A Deductive Calculus of Categories and Transformations

Thomas L. Cornell*
University of Tübingen
Seminar für Sprachwissenschaft
cornell@sfs.nphil.uni-tuebingen.de

March 10, 1998

1 Introduction

Many researchers in the past few years have remarked on the apparent relation between ideas from Chomsky's minimalist program (Chomsky 1995) and fundamental ideas in categorial grammar.[1] In this paper we investigate this connection in some detail. We present a formalization of some ideas from Chomsky (1995) in the framework of multimodal type logical grammar (e.g., Morrill 1994, Moortgat 1996, and refs.). This project is in many respects rather straightforward, but makes use of aspects of the type logical machinery which are generally not used, or at least not used for these purposes, in categorial grammar. This allows us to investigate to some degree the relation between analyses of extraction phenomena based on hypothetical reasoning—the classical categorial approach—and those based on movement, seen here as a complex of structural rules. Besides these concerns relating different frameworks we can also use these techniques to deal with theory internal questions. In this paper we will use this framework to investigate some questions that have dogged transformational grammar since the introduction of trace theory in the early 1970s. In particular, we investigate the relations between derivational and representational means for expressing properties of syntactic structures and observe, via the well-known Curry-Howard correspondence between proofs and terms, an

*This paper began life as an extended version of the handout for a talk presented at the Formal Grammar conference in Aix-en-Provence on August 10, 1997. A preconference proceedings paper is available as Cornell (1997). However, very many improvements have been made since then. This research in all its phases has benefitted tremendously from the help of many people, in particular Dick Oehrle, Michael Moortgat, Ed Stabler and Uwe Mönnich. Much is due to them, but the errors that remain are all my own.

[1]See especially Stabler (forthcoming), but also Berwick and Epstein (1995), and Lecomte (1997) address these issues as well.

extremely tight connection between chain-based approaches—especially as developed in recent work by Brody (see esp. Brody 1997)—and movement based approaches such as that of Chomsky (1995).

1.1 Features and Formulas

The main feature of minimalist derivations which has suggested a connection with categorial grammar is the treatment of *feature checking* in Chomsky (1995), which is very reminiscent of the basic *cancellation rules* of categorial grammar. Consider the following instance of a $L\backslash$ rule, a "rule of use" in a Gentzen style presentation of a Lambek calculus.

$$\frac{K \vdash K \quad V \vdash V}{(K, K\backslash V) \vdash V} L\backslash \tag{1}$$

Suppose we interpret K as the case feature of a DP, and the structure $(K, K\backslash V)$ to be a checking configuration. Then this instance of a $L\backslash$ rule corresponds fairly closely to the minimalist idea that an expression marked with a case feature can cancel that feature in a checking configuration (e.g., the specifier position) with a head marked with the same feature. In effect, then, the axiom $K \vdash K$ marks the feature as cancelled.

Chomsky's principle of *Last Resort*, though it appears in several forms in Chomsky (1995), essentially amounts to the following once a theory of feature checking is in place.

> Transformational derivations should be driven by the need to check features.

In a deductive setting, with feature checking implemented along the lines sketched above, this means neither more nor less than that:

> Proofs are driven by the need to reduce a theorem to axioms.

so this key idea of the minimalist program would receive a purely formal explanation in a deductive setting. Seen in this light, certain aspects of minimalist machinery may appear as special cases of more general mechanisms with a broader foundation in mathematics and logic. It should benefit current syntactic theorizing to know to what extent the theory of syntax can call upon well established techniques and results, the better to isolate what makes the formal system of grammar special.

The $L\backslash$ rule exemplified above is just a directional variant of a rule essentially corresponding to Modus Ponens ("Given A and the implication $A \Rightarrow B$, conclude B,") and so common to a great many deductive systems. What sets categorial grammar style deduction apart in this universe is its strict attention to the way in which deductions *consume resources.* In particular, an assumption once used cannot normally be used again; a feature once cancelled remains cancelled, in effect. This attention to the consumption of resources adds another dimension to the sense in which notions of economy are critical to the minimalist program.

This idea of resource sensitivity can be made precise by comparing the following two variants of a rule for drawing conclusions of the form "A and B."

$$\frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \times B} R\times \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \sqcap B} R\sqcap \tag{2}$$

In the $R\times$ rule, we must divide up our antecedent hypotheses into those which we will use to prove A and those which we will use to prove B. In the $R\sqcap$ rule, we need not do this. Later on we will simply throw away any elements of Γ which we do not need; in the meantime we are free to use the same assumption to prove both A and B, if that is useful.

In linear logic (Girard 1987, Troelstra 1992), one systematically splits the familiar logical connectives like conjunction in just this way, into a "multiplicative" variant, like our $\times$, and an "additive" variant, like our $\sqcap$. In this context, the very resource conscious logics of categorial grammar can be seen as implementing a multiplicative fragment of linear logic.

In contrast to derivations in categorial grammar, the features which drive minimalist transformational derivations are often *subcategorial*, "morphological" features like Case or Wh. Typically, lexical items must be assigned to multiple types, but in implementing the minimalist approach to features and derivations they must be assigned to multiple types in a resource sensitive manner. So we are led to conclude that,

> Subcategorial features can be resources, too.

at least insofar as they drive derivations.

Accordingly, we allow *mulitplicative* product types in our lexicon. For example, a Wh determiner like *which*, needing case and selecting a noun type complement might be represented with a formula like

$$(D \bullet K \bullet W)/N, \tag{3}$$

where we use K for Case and W for Wh. (We treat $\bullet$ as right branching by default, so as to avoid unnecessary parentheses.) This formula represents a function which takes an argument of type N and returns a value whose type is given by the tuple of features $D \bullet K \bullet W$, the type of elements which have all three characteristics, but where those characteristics may have effects on different parts of the syntax.[2]

Just as in categorial grammar the local combinatorial potential of phrases is given in advance by the type of the lexical head, so in this system the movement potential of a phrase (i.e., its nonlocal combinatorial potential) is given in

[2]Other approaches to multiple type assignment, such as Bayer and Johnson (1995) do so in an essentially *additive* manner, so that extra features can be thrown away. Heylen (1997) uses unary modal operators to implement features, but again in such a way that extra features can simply be disregarded. The problems these authors address are common to all frameworks, and so we cannot ignore the necessity of some resource insensitive treatment of features. Clearly both treatments can coexist, but we cannot at this point offer any systematic or predictive theory of how they do so.

advance by the type of its head. For example, the WhP headed by *which*, given the type assignment in (3), will have to occupy three possibly distinct syntactic positions in order to cancel its three "chain" features.

> Roughly: '/' and '\' say how to build phrases, '•' says how to build chains.

In this sense, the approach to features in the minimalist program represents a sort of extension to the approach to features in categorial grammar. One can be more concrete: for Chomsky, the operation Merge is cost-free, but it is subject to some sort of selectional/thematic restrictions which are extra syntactical. For categorial grammarians, the main focus is on (categorial) selection properties of heads: selection features are in an interesting sense logically dual to the notion of syntactic category. Avoiding an argument—for the moment at least—over who gets to be Syntax, we can see here an interesting complementarity of focus: on selection and local grammatical relationships on the part of categorial grammar and on extraction and nonlocal grammatical relationships on the part of the minimalist program. It now appears that both a categorial theory of selection and a minimalist theory of movement can live together within the same logical framework, which we will try to develop in some detail in this paper.[3]

This complementarity of focus is already knitted together to a considerable degree in the work of Michael Brody. Brody develops a theory of chains in which the root position (the deepest chain position) is constrained to be the sole position at which a number of properties including selection can hold (and refs. Brody 1995a, 1997). (This is the Generalized Projection Principle of Brody (1997).) Chomsky, on the other hand, has his theory of feature-driven derivation explicitly given as primarily a theory of the relation between specifically nonroot chain positions and functional categories.

1.2 Numerations and Sequents

As presented in Chomsky (1995), a derivation proceeds from an initially given *numeration*, which is supposed to be a *multiset* of lexical item occurrences. Lexical elements are withdrawn from the numeration and subjected to the action of a pair of *generalized transformations*, **Merge** and **Move**, which transform the raw lexical resources of the numeration into an LF representation, delivering an intermediate representation for PF spell-out along the way. The operation of Select forms syntactic objects from numeration items and the operations of **Merge** and **Move** form new syntactic objects from old. We observe that there is in fact no formal difference between the numeration and the set of syntactic items: both are multisets, in particular, so for the purposes of this paper we will not distinguish between these two structures, using the term "numeration" indiscriminately for both. At the same time, we do away with the Select operation.

[3]In this we are essentially following Stabler (1997b,a, forthcoming), who incorporates a theory of selection based on Koopman (1994) in his algebraic formalization of minimalist derivations.

In the setting of a grammar logic we want the lexical item instances to be type formulas, and we want to isolate in particular collections of such lexical formulas which can be combined to form sentences. We represent the fundamental decision problem of a grammar logic as the question of whether or not a sequent of the form

$$\text{lexical resources} \vdash \text{sentence-category}$$

is derivable.

Grafting minimalist ideas into this setting seems fairly natural: we are interested in sequents of the form

$$\text{Numeration} \vdash C$$

where C (complementizer) is the sentence category.

1.3 Checking Configurations

Chomsky (1995) defines the *checking domain* of a head in such a way that it includes all head-internal positions and specifier positions, and it excludes the complement position, which becomes the *attraction domain*. The flow of movement is then from the attraction domain and into the checking domain.

Chomsky (1995) lacks a theory of selection. To a point one can imagine categorial grammar as providing a theory of selection and the minimalist program as providing a theory of attraction. We wish to combine both theories here, to obtain a theory which is more *uniformly resource conscious*. So here even the complement position will be a checking position, but only for selection features. Note however that we will not trouble ourselves much here over the distinction between features belonging to these two systems, except to suggest along the way how it could be done.

As we will see, the standard $L/$ and $L\backslash$ rules of the Lambek calculus require that the functor and its argument be sisters in the sequent antecedent. In minimalist syntactic structures, on the other hand, in the typical case of the Spec-Head configuration and even—following head movement—in some cases of the Complement-Head configuration as well cancelling pairs of features are not in a sisterhood configuration. That is, there is a mismatch between the definition of checking configuration in the minimalist program and the definition of the equivalent notion in proof theory. Moortgat and Oehrle (1997) offer two sorts of solutions for this sort of problem: “structural normalization” and “structural transparency”. The former adds controlled structural rules so as to allow the relevant resources to be placed into the canonical sister configuration; the latter allows intervening structure to be ignored in the relevant cases. Here we use the former approach: once the generalized transformation **Move** has acted to place features in checking configurations, a further suite of structural rules allows the feature to be cancelled to move into position as a sister of the cancelling functor. We think of this extra set of structural rules as providing a definition of minimalist checking configuration in terms of the more local configurations which are more convenient to proof theory.

2 The Grammar Logic

We introduce the logic which implements the ideas of section 1 in two pieces. In modern proof theory it is quite common to separate the inference rules of a logic into two components: the "logical" or "connective" rules and the "structural" rules. The logical rules are the rules which implement the definitions of the logical operators, that is, which assure us that a conjunction operator behaves like conjunction, an implication operator like implication and so forth. Structural rules deal with the relations among formulas, in ignorance of the logical connectives from which those formulas are built. For example, the rule of Exchange allows us to take a pair of formulas (A, B) and proceed with the proof by replacing that pair with the permuted pair (B, A). Whether A is a conjunction or an implication or whatever makes no difference to the applicability of Exchange. For example, the calculus presented in Lambek (1961) has the same connectives as the calculus presented in Lambek (1958), but it is a different logic because it lacks the structural rule of Associativity.

In a fairly natural sense, the nonassociative Lambek calculus **NL** of Lambek (1961) is the minimal grammar logic, in that it contains no structural rules at all. We will lay out its logical inference rules in the next subsection but consider for now one of these rules, called $L\bullet$ because it tells us how to proceed with a proof search when we encounter a formula with a $\bullet$ connective on the left of the $\vdash$ symbol. (Here the notation $\Delta[X]$ means a collection of formulas in which there is a distinguished occurrence of X; subsequent use of $\Delta[Y]$ should be taken to mean the same structure but with Y in place of that occurrence of X.)

$$\frac{\Delta[(A, B)] \vdash C}{\Delta[A \bullet B] \vdash C}\, L\bullet \tag{4}$$

We eliminate an occurrence of $\bullet$ in this rule, but we introduce at the same time a pair of parentheses. Without Associativity, the sequent $D, (A, B)$ is not at all the same as the sequent $(D, A), B$, so we have to keep track of the associations between our formulas. In effect our sequents are no longer made out of simple sequences of formulas (the notion of "sequence" being inherently associative) but rather out of *trees* of formulas, and a sequent is a pair consising of a tree on the one hand and a formula on the other.

