be modeled within the landscape of term-labeled deduction are discussed in more detail in [38].

11. Summary

The term-labeled categorial type systems studied in this paper are a form of *labeled deduction* [12]. It is not surprising that there are forms of labeled deduction corresponding to known cases of unlabeled deductive systems. Yet it is interesting to observe that the addition of ϕ-term labeling introduces new possibilities for the division of linguistic labor.[9] And among these new possibilities, we encounter not only familiar systems in a new

and abstraction developed in detail by Aczel ancl Lunnon in the context of Situation Theory [1]. These connections might be worth exploring.

[9] For other investigations of ϕ-term labeling in categorial grammar, see Moortgat [27, 28], Morrill [31], Hepple [18], and Calcagno [9]. The treatments in these papers start with the directional type-constructors of **L**, rather than the non-directional **LP** system adopted in **mon:LP**.

guise, but also unfamiliar systems with attractive properties. What I have tried to emphasize here is the interaction between term-labeled type inference and the behavior of quantifiers. This at least allows us to compare formally the behavior of the systems examined in this paper not only with regard to the relation they define between ϕ-terms and types, but with regard to the more informative relation among ϕ-terms, types, and *IL*-terms.

The examples examined above have been elementary ones. It will be useful to see whether such systems as **mon:LP** can be extended to more complex cases involving quantification and binding,[10] or the integration of prosodic properties of strings into proof-structures [44, 25, 36]. Research in these areas, which involve complex interactions among the properties of different linguistic dimensions, demonstrates the need for a deeper abstract understanding of the problem of generalized compositionality.[11]

References

1. P. Aczel: 1994, *Generalised Set Theory and the Modeling of Parametric Objects and Abstraction – a Preliminary Note*, Paper presented at the Conference on Information-Oriented Approaches to Language, Logic, and Computation. Saint Mary's College of California, June 13–15, 1994.
2. K. Ajdukiewicz: 1935, 'Die Syntaktische Konnexität', *Studia Philosophica* **1**, 1–27. (English translation in Storrs McCall (ed.), *Polish Logic*, Oxford University Press, 1967.)
3. Y. Bar-Hillel: 1953, 'A Quasi-Arithmetical Notation for Syntactic Description', *Language* **29**, 47–58; reprinted in Bar-Hillel (1964), pp. 61–74.
4. Y. Bar-Hillel: 1964, *Language and Information*, Addison-Wesley, Reading, MA.
5. J. van Benthem: 1986, *Essays in Logical Semantics*, D. Reidel, Dordrecht.
6. J. van Benthem: 1991, *Language in Action*, North-Holland, Amsterdam.
7. W. Buszkowski: 1988, 'Generative Power of Categorial Grammars', in R. T. Oehrle, E. Bach, and D. Wheeler (eds.), *Categorial Grammars and Natural Language Structures*, D. Reidel, Dordrecht, pp. 69–94.
8. W. Buszkowski: 1987, 'The Logic of Types', in J. Szrednicki (ed.), *Initiatives in Logic*, M. Nijhoff, Dordrecht.
9. M. Calcagno: 1994, *A Sign-Based Extension to the Lambek Calculus for Discontinuous Constituency*, ms., Dept. of Linguistics, Ohio State University.
10. B. Carpenter: 1994, *Quantification: a Deductive Account*, ms., CMU.
11. M. Dalrymple, J. Lamping, F. C. N. Pereira, and V. Saraswat: 1994, 'A Deductive

[10] See [37] for one approach to this problem.

[11] While this paper was under review, I received a copy of a paper by Dalrymple, Lamping, Pereira, and Saraswat [11] which contains an account of quantificational scope ambiguities with many attractive features. There are affinities between the system they propose and the system presented in this paper, including linear use of resources and a method of control relating semantic expressions to another dimension. Deeper understanding of the similarities and differences of these two approches would be interesting and useful.

Account of Quantification in LFG', in M. Kanazawa *et al.* (eds.), *Quantifiers, Deduction, and Context*, CSLI, Stanford, California.