A significant recent innovation in this family of rather weak logics has been the development of multimodal grammar logics (see Moortgat 1996, for an overview). In these logics we can have more than one product operator, and so more than one kind of slash as well. In addition one can have "product" operators which are not binary. By far the most common extension in this direction has been the use of unary modal operators. The key to allowing these various different "modes of grammatical composition" to live together in a single sound and complete logic is to label the parentheses introduced by the various $L\bullet_i$ rules (and $L\Diamond_i$ rules for the "unary products") with an indication of which product introduced them. For example:

$$\frac{\Delta[(A, B)^i] \vdash C}{\Delta[A \bullet_i B] \vdash C}\, L\bullet_i \tag{5}$$

or the following:

$$\frac{\Delta[\langle A\rangle^i] \vdash C}{\Delta[\Diamond_i A] \vdash C}\, L\Diamond_i \tag{6}$$

Then the rules governing the corresponding residuals require that the active formulas be associated with the correct brackets. For example:

$$\frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma,\ A\backslash_i B)^i] \vdash C}\, L\backslash_i \qquad \frac{(A,\ \Delta)^i \vdash B}{\Delta \vdash A\backslash_i B}\, R\backslash_i \tag{7}$$

In effect we have several different sorts of functions (depending on the number of products, i.e., the number of subscripts) and corresponding to this we have several different sorts of function application. Now a function of a certain sort can only be applied to an argument if it is applied in the right way. We can do nothing with a configuration of the form $(A,\ A\backslash_i B)^j$, for example.

Given such labeled trees of formulas, one can write structural rules which relate them. For example, a subtree of the form $\langle\Delta\rangle^1$ might be allowed to permute with another subtree where permutation was barred for all other structures. Or to consider our previous example, we might in certain circumstances be able to convert a j type application to an i type application, allowing us to use a configuration like $(A,\ A\backslash_i B)^j$ after all. Our implementations of Chomsky's Merge and Move will be constructed out of precisely such modally controlled packages of structural rules, which we will describe in section 2.2, after we introduce the basic logic.

2.1 Connectives of the Grammar Logic and Their Inference Rules

Formally, the language of our logic is defined as follows. (This is an utterly standard definition and we present it rather tersely here. The interested reader is refered to Moortgat (1996) for a more detailed treatment.) We assume given a set of basic type symbols *Basic*. We define the well-formed type formulas with the following little grammar.

$$\mathcal{F} \longrightarrow \mathit{Basic} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F}\backslash_i\mathcal{F} \mid \mathcal{F}/_i\mathcal{F} \mid \Diamond_i\mathcal{F} \mid \Box_i^{\downarrow}\mathcal{F} \tag{8}$$

We refer to $\backslash_i$ and $/_i$ as the (left and right) residuals of $\bullet_i$, and to $\Box_i^{\downarrow}$ as the residual of $\Diamond_i$. They are required to obey the following *residuation laws.*

$$B \vdash A\backslash_i C \longleftrightarrow A \bullet_i B \vdash C \longleftrightarrow A \vdash C/_i B \tag{9}$$

$$\Diamond_i A \vdash B \longleftrightarrow A \vdash \Box_i^{\downarrow} B \tag{10}$$

Here the i are taken to range over some index set, but in practice we will often substitute different symbols for the different $\bullet_i$. For example we might write $\bullet_0$ as $\otimes$ and $/_0$ as $/_\otimes$.

From formulas in $\mathcal{F}$ we construct structured configurations of formulas (sometimes called "structured databases of resources") as follows.

$$\mathcal{S} \longrightarrow \mathcal{F} \mid (\mathcal{S},\ \mathcal{S})^i \mid \langle \mathcal{S} \rangle^i \tag{11}$$

A *sequent* here will be a pair consisting of a configuration and a formula, written $\mathcal{S} \vdash \mathcal{F}$. Inference rules will then be templates with one or two upper sequents ("premises") and one goal sequent ("conclusion"), and proofs will be trees labeled with sequents all of whose local trees are instances of inference rules or else (at the leaves) of axiom schemata.

We will present our inference rules and proofs in the style of a Gentzen sequent calculus. This means that every connective will have a "left rule" and "right rule" for dealing with sequents in which that connective appears either to the left or right of the $\vdash$ symbol. The Gentzen system was originally developed because it makes searching for a proof of a given sequent much easier than a Hilbert system or a Natural Deduction system. Because of this emphasis on *proof search*, rather than *proof* per se, we typically read Gentzen inferences from the bottom up, even though logically the direction of inference is from the top down. So, for example, we tend to think of the $L\bullet_i$ rule as telling us what to do when a $\bullet_i$ appears on the left in the goal sequent, though logically it is telling us under what structural conditions in the premises we can infer a $L\bullet_i$ on the left in the conclusion. The left and right rules are sometimes referred to as "rules of use" and "rules of proof", respectively. The left rule $L\bullet_i$ tells us under what structural conditions we can use a $\bullet_i$ to prove a consequent, and the right rule $R\bullet_i$ tells us under what structural conditions we can prove a consequent with a $\bullet_i$ as its main connective.

The logical inference rules for constructing derivations over such formulas are given in figure 1. Note that, read bottom up, each rule (other than the Cut rule) removes a logical operation from the sequent to be proved, so at each step the problem of finding a proof gets strictly smaller. We chose this format here because in addition to making it easier to construct proofs it also makes much clearer the role which the structure of antecedent formulas plays in the proof. As a result, again reading bottom up in the direction of proof search rather than of inference, Gentzen style proofs will be seen to look very similar to minimalist derivations (see section 3).

So far we have presented the logic completely abstractly, without regard for the inventory of connectives we will actually need. The considerations of section 1 lead us to adopt a basic language with the product connectives in table 1. Keep in mind that these are just convenient symbols for the "official" $\bullet_i$ notation, so (a) they all have residuals and (b) the rules of figure 1 apply to all of them and to all of their residuals.

In fact, however, though we will make use of all of these products, we will make use of the residuals of only $\bullet$ and $\Diamond_0$. That is, the logic allows us to reason

$$A \vdash A \quad Axiom \qquad \frac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,Cut \tag{12}$$

$$\frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(B/_i A, \Gamma)^i] \vdash C}\,L/_i \qquad \frac{(\Delta, A)^i \vdash B}{\Delta \vdash B/_i A}\,R/_i \tag{13}$$

$$\frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma, A\backslash_i B)^i] \vdash C}\,L\backslash_i \qquad \frac{(A, \Delta)^i \vdash B}{\Delta \vdash A\backslash_i B}\,R\backslash_i \tag{14}$$

$$\frac{\Delta[(A, B)^i] \vdash C}{\Delta[A \bullet_i B] \vdash C}\,L\bullet_i \qquad \frac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{(\Delta_1, \Delta_2)^i \vdash A \bullet_i B}\,R\bullet_i \tag{15}$$

$$\frac{\Gamma[A] \vdash B}{\Gamma[(\Box_i^{\downarrow} A)^i] \vdash B}\,L\Box_i^{\downarrow} \qquad \frac{(\Gamma)^i \vdash A}{\Gamma \vdash \Box_i^{\downarrow} A}\,R\Box_i^{\downarrow} \tag{16}$$

$$\frac{\Gamma[(A)^i] \vdash B}{\Gamma[\Diamond_i A] \vdash B}\,L\Diamond_i \qquad \frac{\Gamma \vdash A}{(\Gamma)^i \vdash \Diamond_i A}\,R\Diamond_i \tag{17}$$

Figure 1: Logical rules for a general multimodal type logic.

$\otimes$	— The numeration forming operator.
$\bullet$	— The lexical product.
$\Diamond_0$	— For feature checking.
$\Diamond_1$	— For implementing **Move**.
$<, \rhd$	— Complement and specifier attachment products.

Table 1: Connectives of the Grammar Logic.

about and calculate with the full range of connectives, but we will be interested only in *lexical* type formulas, that is, type formulas assigned to lexical items, and in our minimalist lexicons only the residuals of $\bullet$ (written simply $\backslash$ and $/$) and of $\Diamond_0$ (written $\Box_0^{\downarrow}$) will appear. (The unary modal $\Diamond_1$ will also appear, but not its residual.) No rule introduces a connective into a premise, so in general our formulas will be restricted to connectives appearing in the lexicon. The extra products will appear only as labels on structured configurations, and will be put there by structural rules, considered in the next section. It is important to keep in mind that this system is investigated here as a useful device for doing scientific calculations and not as a set of formal universals expressing a theory of Universal Grammar. No special claims are being made here about formal properties of UG. Indeed, any attempt to stipulatively restrict allowable proofs to use only left rules would destroy any chance at completeness for the logic and we would also lose the ability to construct proofs which will be very useful to us below. We will see when we consider the relation of transformational derivations to categorial derivations in section 5 that we can make very good use of the fact that our derivations are embedded in a full grammar logic.

2.2 Packages of Structural Rules

As far as the rules in (12)–(17) are concerned, any binary product is like any other binary product and so also for unary products and all associated residual operators. It is in their access to structural rules that the different connectives differ in more than just typography. We discuss briefly here what we intend for all of these connectives, and then present the actual packages of structural rules which implement these intentions.

Numerations are meant to be multisets, that is, the only thing that matters is the number of times a given structure occurs. Order and hierarchy are irrelevant there. Accordingly, we want the numeration forming product $\otimes$ (and its corresponding "comma", $(\cdot,\cdot)^{\otimes}$) to allow freely the permutation and reassociation of its arguments. Because of this, there is no real need to write the parentheses in a structure built up in the $\otimes$ mode, and the order of formulas is a strictly typographical necessity. Accordingly, we will use a "sugared notation" for the corresponding resource structures. That is, we use unbracketed commas to concretely present the numeration part of a sequent, writing $A, B \vdash C$, rather than $(A, B)^{\otimes} \vdash C$. For example, the antecedent of the sequent $A, B, (C, D)^{<} \vdash E$ is a numeration with 3 elements: two formulas and the tree $(C, D)^{<}$. Unbracketed substructures are granted free access to permutation and association rules, making numerations behave as multisets, as in Chomsky (1995).

For our purposes the most important connectives are $\bullet$ and its residuals $\backslash$ and $/$. These are the connectives which glue features together in lexical items, and they govern the projection of lexical properties into the syntax. This introduces a few special technical problems which have to be dealt with. Formally, the rule cancelling a feature A against a head marked B/A can only take place in a structure with the bracketing $(B/A,\ A)^{\bullet}$, because $/$ is the residual of $\bullet$. The configuration $(B/A,\ A)^{\bullet}$ is, then, the checking configuration which we get, for free, from the logic of residuation. From Chomsky (1995), however, we already have a notion of checking configuration given to us. For example—and indeed this is taken as the paradigm example—the specifier-head configuration is a checking configuration for formal features. In our language we could represent such configurations as resource structures $(A,\ (A\backslash B,\ C)^{<})^{\triangleright}$. This is not the configuration we want, on two counts: first it is structurally wrong, in that the functor and its argument are not structurally adjacent, and second it is mistyped, since $\backslash$ is neither the residual of $\triangleright$ nor of $<$. Formally, the notion of residuation gives us a primitive, mathematically elegant basis for investigating feature cancellation, but we still have to say explicitly, in these terms, what we mean a checking configuration in natural language syntax to be. It is for this reason that we provide the unary modal $\Diamond_0$. This will allow us to say exactly what we mean by a checking configuration (see below, in section 2.2.4), in a way which can change as we gain new empirical insights without changing the underlying mathematical basis of our calculational machinery. Looking ahead, $\Diamond_0$ will give features access to restricted forms of associativity and permutation, which will allow them to move about the resource structure to find the head in whose checking domain they are. In this sense feature cancellation is very much like movement, and in particular it may help to clarify the intuition that, if it weren't for the externally imposed necessity to pied pipe elaborate interface structures, then feature cancellation would just be head movement (as it is supposed to be in covert cases). The relation between $\Diamond_0$ and $\Diamond_1$, which licenses access to movement rules, will be laid out in section 2.2.3.

The notation for the last two "tree-building" products $\triangleright$ and $<$ is derived from the algebraic systems for minimalist derivations of Stabler (Stabler 1997b,a, forthcoming). They can be thought of as *headed* operators, which point in the direction of the head, so that a structure written $(\alpha < \beta)$ would correspond to the structure, written Chomsky-style as $\{\alpha, \{\alpha, \beta\}\}$, in which α and β were merged with α projecting over β.

2.2.1 The Numeration Package

We have already noted that the numeration forming product $\otimes$ is associative and commutative. Formally we have the following rules. (The double line in rule $\otimes A$ means that the inference can go in either direction.)

$$\frac{\Gamma[(\Delta_1, \Delta_2)^{\otimes}] \vdash A}{\Gamma[(\Delta_2, \Delta_1)^{\otimes}] \vdash A}\otimes P \tag{18}$$

$$\frac{\Gamma[(\Delta_1, (\Delta_2, \Delta_3)^{\otimes})^{\otimes}] \vdash A}{\Gamma[((\Delta_1, \Delta_2)^{\otimes}, \Delta_3)^{\otimes}] \vdash A}\otimes A \tag{19}$$

As noted above, we will use a syntactically sugared presentation of these structures and just as we will typically not bother to write the parentheses we will also typically not mention the implicit application of these rules, except where confusion may arise.

The rules $\otimes P$ and $\otimes A$ have a special feature: $\otimes$, or rather its associated configuration-forming comma and brackets, is the only connective which appears. Many of the structural rules to be introduced in the following subsections involve several products at a time, in one of two ways. Some rules simply allow us to convert one product to another. Logically, this represents the claim that the individuals constructed with the first product form a subset of the individuals constructed with the second. The most important rules from the standpoint of transformational grammar, however, are the rules which govern the interaction of several products. We will see many examples of these inclusion and interaction rules in what follows.[4]

2.2.2 Merge Package

Our next example is only slightly more complex: the generalized transformation **Merge**. The essential idea is that **Merge** takes a pair of resources which are more or less free-floating in the numeration and freezes them in a structure in which their sequential and hierarchichal positions are fixed.

$$\frac{\Gamma, (\Delta_1, \Delta_2)^{<} \vdash A}{\Gamma, \Delta_1, \Delta_2 \vdash A}\, Mg1 \tag{20}$$

$$\frac{\Gamma, (\Delta_2, \Delta_1)^{\triangleright} \vdash A}{\Gamma, \Delta_2, \Delta_1 \vdash A}\, Mg2 \tag{21}$$

Rule $Mg1$ corresponds to the attachment of a complment and $Mg2$ to the attachment of a specifier. Which rule we use depends on features of the head (Δ_1 in both cases): if Δ_1 is a formula of the form B/A, then $Mg1$ applies. Otherwise $Mg2$ applies. We require in addition that the Δ_i not contain any instances of $\otimes$ formulas or $\otimes$ structures. Intuitively, we should only be able to merge structures which were themselves built up by **Merge**. We can impose this condition by considering $Mg2$ to depict a family of rules in which Δ_1 is given either as a formula other than B/A or $A \otimes B$, or a structural configuration annotated with

[4] Note that, logically, all of these structural rules are postulates, that is, new axiom schemata. We can convert such postulates into inference rules by "compiling in a Cut" as follows. Suppose we have the postulate

$$A \otimes (B \otimes C) \longrightarrow (A \otimes B) \otimes C$$

Then subproofs of the following form are always possible.

$$\frac{\dfrac{}{A \otimes (B \otimes C) \vdash ((A \otimes B) \otimes C)}\, Axiom \qquad \dfrac{\vdots}{\Gamma[((A \otimes B) \otimes C)] \vdash D}}{\Gamma[A \otimes (B \otimes C)] \vdash D}\, Cut$$

Essentially the same technique allows us to generalize to resource structures of the form $((\Delta_1, \Delta_2)^{\otimes}, \Delta_3)^{\otimes}$ in place of $((A \otimes B) \otimes C)$, for example, giving us the form of the inference rule as it is presented here.

a product other than $\otimes$. Because such products cannot be introduced except by converting a $\otimes$ product, a simple induction shows that such an approach does indeed implement this side condition in the logic. Such an approach is cleaner because we actually can prove theorems in which $\otimes$ configurations do appear embedded in, say, $<$ configurations, but only when the theorem is of that form to start with. We are only assured that when the antecedent of the theorem is a numeration, i.e., a pure $\otimes$-structure of lexical type formulas, then our side condition above is guaranteed to hold.[5]

2.2.3 Move (XP) Package

This package implements XP movement, and can be used to implement a very restricted form of head movement.[6] Out approach implements a technique which is quite often used in applications of multimodal calculi to linguistic problems. The basic idea is to observe first that the sequent $\Diamond_i \Box_i^{\downarrow} A \vdash A$ is valid, for all formulas A in $\mathcal{F}$.

$$\dfrac{\dfrac{A \vdash A}{\langle \Box_i^{\downarrow} A \rangle^i \vdash A}\, L\Box_i^{\downarrow}}{\Diamond_i \Box_i^{\downarrow} A \vdash A}\, L\Diamond_i \tag{22}$$

Now suppose we have some structural rules which are sensitive to the presence of these $\langle \cdot \rangle^i$ structures, for example the following.

$$\frac{\Gamma[(\Delta_1, \langle \Delta_2 \rangle^i)^j] \vdash A}{\Gamma[(\langle \Delta_2 \rangle^i, \Delta_1)^j] \vdash A} \tag{23}$$

If we provide a feature B in the lexicon with the prefix "$\Diamond_i \Box_i^{\downarrow}$", then we have in effect given it a license to permute, after which the diamond that allowed permutation is removed and the position of the feature is fixed. Given such a rule the sequent

$$(A \backslash_j B,\ \Diamond_i \Box_i^{\downarrow} A)^j \vdash B$$

is valid but

$$(A \backslash_j B,\ A)^j \vdash B$$

is not valid, nor is

$$(A \backslash_j B,\ \Diamond_i \Box_i^{\downarrow} A)^{k \neq j} \vdash B.$$

[5]Such a move would allow us to investigate, for example, grammars in which completely nonconfigurational domains could be embedded in much more rigid structures. Such an investigation could then be carried out in a uniform framework in which the rest of the implementations of minimalist ideas were left intact and only this one condition allowed to vary.

[6]We will see examples of this in section 3. Basically, we will only be able to move heads when they are linearly adjacent to their targets. This may be good enough, since cancelled material does not count for adjacency. We discuss head movement on its own in some detail in Cornell (1997).

The unary operator $\Diamond_1$ licenses access to this package, so all moveable lexical material must be marked with the $\Diamond_1$ in the lexicon. If we think of a lexical formula like $(D \bullet K \bullet W)/N$ as denoting a function from N's to $(D \bullet K \bullet W)$ chains, and in general if we think of products in the lexicon as indicating that an item may have an effect on several different points of syntactic structure, then clearly we want the chain portion of this formula to be given access to the movement package: $\Diamond_1(D \bullet K \bullet W)/N$. But we have also to provide the lexical item with the means to stop moving. Since each feature can stop at a different point, we place a box operator before each individual feature: $\Diamond_1(\Box_1^{\downarrow} D \bullet \Box_1^{\downarrow} K \bullet \Box_1^{\downarrow} W)/N$. This would be enough if movement always placed features right next to the heads they cancel against. Because we must say more about what structural positions are actually considered close enough to count as being "logically adjacent", we will ultimately need our boxes to be the residuals of a different diamond, $\Diamond_0$, and we provide an inclusion rule which converts $\Diamond_1$ to $\Diamond_0$ when the derivation goes from a movement phase to a feature cancellation phase. So our final lexical formula for a Wh determiner like *which* will be: $\Diamond_1(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K \bullet \Box_0^{\downarrow} W)/N$.

The main rules of this package which actually execute movement transformations are the rules $Mv1$–3 given below.

$$\frac{\Gamma[(\langle\Delta_2\rangle^1, \Delta_1)^{\triangleright}] \vdash A}{\Gamma[(\Delta_1, \langle\Delta_2\rangle^1)^{<}] \vdash A}\, Mv1 \tag{24}$$

Rule $Mv1$ moves an entire complement expression, Δ_2, across a head Δ_1 to specifier position.

$$\frac{\Gamma[(\langle\Delta_2\rangle^1, (\Delta_1, \Delta_3)^{<})^{\triangleright}] \vdash A}{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^1, \Delta_3)^{\triangleright})^{<}] \vdash A}\, Mv2 \tag{25}$$

Rule $Mv2$ moves only the specifier of a complement to the (lowest) specifier position of a governing head.

$$\frac{\Gamma[(\langle\Delta_2\rangle^1, (\Delta_1, \Delta_3)^{\triangleright})^{\triangleright}] \vdash A}{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^1, \Delta_3)^{\triangleright})^{\triangleright}] \vdash A}\, Mv3 \tag{26}$$

Rule $Mv3$ moves a phrase from a lower specifier position to a higher. Unlike the other movement rules, which can implement what we intuitively think of as entire movement steps, $Mv3$ does not correspond to any entire movement step in a derivation. Rather, it is used because we have no other way of stating that a moving expression may move across any number of intervening specifiers (as long as locality or economy conditions are satisfied, see below). For example, there is no other way for a moving phrase to reach the highest of multiple specifier positions without $Mv3$, nor is there any way for a moving phrase to escape from the domain of a head with another specifier. Absence of this rule would prevent an object Wh phrase from moving over the subject to reach the specifier of CP position, for example.

However, presence of this rule does permit multiple specifiers to appear in virtually any order. Some control over this situation can be recovered, if necessary, by appeal to the feature cancellation system (see below on movement extending a structure). As we are about to see, the feature cancellation process cannot start until an expression gives up its license to move.[7]

$$\frac{\Gamma[(\langle\Delta_1\rangle^1, \langle\Delta_2\rangle^1)^i] \vdash A}{\Gamma[\langle(\Delta_1, \Delta_2)^i\rangle^1] \vdash A} K \tag{27}$$

$$\frac{\Gamma[\langle\Delta\rangle^0] \vdash A}{\Gamma[\langle\Delta\rangle^1] \vdash A} Mm \tag{28}$$

The final two rules in the package provide an exit from a movement subproof. The rule K is simply a rule of distributivity, which allows us in effect to peel a feature off of a larger complex of features. In particular, it allows us to peel off a feature which we intend to cancel in this position.[8] Then rule Mm, a containment rule, says in effect that anything that has access to movement rules has access to cancellation rules as well, converting a structure $\langle\Delta\rangle^1$ to a structure $\langle\Delta\rangle^0$.

Our system always restricts inference rules to act on very local configurations of resources. Long distance effects are obtained from the recursive structure of proofs, and by the corresponding iteration of a number of very local rules. As a result this system looks rather different in detail than a traditional transformational approach. For purposes of comparison, it is best not to think of the application of individual Mv rules as each implementing a movement step. Rather, an application of the generalized transformation **Move** corresponds to the entire subproof between the application of the first Mv rule to an expression to the application of the Mm rule, which switches the system from **Move** mode to feature cancellation mode.

Locality conditions may be placed as side conditions on these rules. For example, suppose we assume that (reading bottom up) a formula cannot cross another (c-commanding) formula with the same feature (with the same polarity). Nor can it move away from a head with an oppositely charged version of the same feature. Take for example $Mv2$, repeated below.

$$\frac{\Gamma[(\langle\Delta_2\rangle^1, (\Delta_1, \Delta_3)^{<})^{\triangleright}] \vdash A}{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^1, \Delta_3)^{\triangleright})^{<}] \vdash A} Mv2 \tag{29}$$

If $\Delta_2 = A$ and $\Delta_3 = A\backslash B$, for example, then the application of $Mv2$ should be blocked. In $Mv3$, if (the head of) Δ_3 contains an oppositely charged version of a feature found in Δ_2, or Δ_1 contains a similarly charged feature, then the

[7] Of course, placing a limit on the number of specifiers that can appear would allow us to write finitely many rules which would move expressions across sequences of specifiers. However, if that bound were greater than 1, we would run afoul of the "Syntax can't count" doctrine.

[8] The name of this rule is traditional, stemming from classical modal logic. It should not be confused with our use of K as an abbreviation for a Case feature.

application of the rule is blocked. In all these cases we would be violating Shortest Move, alias the Minimal Link Condition.

Also so far nothing has required that movement extend the structure, i.e., that all movement should be to the root. Here it is particularly important that we keep the notion of a **Move** step in a transformational derivation separate from the notion of the application of a Mv rule. Conditions on the target of **Move** should in general be translated as conditions on Mm, since this marks the end of a movement subproof. For example, if we rewrite Mm as a family of rules like the following

$$\frac{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^0, \Delta_3)^{\triangleright})^{\otimes}] \vdash A}{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^1, \Delta_3)^{\triangleright})^{\otimes}] \vdash A}\, Mm' \tag{30}$$

with variations for all the possible local configurations in which $\langle\Delta_2\rangle^1$ could appear, then we could prevent movement from halting prematurely. (Recall that $\otimes$ is the numeration forming connective, so this expression depicts the interface between a syntactic object and the collection of syntactic objects in the numeration.) Movement could still halt prematurely, but the feature(s) being moved could not ever cancel, since now Mm could not apply, and so no proof could ever be found that way.