12. D. Gabbay: 1991, *Labeled Deductive Systems*, ms., Imperial Collcge, London.
13. G. Gentzen: 1934–5, 'Untersuchungen über das Logische Schliessen', *Mathematische Zeitschrift* **39**, 176–210, 405–431. (English translation in M. E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam).
14. J.-Y. Girard: 1987, 'Linear Logic', *Theoretical Computer Science* **50**, 1–102.
15. J.-Y. Girard, Y. Lafont, and P. Taylor: 1989, *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, Cambridge.
16. H. Hendriks: 1987, 'Type Change in Semantics: the Scope of Quantification and Coordination', in E. Klein and J. van Benthem (eds.), *Categories, Polmorphism and Unification*, University of Edinburgh: Centre for Cognitive Science, and Amsterdam: Institute for Language, Logic, and Information.
17. H. Hendriks: 1989, *Cut Elimination and Semantics in Lambek Calculus*, ms., ITLI, Department of Philosophy, University of Amsterdam.
18. M. Hepple: 1994, *Discontinuity and the Lambek Calculus*, ms., Department of Computer Science, University of Sheffield.
19. J. R. Hindley and J. P. Seldin: 1986, *Introduction to Combinators and λ-Calculus*, London Mathematical Society Student Texts 1. Cambridge University Press, Cambridge.
20. W. A. Howard: 1980, 'The Formulae-as-Types Notion of Construction', in J. R. Hindley and J. P. Seldin (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press.
21. P. T. Johnstone: 1987, *Notes on Logic and Set Theory*, Cambridge University Press, Cambridge.
22. E. L. Keenan: 1987, 'Semantic Case Theory, in J. Groenendijk, M. Stokhof, and F. Veltman (eds.), *Proceedings of the Sixth Amsterdam Colloquium*, ITLI, University of Amsterdam.
23. J. Lambek: 1958, 'The Mathematics of Sentence Structure', *American Mathematical Monthly* **65**, 154–170.
24. D. Miller: 1991, 'A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification', *Journal of Logic and Computation* **1**(4), 497–536.
25. M. Moortgat: 1988, *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*. Foris, Dordrecht.
26. M. Moortgat: 1992, 'The Logic of Discontinuous Type Constructors', in W. Sijtsma and A. van Horck (eds.), *Discontinuous Constituency*, Mouton de Gruyer, Berlin.
27. M. Moortgat: 1990, 'Unambiguous Proof Representations for the Lambek Calculus', in *Proceedings of the 7th Amsterdam Colloquium*, ITLI, University of Amsterdam.
28. M. Moortgat: 1991, 'Labeled Deductive Systems for Categorial Theorem Proving', *Proceedings of the 8th Amsterdam Colloquium*, Universiteit van Amsterdam.
29. M. Moortgat and R. T. Oehrle: 1993, *Categorial Grammar: Logical Parameters and Linguistic Variation*, Lecture notes, European Summer School in Logic, Language, and Information. Faculdade de Letras, Universidade de Lisboa, Portugal.
30. M. Moortgat and R. T. Oehrle: *Elements of Categorial Grammar*, in preparation.
31. G. Morrill: 1994, *Clausal Proof Nets and Discontinuity*, Paper presented at the London Workshop on Proof Theory and Linguistic Analysis.
32. R. T. Oehrle: 1981, 'Lexical Justification', in M. Moortgat, H. v.d. Hulst, and T. Hoekstra (eds.), *The Scope of Lexical Rules*, Foris, Dordrecht, pp. 201–228.
33. R. T. Oehrle: 1992, 'Dynamic Categorial Grammars, in R. Levine (ed.), *Formal Grammar: Theory and Implementation*, Vancouver Studies in Cognitive Science, Vol. 2, Oxford University Press, Oxford, pp. 79–128.
34. R. T. Oehrle: 1988, 'Multi-Dimensional Compositional Functions as a Basis for Grammatical Analysis', in R. T. Oehrle, E. Bach, and D. Wheeler (eds.), *Categorial Grammars and Natural Language Structures*, D. Reidel, Dordrecht, pp. 349–389.

35. R. T. Oehrle: 1990, 'Categorial Frameworks, Coordination, and Extraction', in A. Halpern (ed.), *Proceedings of the Ninth West Coast Conference on Formal Linguistics*, CSLI, Stanford, pp. 411–425.
36. R. T. Oehrle: 1991, *Grammatical Structural and Intonational Phrasing: a Logical Perspective*, Working papers of the AAAI Fall Symposium on Discourse Structure in Natural Language Understanding and Generation, Asilomar.
37. R. T. Oehrle: 1992, *Referential Types, Dynamic Context, and Referential Relations*, ms., University of Arizona, Tucson.
38. R. T. Oehrle: 1994, 'Some 3-Dimensional Systems of Labeled Deduction', *Proceedings of the London Workshop on Proof Theory and Linguistic Analysis*, In preparation.
39. F. C. N. Pereira: 1990, 'Prolog and Natural-Language Analysis: into the Third Decade', in S. Debray and M. Hermenegildo (eds.), *Logic Programming: Proceedings of the 1990 North American Conference*, MIT Press, Cambridge, MA.
40. F. C. N. Pereira: 1991, 'Semantic Interpretation as Higher-Order Deduction', in J. van Eijck (ed.), *Logics in AI*, Springer, Berlin, pp. 78–96.
41. D. Roorda: 1990, *Proof Nets for Lambek Calculus*, ms. ITLI, University of Amsterdam.
42. D. Roorda: 1991, *Resource Logics: Proof-Theoretical Investigations*, Ph.D. thesis, Faculteit van Wiskunde en Informatica, Universiteit van Amsterdam.
43. R. Smullyan: 1968, *First-Order Logic*, Springer, Berlin.
44. M. Steedman: 1991, 'Structure and Intonation', *Language* **67**, 260–296.
45. R. Thomason: 1974, *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven.
46. L. A. Wallen: 1990, *Automated Proof Search in Non-Classical Logics: Efficient Matrix Methods for Modal and Intuitionistic Logics*, MIT Press, Cambridge, MA.