There is one further feature of the **Move** package which deserves comment, but we postpone discussion until section 4: movement here leaves no traces. It actually destroys embedded structure in order to create new structure. The reason for this is straightforward. This is a calculus of syntactic features, and traces, being syntactically inert, have no syntactic (or rather, syntactically active) features. Nonetheless we have spoken frequently of tuples of features as being the types of chains, and we have not (knowingly) spoken this way irresponsibly. Again, the reader is referred to section 4, where we will discuss in some detail the formal relation between derivations and representations, and show how derivations as we have presented them here are mapped to "T-markers", in which traces, indices and chains all appear in their familiar roles.

2.2.4 Feature Checking Package ("Micro-Movement")

These rules act to break up a lexical product in one place, and form a new one in another. The c variants act to remove a feature from complement position, and the s variants act in the same way on specifiers. Note however that there can be multiple specifiers in Chomsky's system, but only a single complement. So the rules which (read bottom up) move a feature from the specifier to the head are somewhat more complex than those moving a feature from the complement to the head.[9]

[9]We have done nothing here to assure that only selectional requirements of a head can be satisfied in a complement configuration, that is, that the head-complement configuration is not a checking configuration for formal features. Probably the best way to do this would be to distinguish selectional and categorial features from formal and licensing features in the lexicon, by using either separate unary modalities or by splitting the $\bullet$ product in two. The increase in precision would be valuable, but we postpone its implementation for now.

Rules $F0c$ and $F0s$ handle the case where the feature to be cancelled is already adjacent to the feature it is to be cancelled against, so these take the form of embedding rules, but in a contextually restricted form.

$$\frac{\Gamma[(\Delta_1, \langle\Delta_2\rangle^0)^\bullet] \vdash A}{\Gamma[(\Delta_1, \langle\Delta_2\rangle^0)^<] \vdash A} F0c \qquad \frac{\Gamma[(\langle\Delta_1\rangle^0, \Delta_2)^\bullet] \vdash A}{\Gamma[(\langle\Delta_1\rangle^0, \Delta_2)^\triangleright] \vdash A} F0s \tag{31}$$

Rule $F1c$ acts in structurally similar circumstances to rule $F0c$; the difference is that in this case the complement is a complex of features, from which we detach one (Δ_2). Rule $F1s$ is similar: it detaches Δ_1 from a complex of features in specifier position but again, as in $F0s$, a specifier of a head with no complements. (Recall that our system can't see traces, so it is possible, after movement, to have a head with a specifier but no complement.) Note however that in this case the result is another specifier configuration, rather than a $(\cdot,\cdot)^\bullet$ configuration. This is because the head could have several specifiers (and still no complement), and our feature has to work its way across all of them to find the head (at which point $F0s$ applies).

$$\frac{\Gamma[((\Delta_1, \langle\Delta_2\rangle^0)^\bullet, \Delta_3)^<] \vdash A}{\Gamma[(\Delta_1, (\langle\Delta_2\rangle^0, \Delta_3)^\bullet)^<] \vdash A} F1c \quad \frac{\Gamma[(\Delta_2, (\langle\Delta_1\rangle^0, \Delta_3)^\triangleright)^\triangleright] \vdash A}{\Gamma[((\langle\Delta_1\rangle^0, \Delta_2)^\bullet, \Delta_3)^\triangleright] \vdash A} F1s \tag{32}$$

Rule $F2$ handles moving across multiple intervening specifiers, and rule $F3$ handles the cancelation of a feature originating in a specifier in the case where the head *does* have a complement.

$$\frac{\Gamma[(\Delta_2, (\langle\Delta_1\rangle^0, \Delta_3)^\triangleright)^\triangleright] \vdash A}{\Gamma[(\langle\Delta_1\rangle^0, (\Delta_2, \Delta_3)^\triangleright)^\triangleright] \vdash A} F2 \tag{33}$$

$$\frac{\Gamma[((\langle\Delta_1\rangle^0, \Delta_2)^\bullet, \Delta_3)^<] \vdash A}{\Gamma[(\langle\Delta_1\rangle^0, (\Delta_2, \Delta_3)^<)^\triangleright] \vdash A} F3 \tag{34}$$

3 (Minimalist) Derivation as Proof Search

In figure 2 we see a very simple example of some of this machinery at work. The derivation here is the trivial derivation which merges a D with a V that selects a D, canceling the categorial D feature against the selection feature of the V. In passing we note again that the minimalist program proper, as laid out in Chomsky (1995), does not include selection in the syntax. The conditions under which it is okay to merge a nominal complement with a predicate are off-loaded to the interpretive components. Here we treat both systems together, noting only that both phenomena associated with the lowest chain position (e.g., selection) and phenomena associated with higher chain positions can be accommodated in a single residuation calculus with a single primary product ($\bullet$), supported by a number of control modalities.

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{D \vdash D \quad V \vdash V}{(V/D, D)^{\bullet} \vdash V}\, L/}{(V/D, \langle \Box_0^{\downarrow} D\rangle^{0})^{\bullet} \vdash V}\, L\Box_0^{\downarrow}}{(V/D, \langle \Box_0^{\downarrow} D\rangle^{0})^{<} \vdash V}\, F0c}{(V/D,\ \langle \Box_0^{\downarrow} D\rangle^{1})^{<} \vdash V}\, Mm}{(V/D,\ \Diamond_1 \Box_0^{\downarrow} D)^{<} \vdash V}\, L\Diamond_1}{V/D, \Diamond_1 \Box_0^{\downarrow} D \vdash V}\, Mg1
$$

Figure 2: An elementary derivation, cancelling one selection feature.

In the proof in figure 2, read bottom up as a proof-search, we could have equally well started with rule $L\Diamond_1$ and then afterwards used $Mg1$. The real action begins with the application of rule Mm. We decide that we have established a checking configuration (the head-complement configuration is certainly a checking configuration for selection features) and so we convert $\langle \Box_0^{\downarrow} D\rangle^1$ to $\langle \Box_0^{\downarrow} D\rangle^0$.

At this point we have a formula to which we could apply $L\Box_0^{\downarrow}$. However, if we did so we would be left with a configuration of resources $(V/D, D)^{<}$. Keep in mind that all of our slashes are shorthand for $/_{\bullet}$, that is, they are the residuals of $\bullet$, not of $<$. So this configuration does not allow the application of $L/$. We can, however, apply $F0c$, which tells us that a single feature in a complement configuration can be considered adjacent to a functor for purposes of feature cancellation. Now that we have applied $F0c$, it is safe to remove the protective wrapping from the D feature, leaving us with the configuration $(V/D, D)^{\bullet}$, as desired. From there the proof is by the application of $L/$.

The dependencies between the various packages can be seen as follows. The **Merge** package gives us access to all of the Move package by converting numeration configurations into specifier or complement configurations; then the **Move** package—which applies trivially in figure 2—gives access to the Feature Cancellation package, which converts specifier and complement configurations into function application configurations and so finally gives access to the base logic of residuation.

In figure 3 we see a slightly more complex derivation, which includes a simple but not quite trivial instance of **Move**. Here we have a V which selects a D complement and assigns case to its specifier, together with a nominal with a categorial D feature and a K formal feature both of which must be canceled. The type formula $\Diamond_1(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K)$ is the type of a simple nominal A-chain.

The beginning of the proof search is similar to the one in figure 2. The main difference is that now our complement contains two features, so we have to do some extra work to peel off the D feature. This accounts for the two

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{D \vdash D \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{K \vdash K \quad V \vdash V}{(K,\ K\backslash V)^{\bullet} \vdash V}\,L\backslash}{(\langle \Box_0^{\downarrow} K\rangle^{\Diamond_0},\ K\backslash V)^{\bullet} \vdash V}\,L\Box_0^{\downarrow}}{(\langle \Box_0^{\downarrow} K\rangle^{0},\ K\backslash V)^{\rhd} \vdash V}\,F0s}{(\langle \Box_0^{\downarrow} K\rangle^{1},\ K\backslash V)^{\rhd} \vdash V}\,Mm}{(K\backslash V,\ \langle \Box_0^{\downarrow} K\rangle^{1})^{<} \vdash V}\,Mv1}{(((K\backslash V)/D,\ D)^{\bullet},\ \langle \Box_0^{\downarrow} K\rangle^{1})^{<} \vdash V}\,L/}{(((K\backslash V)/D,\ \langle \Box_0^{\downarrow} D\rangle^{0})^{\bullet},\ \langle \Box_0^{\downarrow} K\rangle^{1})^{<} \vdash V}\,L\Box_0^{\downarrow}}{((K\backslash V)/D,\ (\langle \Box_0^{\downarrow} D\rangle^{0},\ \langle \Box_0^{\downarrow} K\rangle^{1})^{\bullet})^{<} \vdash V}\,F1c}{((K\backslash V)/D,\ (\langle \Box_0^{\downarrow} D\rangle^{1},\ \langle \Box_0^{\downarrow} K\rangle^{1})^{\bullet})^{<} \vdash V}\,Mm}{((K\backslash V)/D,\ \langle(\Box_0^{\downarrow} D,\ \Box_0^{\downarrow} K)^{\bullet}\rangle^{1})^{<} \vdash V}\,K}{\dfrac{((K\backslash V)/D,\ \langle(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K)\rangle^{1})^{<} \vdash V}{\dfrac{((K\backslash V)/D,\ \Diamond_1(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K))^{<} \vdash V}{(K\backslash V)/D, \Diamond_1(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K) \vdash V}\,Mg1}\,L\Diamond_1}\,L\bullet
$$

Figure 3: A slightly more complex derivation, canceling one selection feature and one formal feature.

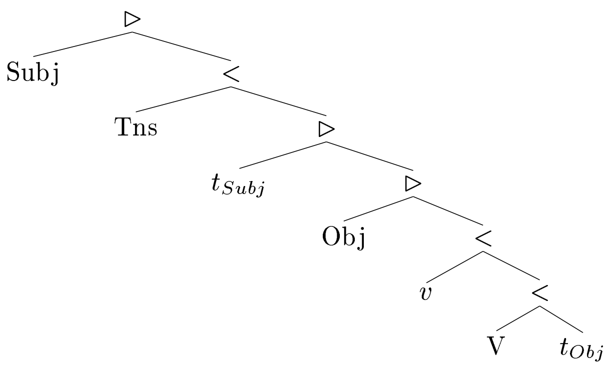
new steps—the applications of rules $L\bullet$ and K—that now intervene between the $L\Diamond_1$ and Mm steps. Also we now have to use $F1c$ rather than $F0c$ to feed the $L/$ rule that cancels the D feature.

We are then left with a verb which assigns case to its left ($K\backslash V$), and a case-needing chain element ($\langle \Box_0^{\downarrow} K \rangle^1$) to its right. All we really need to do here is simply to permute the functor and its argument, but since we cannot allow the permutation rule P to apply indiscriminately we first have to check that the potential argument formula really is allowed to permute. This is indeed the case: the $\langle\cdot\rangle^1$ wrapper fits the requirements of the $Mv1$ rule on its output. The rest of the proof is similar to the cancellation of D feature, except that we move the K feature in from the left instead of from the right.

3.1 A More Elaborate Derivation

So far we have considered derivations in which we only ever have one instance of any given feature. The hard problems arise when we have more than one. For example, suppose we have two nominal expressions, both with type $\Diamond_1(\Box_0^{\downarrow} D \bullet \Box_0^{\downarrow} K)$. It is in principle always possible to use our proof machinery to break up such feature clusters into separately movable pieces, and there is always a danger that the pieces we free up in this way may attach to the wrong places (cf. Cornell 1997). A classic example would be the case where the external argument robs the accusative case of an active voice verb from the internal argument, which then has to raise to subject position to get case, creating in effect a passive structure based on an active voice verb.

In Chomsky (1995), the following structure is considered for ordinary transitive clauses (disregarding head movement, and assuming overt object raising), along with a variant in which the overt object outscopes the subject trace.



Assuming that accusative case is assigned by the "light verb" v, we need

$$
\begin{prooftree}
\AxiomC{$D \vdash D$}
\AxiomC{$V \vdash V$}
\AxiomC{$K \vdash K$}
\AxiomC{$D \vdash D$}
\AxiomC{$V \vdash V$}
\AxiomC{$K \vdash K$}
\AxiomC{$T \vdash T$}
\RightLabel{$L\Diamond_1, Mm, L\Box_0^{\downarrow}$}
\UnaryInfC{$\Diamond_1\Box_0^{\downarrow}T \vdash T$}
\RightLabel{$F0s, L\Box_0^{\downarrow}, L\backslash$}
\BinaryInfC{$(\langle\Box_0^{\downarrow}K\rangle^0,\ K\backslash\Diamond_1\Box_0^{\downarrow}T)^{\triangleright} \vdash T$}
\RightLabel{$L\Diamond_1, Mm, F0c, L\Box_0^{\downarrow}, L/$}
\BinaryInfC{$(\langle\Box_0^{\downarrow}K\rangle^0,\ ((K\backslash\Diamond_1\Box_0^{\downarrow}T)/V,\ \Diamond_1\Box_0^{\downarrow}V)^{<})^{\triangleright} \vdash T$}
\RightLabel{$Mv2, Mm$}
\UnaryInfC{$((K\backslash\Diamond_1\Box_0^{\downarrow}T)/V,\ (\langle\Box_0^{\downarrow}K\rangle^1,\ \Diamond_1\Box_0^{\downarrow}V)^{\triangleright})^{<} \vdash T$}
\RightLabel{$Mg1$}
\UnaryInfC{$\mathrm{Tns}, (\langle\Box_0^{\downarrow}K\rangle^1,\ \Diamond_1\Box_0^{\downarrow}V)^{\triangleright} \vdash T$}
\RightLabel{$L\Diamond_1, L\bullet, K, Mm, F1s, F0s, L\Box_0^{\downarrow}, L\backslash$}
\BinaryInfC{$\mathrm{Tns}, (\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K),\ D\backslash\Diamond_1\Box_0^{\downarrow}V)^{\triangleright} \vdash T$}
\RightLabel{$Mg2$}
\UnaryInfC{$\mathrm{Tns}, \mathrm{Subj}, D\backslash\Diamond_1\Box_0^{\downarrow}V \vdash T$}
\RightLabel{$F0s, L\Box_0^{\downarrow}, L\backslash$}
\BinaryInfC{$\mathrm{Tns}, \mathrm{Subj}, (\langle\Box_0^{\downarrow}K\rangle^0,\ K\backslash(D\backslash\Diamond_1\Box_0^{\downarrow}V))^{\triangleright} \vdash T$}
\RightLabel{$L\Diamond_1, Mm, F0c, L\Box_0^{\downarrow}, L/$}
\BinaryInfC{$\mathrm{Tns}, \mathrm{Subj}, (\langle\Box_0^{\downarrow}K\rangle^0,\ ((K\backslash(D\backslash\Diamond_1\Box_0^{\downarrow}V))/V,\ \Diamond_1\Box_0^{\downarrow}V)^{<})^{\triangleright} \vdash T$}
\RightLabel{$Mv1, Mv2, Mm$}
\UnaryInfC{$\mathrm{Tns}, \mathrm{Subj}, ((K\backslash(D\backslash\Diamond_1\Box_0^{\downarrow}V))/V,\ (\Diamond_1\Box_0^{\downarrow}V,\ \langle\Box_0^{\downarrow}K\rangle^1)^{<})^{<} \vdash T$}
\RightLabel{$Mg1$}
\UnaryInfC{$\mathrm{Tns}, \mathrm{Subj}, v, (\Diamond_1\Box_0^{\downarrow}V,\ \langle\Box_0^{\downarrow}K\rangle^1)^{<} \vdash T$}
\RightLabel{$L\Diamond_1, L\bullet, K, Mm, F1c, L\Box_0^{\downarrow}, L/$}
\BinaryInfC{$\mathrm{Tns}, \mathrm{Subj}, v, (\Diamond_1\Box_0^{\downarrow}V/D,\ \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K))^{<} \vdash T$}
\RightLabel{$Mg1$}
\UnaryInfC{$\mathrm{Tns}, \mathrm{Subj}, v, \mathrm{V}, \mathrm{Obj} \vdash T$}
\end{prooftree}
$$

Figure 4: An abbreviated derivation with two DPs.

something like the following lexical assignments.

$$\mathrm{Tns} \mapsto (K\backslash T)/V \tag{35}$$

$$v \mapsto (K\backslash(D\backslash\Diamond_1\Box_0^{\downarrow}V))/V \tag{36}$$

$$\mathrm{V} \mapsto \Diamond_1\Box_0^{\downarrow}V/D \tag{37}$$

$$\mathrm{Subj},\ \mathrm{Obj} \mapsto \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K) \tag{38}$$

The following is a theorem in our system. (The subscripts $Subj$ and Obj have no status in our system. They are just here for readability.)

$$(K\backslash T)/V, \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K)_{Subj}, (K\backslash(D\backslash(\Diamond_1\Box_0^{\downarrow}V)))/V, \\ (\Diamond_1\Box_0^{\downarrow}V)/D, \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K)_{Obj} \vdash T \tag{39}$$

A proof is given in figure 4. Note that we collapse entire sequences of rule applications where these do things we have already seen examples of. We employ here an abbreviatory convention according to which we use the lexical item in place of its type formula until the type formula is actually needed.

Observe that the first feature to be cancelled must be the D feature of one of the two DPs. Choosing the one marked "Obj", that leaves us with the following goal sequent.

$$\mathrm{Tns}, \mathrm{Subj}, v, (\Diamond_1\Box_0^{\downarrow}V, \langle\Box_0^{\downarrow}K\rangle^1)^{<} \vdash T \tag{40}$$

Note that we could at this point (after some permutations and reassociations) merge the subject formula with the tree we have built so far. That would put it closer to the light verb's K feature, blocking movement of the object's K feature. But the subject cannot cancel the the verb's K feature itself: it has to discharge its D feature in order to "release" its K feature. So that move is blocked by the nonassociative and noncommutative nature of the $\bullet$ product. Instead we merge the current tree with the light verb.

$$\mathrm{Tns}, \mathrm{Subj}, ((K\backslash(D\backslash(\Diamond_1\Box_0^{\downarrow}V)))/V,\ (\Diamond_1\Box_0^{\downarrow}V,\ \langle\Box_0^{\downarrow}K\rangle^1)^{<})^{<} \vdash T \tag{41}$$

Now we can move the object K feature out of the lower VP.

$$\begin{aligned}&(K\backslash T)/V, \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K)_{Subj},\\ &\qquad (\langle\Box_0^{\downarrow}K\rangle^1,\ ((K\backslash(D\backslash(\Diamond_1\Box_0^{\downarrow}V)))/V,\ \langle\Box_0^{\downarrow}V\rangle^1)^{<})^{\triangleright} \vdash T\end{aligned} \tag{42}$$

Now we can cancel the V feature. Note that, without a Head Movement package, we could not do this until we had evacuated all uncancelled features from around the head. This trick works, but note that it runs counter to proposals in Chomsky (1995), according to which the head should move first. The rest of the derivation offers nothing new; relevant snapshots of the main goal sequent are given below.

$$\mathrm{Tns}, \mathrm{Subj}, (\langle\Box_0^{\downarrow}K\rangle^1,\ K\backslash(D\backslash(\Diamond_1\Box_0^{\downarrow}V)))^{\triangleright} \vdash T \tag{43}$$

$$\mathrm{Tns}, (\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}K)_{Subj},\ D\backslash(\Diamond_1\Box_0^{\downarrow}V))^{\triangleright} \vdash T \tag{44}$$

$$\mathrm{Tns}, (\langle\Box_0^{\downarrow}K\rangle^1,\ \langle\Box_0^{\downarrow}V\rangle^1)^{\triangleright} \vdash T \tag{45}$$

$$(\langle\Box_0^{\downarrow}K\rangle^1,\ ((K\backslash T)/V,\ \langle\Box_0^{\downarrow}V\rangle^1)^{<})^{\triangleright} \vdash T \tag{46}$$

Step (43) is the result of cancelling the lower V feature. Step (44) is the resultof cancelling the object's K feature against the higher verb. Then the subject's D feature is cancelled against the verb leading to (45). Finally, after some applications of rules from the **Merge** package, the subject's K feature can move to specifier of T, leaving the verb free to cancel its V feature against the V feature of T, which will then allow the cancellation of the subject's K feature, finishing the proof.

4 Derivation and Representation

In this section we consider the structure of proofs in our system, using techniques of labeled deduction (Gabbay 1994) generalizing the well-known Curry-Howard correspondence between proofs and terms of the lambda calculus. A generative linguist can see in these techniques a generalization of the familiar idea that a phrase structure tree for a sentence can be extracted from a phrase structure derivation serving at once as a record of that derivation and as a static analysis of the sentence so derived. Here we will use the record of a transformational derivation to provide a static analysis of the sentence so derived. Since the derivation here is a transformational one, and not a phrase structure derivation, we can think of the term representing the essential structure of the derivation as a T-marker, to revive some terminology from Chomsky (1975/1955).

In the structural rules which implement the **Move** transformation we saw a version of movement which destroys the local structure in which the moving expression is initially embedded. So, to take a very simple example, movement from complement to specifier of a single head takes a configuration like $(\Gamma,\ \Delta)^{<}$ as input and produces a configuration like $(\Delta,\ \Gamma)^{\triangleright}$ as output, crucially without leaving a trace behind. The reason for this is that this is a calculus of syntactically active features. Syntactic structure which is inactive is invisible to this system. In particular, traces—which bear no features that might still need to be checked—are invisible to the derivational machinery. However, if we examine the structure of the derivations themselves, rather than the transient structures that they create, we can recover our familiar traces, chains and indices and study them. In a sense, we can use the Curry-Howard morphism to connect derivations with the flavor of the 60's era Standard Theory of transformational grammar to representations with more of the flavor of 80's era Government-Binding Theory and the minimalist program.

The assignment of proof terms to formulas works as follows in the setting of a Gentzen sequent calculus. With each basic type appearing in an axiom we associate a variable, so that the axiom $A \vdash A$ is now written $x:A \vdash x:A$. Then the inference rules tell us how to manipulate these variables so as to record the essential structure of the proof. Note in particular the following property of our new labeled rules, which is an invariant on the recursive structure of proofs: the formulas of sequent antecedents are always labeled with variables (which may be fresh variables introduced by the inference rule). The consequence, on the other hand, gets increasingly complex at each step. When we are done, we have a sequent with a numeration of lexical type formulas in the antecedent—each labeled with a variable—and the sentence category in the consequence, labeled with a lambda term built out of those variables.

This device is familiar from formal semantics, where we can use it to derive the semantics of a sentence by assigning to the variables in the goal sequent terms drawn from the lexicon giving the semantics of individual words. The proof term assigned to the consequence of the goal sequent then provides a recipe or program for constructing the semantics of the sentence from the semantics of its constituent lexical items. One can, however, just as easily assign to these variables terms which reflect the theory of syntax in such a way that the lambda term will be converted to an LF level syntactic structure. On the surface this may not seem like a very interesting thing to do: the lambda term itself is already a tree structure with obvious and transparent relations to the kinds of trees one is familiar with from the syntax literature. However, we will see that this gives us a nice, rigorous mathematical framework for investigating questions like the copy theory of movement, spell out, and reconstruction, that is, questions proper to the theory of chains. We derive in this way a clean break between the theory of chains on the one hand and the theory of movement on the other, which we take to have to do with locality or economy conditions restricting

$$A : x \vdash A : x \quad \textit{Axiom} \qquad \frac{\Gamma \vdash t{:}A \quad \Delta[x{:}A] \vdash u{:}B}{\Delta[\Gamma] \vdash u[t/x]{:}B}\,Cut \tag{47}$$

$$\frac{\Gamma \vdash t{:}A \quad \Delta[x{:}B] \vdash u{:}C}{\Delta[(y{:}B/_iA,\ \Gamma)^i] \vdash u[y(t)/x]{:}C}\,L/_i \qquad \frac{(\Delta,\ x{:}A)^i \vdash u{:}B}{\Delta \vdash \lambda x.u{:}B/_iA}\,R/_i \tag{48}$$

$$\frac{\Gamma \vdash t{:}A \quad \Delta[x{:}B] \vdash u{:}C}{\Delta[(\Gamma,\ y{:}A\backslash_i B)^i] \vdash u[y(t)/x]{:}C}\,L\backslash_i \qquad \frac{(x{:}A,\ \Delta)^i \vdash u{:}B}{\Delta \vdash \lambda x.u{:}A\backslash_i B}\,R\backslash_i \tag{49}$$

$$\frac{\Delta[(x_1{:}A,\ x_2{:}B)^i] \vdash u{:}C}{\Delta[z{:}A \bullet_i B] \vdash u[(z)_1/x_1, (z)_2/x_2]{:}C}\,L\bullet_i \qquad \frac{\Delta_1 \vdash x_1{:}A \quad \Delta_2 \vdash x_2{:}B}{(\Delta_1,\ \Delta_2)^i \vdash \langle x_1, x_2\rangle{:}A \bullet_i B}\,R\bullet_i \tag{50}$$

$$\frac{\Gamma[x{:}A] \vdash u{:}B}{\Gamma[\langle y{:}\Box_i^{\downarrow}A\rangle^i] \vdash u[{}^{\vee}y/x]{:}B}\,L\Box_i^{\downarrow} \qquad \frac{\langle\Gamma\rangle^i \vdash t{:}A}{\Gamma \vdash {}^{\wedge}t{:}\Box_i^{\downarrow}A}\,R\Box_i^{\downarrow} \tag{51}$$

$$\frac{\Gamma[\langle x{:}A\rangle^i] \vdash u{:}B}{\Gamma[y{:}\Diamond_i A] \vdash u[{}^{\cup}y/x]{:}B}\,L\Diamond_i \qquad \frac{\Gamma \vdash t{:}A}{\langle\Gamma\rangle^i \vdash {}^{\cap}t{:}\Diamond_i A}\,R\Diamond_i \tag{52}$$

Figure 5: Logical inference rules, annotated with proof term labels.

derivations. It is in many ways easier to deal with chains representationally and locality derivationally, so it is nice to have a single system in which these two approaches are not merely compatible but actually two sides of the same coin.

The logical rules of figure 1 are given in their labeled forms in figure 5. In figure 5 we use the following notational conventions. Given a term u, the notation $u[t/x]$ denotes the substitution of t for x, x a leaf node in the term u. Given a term z, which should represent an ordered pair, $(z)_1$ and $(z)_2$ represent the first and second projections of that pair, i.e., its first and second coordinates, respectively. We use x, y and z to represent variables, that is, terms assigned to type formulas in sequent antecedents, and we use s, t and u to represent proper terms, that is, terms associated with type formulas in sequent consequences. Note that proof terms are associated with formulas, not with configurations, so that we might have, for instance, $x_1{:}\Diamond A$ but $\langle x_2{:}A\rangle$.

The proof terms for the binary products are based on the familiar Curry-Howard correspondence between proofs and lambda terms: the left slash rules correspond to function application, the right slash rules to lambda abstraction, the left product rules to projection and the right product rules to pairing. We observe the following formal generalization: the right rules implement what we might think of abstractly as *constructor* functions (which construct function types and product types in these cases), while the left rules implement what we might think of as the corresponding *destructor* rules. This pattern is carried over to the unary modalities, where the $^\cap$ and $^\wedge$ functions stand for constructors for the diamonds and their residuals, respectively, and the $^\cup$ and $^\vee$ functions stand for corresponding destructors. Unlike the case for binary modes of combination there is no traditional concrete instantiation of this notion of constructor and destructor. They can be anything, as long as destructors and constructors cancel each other out. Because the unary modals play such a large role in the theory of movement, we will consider below how these abstract functions might be realized in the context of the theory of chains.

As usual in a Gentzen calculus, the left slash rules are probably the most difficult to understand. Take for example the $L/_i$ rule.

$$\frac{\Gamma \vdash t\!:\!A \quad \Delta[x\!:\!B] \vdash u\!:\!C}{\Delta[(y\!:\!B/_iA,\ \Gamma)^i] \vdash u[y(t)/x]\!:\!C}\ L/_i$$

The key idea is that constructing a C from $\Delta[B]$ will insert an occurrence of B's associated variable (here x) somewhere in the term associated with C (here u). That is, we know that u contains an object x of type B. We also know at least one way to construct new objects of type B, namely by applying a function of type B/A to an object of type A. So if we have an object y of type B/A and an object t of type A (or rather, the resources Γ that we know are sufficient to build such a thing), then $y(t)$ must be an object of type B. Therefore we can substitute it for x in u, which is what we do in the conclusion of the $L/_i$ rule.

The way in which proofs and proof terms are related is relatively fixed. It is at the level of the lexical term assignments that we can examine the effects of different theoretical proposals. We have already observed that lexical type formulas represent "in miniature" the phrases and chains into which the syntax will extend them. In general, lexical types represent functions from phrases to chains. Consider again the assignment to the Wh determiner *which*, ignoring the control modals for the moment:

$$\textit{which} \mapsto (D \bullet K \bullet W)/N \tag{53}$$

Any lexical term assignment which we might consider must make sense as an object of this type. That is, it should take a single argument and return a triple of objects.[10] Observe that we can compute a general form for such a term by labeling the proof of the "complex axiom"

$$(D \bullet K \bullet W)/N \vdash (D \bullet K \bullet W)/N.$$

This yields the following labeled sequent as a theorem of our system (the proof

[10] Actually the situation is slightly more complicated because of the presence of the control modals. We will consider their added effects below.

is left as a simple and useful exercise).[11]

$$y:(D \bullet (K \bullet W))/N \vdash \lambda x.\langle (y(x))_1, (y(x))_2, (y(x))_3\rangle : (D \bullet (K \bullet W))/N \qquad (54)$$

Accordingly we derive the equation

$$y = \lambda x.\langle (y(x))_1, (y(x))_2, (y(x))_3\rangle.$$

This equation is satisfied by any function which maps a single argument to a triple, and so gives a general form for such functions.

One possible term assignment would then be something like the following, where α represents the lexical determiner.

$$(D \bullet K \bullet W)/N : \lambda y.\langle (\alpha < y), (\alpha < y), (\alpha < y)\rangle \qquad (55)$$

(We use Stabler's notation here rather than Chomsky's; to recover that notation, substitute $\{\alpha, \{\alpha, x\}\}$ for $(\alpha < x)$.) Here we are treating the application of α to x as literal head-complement formation. Such a term implements the Copy Theory of movement or chain formation. Cancelling the N feature with the $L/$ rule will have the effect of applying this function to the proof term of the NP, making 3 copies of it. Then as each of the DP's own features are cancelled, one copy will be peeled off of the tuple and attached into the phrase of the cancelling head.

Note that this offers a solution to a problem of Brody's (Brody 1997). The theory presented there, like the theory underlying Brody (1995a), combines the idea of presyntactic chain formation with the copy theory of movement. A problem for such a theory is that, presyntactically, it is not yet known what material will be attached into the X-bar projection of the head. Accordingly Brody must place well-formedness conditions on his representations which assure that all information adhering to one copy is propagated to all the others. These conditions are rather complex and intricate. Here we see that nothing extra is really required. The feature makeup of the Wh determiner determines its type: features which it can cancel appear under the slash and features which need to be cancelled appear over the slash. The elements of the chain appearing over the slash can really be anything: the stipulation that they be identical is a substantive condition which forms the core assumption of the copy theory of movement. Given that assumption, and stating things with reference to the proof term in (54), we know that $(y(x))_1$ equals $(y(x))_2$ equals $(y(x))_3$. All that is left is to decide what structure should be built when a determiner combines with its (internal) argument. Again, the stipulation that that should be a head-complement configuration seems entirely unproblematic. So we are driven to a representation which, when combined with an argument, will propagate the structure of that argument to all chain positions without further stipulation. In particular, the fact that the λx takes scope over three copies of x simply falls out of the structure of the formula and the way it is used in proofs.

[11] Note that this proof is not, technically, a derivation, since it is not a proof of a basic category from a numeration. We see again the utility of being able to do proofs beyond just derivations.

It is interesting that, as far as proof terms are concerned, it is the $L/$ and $L\backslash$ rules which implement **Merge**. That is, these are the rules which implement function application, and therefore which will trigger the application of structure building functions to their arguments. Indeed, the proportion "**Merge** is to Application as **Move** is to Abstraction" is another reason why the relationship of the minimalist program to categorial grammar has been so compelling. However, this means that it is really **Merge** which is responsible for feature cancellation, whereas in Chomsky (1995) feature cancellation is supposed to be an essential part of **Move**. However, this connection between **Merge** and feature cancellation, or something very much like it, has already been proposed in other minimalist literature (Collins 1997, Bobaljik 1995). In particular, it has been proposed that **Move** should be treated as removing uncancelled features from the tree and returning them to the numeration, whence they are later merged again at a higher point, or that **Move** at very least includes **Merge** as a suboperation. On such a view it will always be **Merge** which (a) builds structure and (b) in so doing places features in checking configurations.

This relationship of our calculational machinery and the theoretical literature is especially interesting inasmuch as Brody espouses a representational, chain-based perspective on extraction while Collins, Bobaljik and company embrace a derivational, movement-based perspective. Normally, these two perspectives are taken to be competing theories, but in the light of a deductive approach they appear as two sides of the same coin.

A more traditional view of chains would be obtained with an assignment like the following.

$$(D \bullet K \bullet W)/N : \lambda x.\langle t_\alpha, t'_\alpha, \alpha < x\rangle \tag{56}$$

In this version we have committed to interpreting the DP in its highest position, though without saying whether we mean this to represent its LF interpretation, its PF interpretation, or both. We might consider how we would go about representing a DP chain in which distinct positions were interpreted at PF and LF, respectively. One way to do this would be to assign to each lexical item a pair of proof terms. This would implement in a minimalist setting a research program like Oehrle's (Oehrle 1994, , and refs.). An alternative, inspired by Stabler's proposals for "one-pass" derivations (Stabler forthcoming, 1997a), would be to use the same proof term but with subexpressions like $[\![\alpha]\!]$ and $/\alpha/$, for the LF and PF interpretations of α respectively, and similarly for $[\![x]\!]$ and $/x/$.

One would like to be able to explore positions intermediate between the copy theory and more traditional theories about chains. One possibility is suggested by Collins (1997): starting with a copying operation, we then apply partial deletion to the copies ("up to recoverability"). In effect, we just have to be sure that, knitting all the copies back together again, we obtain the features of the original lexical item. For studying these rather more complex theories it becomes important to look again at our control modals and their effects on the proof.

In figure 6 we exhibit the simple derivation from figure 3, this time with labels. The essential point is the proof term we derive in the consequent of the

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{x_4 : D \vdash x_4 : D \quad \cfrac{\cfrac{\cfrac{x_1 : K \vdash x_1 : K \quad x_2 : V \vdash x_2 : V}{(x_1 : K,\ f' : (K\backslash V))^\bullet \vdash f'(x_1) : V}\, L\backslash}{(\langle x_3 : \Box_0^\downarrow K\rangle^0,\ f' : K\backslash V)^\bullet \vdash f'({}^\vee x_3) : V}\, L\Box_0^\downarrow}{f' : (K\backslash V), \langle x_3 : \Box_0^\downarrow K\rangle^1 \vdash f'({}^\vee x_3) : V}\, \Psi}{(f : (K\backslash V)/D,\ x_4 : D)^\bullet, \langle x_3 : \Box_0^\downarrow K\rangle^1 \vdash f(x_4)({}^\vee x_3) : V}\, L/}{(f : (K\backslash V)/D,\ \langle x_5 : \Box_0^\downarrow D\rangle^0)^\bullet, \langle x_3 : \Box_0^\downarrow K\rangle^1 \vdash f({}^\vee x_5)({}^\vee x_3) : V}\, L\Box_0^\downarrow}{f : (K\backslash V)/D, \langle (x_5 : \Box_0^\downarrow D,\ x_3 : \Box_0^\downarrow K)^\bullet\rangle^1 \vdash f({}^\vee x_5)({}^\vee x_3) : V}\, \Phi}{f : (K\backslash V)/D, \langle x' : \Box_0^\downarrow D \bullet \Box_0^\downarrow K\rangle^1 \vdash f({}^\vee(x')_1)({}^\vee(x')_2) : V}\, L\bullet}{f : (K\backslash V)/D, x : \Diamond_1(\Box_0^\downarrow D \bullet \Box_0^\downarrow K) \vdash f({}^\vee({}^\cup x)_1)({}^\vee({}^\cup x)_2) : V}\, L\Diamond_1
$$

Figure 6: The derivation of figure 3 with labels.

goal sequent, repeated below.

$$f({}^\vee({}^\cup x)_1)({}^\vee({}^\cup x)_2)$$

In particular, notice that the ${}^\cup$ function, which represents the destructor for the $\Diamond_1$ modal, which in turn marks chains as moveable, appears under the scope of the projection functions which break up the chain for attachment in multiple locations. We have $({}^\cup x)_1$ and $({}^\cup x)_2$. The variable x corresponds to the proof term for the argument DP, in this case a two element chain. If we think of ${}^\cup x$ as representing the application of a partial deletion function to a chain, then we can read full proof term for the VP in the following way. We calculate the effects of partial deletion on the chain x, then peel off the root of that chain and apply f to it, making it the verb's structural complement in the T-marker. We then apply the resulting function to the head of the chain, or what is left of the head after partial deletion, yielding a specifier construction. Note that we must be sure that ${}^\cup$ cancels with ${}^\cap$. One way to do that might be to implement the constructor ${}^\cap$ as the (bidirectional) propagation of information along a chain. In this way we force the condition of recoverability on partial deletion: if too much is deleted then propagation will not recover the original chain.

In this section we have seen how chains arise out of the primitive assumption that features are resources which are consumed by a derivation. The resource conscious approach to formal features characteristic of the minimalist program forces us to admit lexical type formulas containing tuples of features. As soon as we try to say what kinds of syntactic objects these tuples may be the types of, we arrive at chains. Finally, the mechanisms of modal control over movement provide us at the same time with devices which can be used in the theory of chain formation to regulate the distribution of syntactic information among the various chain positions. In this way we derive the mathematical foundations of a syntactic theory which uses both movement and chains, as the theory of Chomsky (1995) does, while at the same time keeping them separate and synchronized. Brody (1995b) is critical of the minimalist program as it was developing at the time precisely because it mixed chains and movement transformations, which Brody argued were redundant. We claim here, on the strength of these investigations, that the problems with such redundancies can be tamed if there is a precise correspondence between chains and movement so that they are in fact, as they are here, two aspects of the same mathematical object. Chains may be seen as the static reflex of the way in which features are cancelled in a derivation, or derivations may be seen as the way in which conditions on the interaction of chains and chain elements are verified. As these perspectives are equivalent, neither can really be more correct than the other, and both become more useful as it becomes possible to use whichever one is better suited to the purpose at hand.

5 On the Relation Between Movement and Hypothetical Reasoning

In figure 7 we see an example of a classical derivation of a relative clause in the associative Lambek calculus **L**. We see that the relative pronoun *which* is assigned in the lexicon the type of a function from 1-place predicates needing an argument on their right (S/NP) to whatever type (complete) relative clauses should have. The bulk of the proof involves proving that the resources of the clause other than the relative pronoun are sufficient to construct an instance of its argument type. Since that argument type is higher order (it is the type of a function and not just a basic type) the main subproof is a proof of a formula on the right of the $\vdash$ whose main functor is $/$, so we require the use of the $R/$ rule to deal with this. The $R/$ rule is the Gentzen version of "hypothetical reasoning": we prove that we can construct a functional type by proving that, if we had its argument type on the *left* of the $\vdash$, then we could construct the type of its output. In type-logical terms, then, the left residual rules correspond to the application of a function to its argument and the right residual rules tell us how to construct a function by abstraction over its argument. (In terms of more traditional implicational logic, the left residual rules correspond to modus ponens while the right rules correspond to the deduction theorem.)

It is the introduction of hypothetical reasoning into categorial grammar which distinguishes the systems derived from Lambek's calculi **L** and **NL** from the Ajdukiewicz-Bar Hillel system (**AB**), which has only function application without abstraction. It is this move which really turns a rule based grammar formalism into a logic, and so the utility of hypothetical reasoning for handling linguistic extraction phenomena has been an important argument over the years for this approach to categorial grammar.

However, in our minimalist-style derivations we have made no use of hypo-

$$\dfrac{\dfrac{\dfrac{\dfrac{NP \vdash NP \qquad \dfrac{NP \vdash NP \quad S \vdash S}{NP, NP\backslash S \vdash S}\,L\backslash}{NP, (NP\backslash S)/NP, NP \vdash S}\,L/}{NP, (NP\backslash S)/NP \vdash S/NP}\,R/ \qquad R \vdash R}{R/(S/NP), NP, (NP\backslash S)/NP \vdash R}\,L/}{\text{which mary proved} \vdash R}\,\mathit{Lookup}$$

Figure 7: which Mary proved $\vdash R$ in **L**.

thetical reasoning. We have used only left rules.[12] This is because our grammar makes no use of higher order types: there are no types in minimalist lexicons which take functions as arguments.[13] So it is important to examine at this point in more detail just how we manage to derive extraction phenomena without appeal to hypothetical reasoning. We will see immediately that certain features of transformational grammar are rather surprisingly not relevant to distinguishing it from categorial grammar. For example, the appeal to empty trace categories is irrelevant: we have seen already that these have no place in the derivational calculus in any case. The appeal to movement is also irrelevant: in order to deal with a host of real syntactic phenomena, hypothetical reasoning in categorial derivations must also be augmented with modally controlled access to permutation and association, though typically the direction of movement is the reverse of our system. Perhaps most surprisingly, the appeal to hypothetical reasoning itself is not particularly significant. We will see that this is essentially an artifact of the hierarchical order in which we wish to combine resources.

In figures 9 and 10 we present sketches of the labeled derivations of a subject-relative clause like ... *who sneezed* ... in CG and TG. The CG type for an English-style relative clause should be something like $N\backslash N$, the type of a nominal post modifier, hower we write it here simply as R. Variables have been given suggestive names to help in comparing the actions of the two different derivation systems. The main thing to observe is the way in which the TG derivation uses the resource W (for the Wh feature) and the null Comp $(W\backslash C)/V$.

In figure 10 we have suppressed Tense and Case, allowing the complementizer to select a verb directly. We have in both cases also suppressed the application of structural rules and the modal operators needed to control access to those rules. The structural rules, in particular, have no effect on the proof terms we derive. These proof terms are repeated below for convenience.

[12]Note that our heavy use of structural rules is not really relevant to this discussion. Indeed the multimodal control of structural rule packages was developed within categorial grammar because proofs like that in figure 7, with relatively light and uniform use of structural rules, proved deceptively simple and inadequate in the face of natural language data.

[13]There may be an exception to this. If a head can raise to adjoin to another head bringing some uncancelled features along with it, to be cancelled in its new position as part of a complex head, then the higher head may need to select the lower as a functional type.

	CG	TG
who	$R/(NP\backslash S)$	$D \bullet W$
sneezed	$NP\backslash S$	$D\backslash V$
Comp	—	$(W\backslash C)/V$

Figure 8: Lexica for the CG and TG derivations.

$$\dfrac{\dfrac{\dfrac{Nom:NP \vdash Nom:NP \quad x_1:S \vdash x_1:S}{Nom:NP, Pred:NP\backslash S \vdash Pred(Nom):S}\,L\backslash}{Pred:NP\backslash S \vdash \lambda Nom.Pred(Nom):NP\backslash S}\,R\backslash \quad x_2:R \vdash x_2:R}{Wh:R/(NP\backslash S), Pred:NP\backslash S \vdash Wh(\lambda Nom.Pred(Nom)):R}\,L/$$

Figure 9: CG derivation of *who sneezed.*

$$\dfrac{\dfrac{Nom:D \vdash Nom:D \quad \dfrac{x_3:V \vdash x_3:V \quad \dfrac{Wh:W \vdash Wh:W \quad x_1:C \vdash x_1:C}{Wh:W, x_2:W\backslash C \vdash x_2(Wh):C}\,L\backslash}{Wh:W, App:(W\backslash C)/V, x_3:V \vdash App(x_3)(Wh):C}\,L/}{Wh:W, App:(W\backslash C)/V, Nom:D, Pred:D\backslash V \vdash App(Pred(Nom))(Wh):C}\,L\backslash}{App:(W\backslash C)/V, Subj:D \bullet W, Pred:D\backslash V \vdash App(Pred((Subj)_1))((Subj)_2):C}\,L\bullet$$

Figure 10: TG derivation of *who sneezed.*

Fig. 9: $Wh(\lambda Nom.Pred(Nom))$
Fig. 10: $App(Pred(Nom))(Wh)$

The main ways in which the TG term differs from the CG term are the appearance of the variable *App*, which corresponds in the syntax to the presence of the null complementizer, the reduction of Wh from a function to a constant of some sort and the lack of a λ abstractor in front of the $Pred(Nom)$ expression. The first two differences are exactly what we would expect, given that the Wh operator has been given an atomic type. When we "reify" function symbols (such as variable binding operators) as individuals then we need to provide a new explicit function symbol to represent the application of a functor to its arguments. Where the CG derivation constructs a term which represents directly the application of a Wh operator to a one place predicate, the TG derivation yields instead a term which requires further interpretation, in particular a term in which *App* must still be interpreted as function application in order to derive the semantics.

The third distinction goes deeper into the relations between these approaches. The TG term lacks the λ abstractor because the derivation uses only application ($L/$, $L\backslash$) steps with no abstraction ($R/$, $R\backslash$) steps. These abstraction steps are certainly required in a syntax without explicit chains: this is the mechanism whereby the Wh operator will bind the variable Nom without knowing the variable's name in advance. Suppose we interpret Wh as

$$\lambda P.\text{which is the } x \text{ such that } P(x)$$

and apply it to $\lambda Nom.\text{sneezed}(Nom)$. We get

$$\text{which is the } x \text{ such that } (\lambda Nom.Nom \text{ sneezed})(x)$$

which reduces to

$$\text{which is the } x \text{ such that } x \text{ sneezed.} \tag{57}$$

That is, we have managed to rename the variable in the predicate to the variable bound by the Wh operator.

Chain formation is supposed to solve the problem of variable binding in a rather different way. The idea is essentially to rely on movement or chain formation to assure that the operator and bound variable are appropriately named. We can think of the identity of the occurrence of $\langle$*Nom*, *Wh*$\rangle$ (call it "*Subj*") as the "index" which is born by *Nom* and *Wh*. We thus have enough information to ensure that *Wh* can be interpreted as something which binds occurrences of the variable named *Subj* in its scope. For example, if we wished to derive something like the semantic logical form "which is the x such that x sneezed," from the proof term $App(Pred(Nom))(Wh)$, we could try to make the following assignments.

$$\begin{aligned} App &\mapsto \lambda P.\lambda Q.Q(P) \\ Pred &\mapsto \lambda x.x \text{ sneezed} \\ Nom &\mapsto x_{Subj} \\ Wh &\mapsto \text{`}\lambda\text{'}R.\text{which is the } x_{Subj} \text{ such that } R \end{aligned}$$

The λ in the assignment to *Wh* is in scare quotes because we must violate a condition on the reduction of lambda terms in order to obtain the expression we want. Normally, it is not allowed to substitute one expression into another in such a way that a variable free in the former becomes bound in the result. That is precisely what we want to have happen here: we want x_{Subj} in "x_{Subj} sneezed" to be bound by the Wh operator "which is the x_{Subj} such that" when the former is substituted for R. We have taken great pains in the naming of the variable corresponding to Nom with just this end in view. Interestingly, this approach shows some similarities to a recent approach to categorial deduction developed by Mark Hepple (Hepple 1997, 1996). Hepple presents specialized deduction rules which allow him to go from higher order types to first order types. These devices crucially rely on a variant of substitution into lambda terms which allows variable capture by a variable binder, which is exactly the mechanism we are relying on here. Again, the magic works because of the care we have taken with the naming of our variables, via coindexation, so that variable capture should never be *accidental*.

These distinctions seem to derive rather directly from the focus in CG on semantics and in TG on syntax. In particular, CG avoids headaches of variable naming and intentional variable capture by making sure that the syntactic type of the relative pronoun reflects its semantic type. The TG approach on the other hand focuses on building up a strictly syntactic structure to serve as the LF interface. For example, the TG approach bears strong similarities to the way in which well formed formulas are built up in first order logic. There for example we have a rule of formation which says that if ϕ is a wff, then so is $\forall x\phi$, together with a definition of bound and free variables which assures that $\forall x$ will capture all occurrences of x free in ϕ. There is no concern here for the eventual interpretation of $\forall x$ as the universal quantifier. In particular, if we try to implement this formation rule as a structure building function $\lambda P.\forall xP$, then we risk, indeed we really intend, exactly the kind of variable capture that we have seen above. In effect, these differences reflect the fundamental idea in TG that scope-taking is a matter of syntactic configuration, whereas in CG (with its close ties to generalized quantifier theory) scope-taking is really an instance of the function-argument relation.

This is not by any means the end of the story, however, or even of such a story as can be told in this limited space. We have a calculus which is a real grammar logic, complete with left and right rules for all connectives. To be sure, we have focused our attention on a particular class of lexicons, which have properties reflecting ideas of the minimalist program, and these lexica do not require us to use right rules in grammatical derivations. However, we can easily extend our investigation, holding the grammar logic constant. In particular, we can ask what would happen if we *did* use hypothetical reasoning in the derivation of extraction phenomena.

For example, it is a theorem of our system that the partial clause [*Comp sneezed*] can be assigned the type $C/_{\otimes}\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W)$.

$$(W\backslash C)/V,\ D\backslash\Diamond_1\Box_0^{\downarrow}V \vdash C/_{\otimes}\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W)$$

The proof search is elementary and obvious after its first step: we must start with an application of $R/_{\otimes}$. This gives us as the immediate premise of the goal sequent the very same numeration we would have started from in the transformational derivation restricted to first order types.

$$(W\backslash C)/V,\ D\backslash\Diamond_1\Box_0^{\downarrow}V,\ \Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W) \vdash C$$

We can conclude, then, that we could assign to the relative WhP the higher order type

$$C/_{\otimes}(C/_{\otimes}\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W)).$$

This allows it to combine with a partial relative clause in the same way in which the categorial relative pronoun combines with its clause (see figure 9). That is, the goal sequent is derived by the application of $L/_{\otimes}$, with the left premise being the claim that the material in the scope of the pronoun is sufficient to provide it with its argument. Again, the key step of that proof will be the derivation of the goal's left premise by the application of $R/_{\otimes}$. In this case, the main subproof down to the application of $R/_{\otimes}$, is the same as the proof in figure 10. Therefore, we know that the proof term assigned to the succedent type before the application of $R/_{\otimes}$ will be of the form

$$App(Pred((Subj)_1))((Subj)_2)$$

(plus constructors and destructors corresponding to the unary control modals). The action of the $R/_{\otimes}$ rule will cause the chain-variable $Subj$ to be bound, yielding the following labelled sequent.

$$\begin{aligned} &App\colon(W\backslash C)/V,\ Pred\colon D\backslash\Diamond_1\Box_0^{\downarrow}V \vdash \\ &\qquad \lambda Subj.App(Pred((Subj)_1))((Subj)_2)\colon C/_{\otimes}\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W) \end{aligned} \tag{58}$$

If we associate our new higher-order relative pronoun with a fresh variable F then the entire relative clause is of type C with proof term

$$F(\lambda Subj.App(Pred((Subj)_1))((Subj)_2))$$

The F here is strongly analogous to the role of the functional variable Wh in the proof term assigned by the CG proof in figure 9. Semantically, we want to assign it the same kind of value, namely a function which applies another function to its bound variable. So, given that $Subj$ was, in our first-order type based derivation, assigned to the pair

$$\langle x, \lambda S.\text{which is the } x \text{ such that } S\rangle$$

then F should be assigned the following logical form.

$$\lambda R.R(\langle x, \lambda S.\text{which is the } x \text{ such that } S\rangle) \tag{59}$$

All this should be completely unsurprising, despite the sheer weight of mathematical ink, since the type $C/_{\otimes}(C/_{\otimes}\Diamond_1(\Box_0^{\downarrow}D \bullet \Box_0^{\downarrow}W))$ is just an instance of a type formula of the general form $B/_{\otimes}B/_{\otimes}A$, and we have as a theorem that the following sequent

$$A \vdash B/_{\otimes}B/_{\otimes}A$$

is valid in the associative, commutative calculus **LP** (recall that $\otimes$ is just the **LP** product, with free access to permutation and association rules). This is just type raising, and the expression in (59), though made somewhat obscure by the appeal to pairing and projection, is just the semantics of a type-raised argument, taking a function and applying it to itself. The important point here is that, because this is a valid sequent, we do not even need to modify our lexicons in order to accommodate approaches to extraction based on hypothetical reasoning. Once we have a definition of *who* which works in a transformational derivation without hypothetical reasoning, we automatically can derive a type assignment and logical form assignment which works with hypothetical reasoning. In particular, core ideas about the locality of possible movement transformations remain in place.

However, the appeal to hypothetical reasoning on its own does not turn TG into CG, really *because* we do not have to make any modifications to our lexica. In particular, chains persist, leaving us with expressions like

$$\langle x, \lambda S.\text{which is the } x \text{ such that } S\rangle$$

where we have an occurrence of x outside scope of the Wh operator which we intend ultimately to place within that scope. Embedding this in layers of applications and abstractions does not change things, and transformational grammar remains a theory of formation distinct from a theory of interpretation. In light of this, a distinction which reflects concerns of very long standing in both traditions, it is somewhat remarkable to observe once again that the appearance of chains in this system stems directly from the resource conscious use of features which is a very recent feature of transformational grammar, belonging specifically to the minimalist program. That is, our multiplicative treatment of multiple feature assignment leads us to type formulas whose natural structural interpretation is as tuples of syntactic objects all extending a single lexical head, that is, chains. The more things change, the more they stay the same.

The most important lesson here is then that the use of hypothetical reasoning *per se* does not appear to be really relevant to the comparison of CG and TG. We need modify neither our logic nor our lexicons to support CG-style derivations of relative clauses in TG, and our approach to TG derivations uses no resources which are not in principle available to CG. However, it seems fairly clear that the introduction of product types into a categorial lexicon would have profound effects on the hitherto very tight connections in that framework between syntax and semantics.

6 Missing Pieces, Future Work

A number of important gaps remain in these proposals which will have to be filled in future work. We have so far done nothing to implement successive cyclic movement, for example. In Chomsky (1995) this is done by deleting (or erasing) only the feature on the target. In our terms this amounts to allowing moving features access to the structural rule of Contraction as well as movement.

$$\frac{\Gamma[(\Delta,\ \Delta)^j]}{\Gamma[\langle\Delta\rangle^i] \vdash A}\,Contraction$$

There is also another way to give a formula more restricted access to Contraction "through the back door", namely via structural rules of (reverse) distributivity.

$$\frac{\Gamma[((\Delta_1,\ \Delta_2)^i,\ (\Delta_1,\ \Delta_3)^i)^j] \vdash A}{\Gamma[(\Delta_1,\ (\Delta_2,\ \Delta_3)^j)^i] \vdash A}\,Distributivity^{-1}$$

Here the required presence of a j type configuration adds a certain amount of context sensitivity to the multiplication of the resources in Δ_1. However, it is not at this point clear how this can be employed in the context of a movement derivation.

We have also done nothing to implement the distinction between overt and covert movement. In particular, we have not implemented the minimalist idea that features come in strong and weak varieties, and that only strong features force overt movement. Part of the reason for this is that the theory is somewhat vague on this point. In various places, Chomsky seems to want "strength" itself to be a feature. So it is not yet clear whether "strength" should be treated in a resource conscious manner, or rather in an additive manner, or by some other means completely (e.g., with yet more modal control).

Finally, our implementation of locality conditions on movement as side conditions on the application of structural inference rules leaves a good deal to be desired. In particular such side conditions threaten the compleness of the grammar logic, and in general have a sort of extra logical flavor. Ideally the same conditions should be implemented directly in the logic via structural postulates and corresponding inference rules. It is not guaranteed that this is possible. For example, global economy conditions are unlikely to be implementable except as extra conditions on which otherwise well formed proofs are admissible. However, local conditions, in particular c-command based conditions like the versions of Minimal Link found towards the end of Chomsky (1995) look more tractable, since in a strictly binary branching structure such as our resource configurations the c-command configuration is relatively easy to describe.

In spite of these shortcomings of what is after all still a very preliminary investigation, it seems likely that the devices of multimodal type logics can provide us with an extremely useful tool for investigating different theories within a transformational framework, and also for investigating different frameworks whose fundamental biases are rather different. It is important to keep in mind that the system we have presented here is like other type logical grammatical calculi in the Lambek tradition in that it is not meant as a formal theory of Universal Grammar. The logic itself is useful precisely because it is not restrictive. This is what allows such different syntactic theories as categorial grammar and transformational grammar to be embedded within it. It is not a formal theory of Universal Grammar, but rather a tool for performing precise scientific calculations within the theory of grammar, very broadly construed. Because this system is intended as a very general grammatical calculus, and because it is founded on mathematical principles as well as grammatical ones, it helps to dissolve the barriers between frameworks which restrictive formalisms impose, and is thus a healthy antidote to (but not a replacement for) formalistic theories of Universal Grammar. We have seen, for example, that different theories of chains can be compared without regard for whether they approach chains derivationally or representationally, because both derivations and representations are simultaneously present and synchronized in type logical grammar. This approach remains useful even when we are considering extraction phenomena more abstractly, and in particular theoretical approaches to them which do not utilize chains at all. We have seen that we can gain considerable precision in meta-theoretic investigations, isolating deep pretheoretical commitments from the mechanisms which help to realize them. Perhaps the best way to see the utility of a general grammatical calculus is to view it as an approach which respects individual ideas about syntax as paramount, taking precedence over the larger frameworks which knit them together in various ways.

References

Bayer, S. and Johnson, M. (1995). Features and agreement, *Proceedings of the ACL*, Association for Computational Linguistics, pp. 70–76.

Berwick, R. C. and Epstein, S. D. (1995). Computational minimalism: The convergence of the minimalist syntactic program and categorial grammar, *TWLT-10: Algebraic Methods in Language Processing*, AMAST, Faculty of Computer Science, University of Twente, Enschede, the Netherlands.

Bobaljik, J. (1995). In terms of merge: Copy and head movement, *MIT Working Papers in Linguistics 27*, MIT Linguistics Dept., pp. 41–64.

Brody, M. (1995a). *Lexico-Logical Form: A Radically Minimalist Theory*, MIT Press, Cambridge, MA.

Brody, M. (1995b). Perfect syntax, Colloquium talk given at the University of Tübingen.

Brody, M. (1997). Projection and phrase structure, *Technical Report 98*, SFB 340, University of Tübingen. Draft of an article forthcoming in *Linguistic Inquiry*.

Chomsky, N. (1975/1955). *The Logical Structure of Linguistic Theory*, University of Chicago Press, Chicago.

Chomsky, N. (1995). *The Minimalist Program*, MIT Press, Cambridge, MA.

Collins, C. (1997). *Local Economy*, MIT Press, Cambridge, MA.

Cornell, T. L. (1997). A type-logical perspective on minimalist derivations, *Proceedings of Formal Grammar '97*, Aix-en-Provence, France.

Gabbay, D. M. (1994). Lds—labeled deduction systems, *Technical Report MPI-I-94-223*, Max Planck Institut für Informatik, Saarbrücken. To appear from Oxford University Press.

Girard, J.-Y. (1987). Linear logic, *Theoretical Computer Science* **50**: 1–102.

Hepple, M. (1996). A compilation-chart method for linear categorial deduction, *Proceedings of COLING-96*.

Hepple, M. (1997). Maximal incrementality in linear categorial deduction, *ACL-EACL '97 Proceedings*, ACL and EACL, pp. 344–351.

Heylen, D. (1997). Generalisation and coordination in categorial grammar, *in* G.-J. M. Kruijff, G. V. Morrill and R. T. Oehrle (eds), *Formal Grammar 1997*, Aix-en-Provence, France.

Koopman, H. (1994). Licensing heads, *in* D. Lightfoot and N. Hornstein (eds), *Verb Movement*, Cambridge University Press, Cambridge, pp. 261–296.

Lambek, J. (1958). The mathematics of sentence structure, *American Mathematical Monthly* **65**: 154–170.

Lambek, J. (1961). On the calculus of syntactic types, *in* R. Jakobson (ed.), *Structure of Language and Its Mathematical Aspects*, Proceedings of the Twelfth Symposium in Applied Mathematics, Providence, Rhode Island.

Lecomte, A. (1997). POM-nets and minimalism, *in* C. Casadio (ed.), *Proceedings of the IV Roma Workshop: Dynamic Perspectives in Logic and Linguistics*, SILPS Group in Logic and Natural Languages.

Moortgat, M. (1996). Categorial type logics, *in* J. van Benthem and A. ter Meulen (eds), *Handbook of Logic and Language*, Elsevier, chapter 2.

Moortgat, M. and Oehrle, R. T. (1997). Structural abstractions, *in* V. M. Abrusci and C. Casadio (eds), *Proceedings of the 1996 Roma Workshop: Proofs and Linguistic Categories*, Dept. of Philosophy, Univ. of Bologna, Bologna, Italy.

Morrill, G. (1994). *Type Logical Grammar*, Kluwer, Dordrecht.

Oehrle, R. T. (1994). Term-labeled categorial type systems, *Linguistics and Philosophy* **17**(6): 633.

Stabler, E. (1997a). Derivational minimalism, *in* C. Retoré (ed.), *Logical Aspects of Computational Linguistics*, Springer, Berlin, pp. 68–95. LNAI 1328.

Stabler, E. P. (1997b). Computing quantifier scope, *in* A. Szabolcsi (ed.), *Ways of Scope Taking*, Kluwer, Boston.

Stabler, E. P. (forthcoming). *Acquiring and Parsing Languages with Movement*, Basil Blackwell, Oxford.

Troelstra, A. S. (1992). *Lectures on Linear Logic*, Vol. 29 of *Lecture Notes*, CSLI, Stanford.