Chapter 1

Island Effects in Type Logical Approaches to the Minimalist Program

Thomas L. Cornell
University of Tübingen

Abstract. Categorial approaches to the minimalist program have problems with cyclic movement and island effects, where the island is induced by a feature that has been checked. Chomsky [2] proposes a single solution to both problems: "interpretable" features do not delete even when checked, and therefore (a) they can be reused in cyclic movement and (b) they induce more persistent islands. We propose an implementation of Chomsky's Minimal Link Condition (MLC) in a multimodal categorial grammar and show how interpretable features can be given (very limited) access to contraction and weakening so as to prevent MLC violations even after they have been checked. Since interpretable features only appear negatively in the sequent antecedent, giving them access to contraction does not threaten the decidability of the logic.

1 The Problem

Categorial minimalist systems [7, 3, 1] run into a problem in handling island effects. These are typically induced by expressions all of whose features have already been checked. If feature checking is to be implemented as a logical elimination rule, then these expressions are gone from the proof at the point where they are needed to block communication between a higher functor and a lower argument.

(1.1) *(guess) to whom they remembered *which book* to give

(1.2) *John seems that *it* was told that IP

In (1.1), the Wh feature of *which book* should have been cancelled against the Wh feature of the complementizer selected by *remember*. In (1.2) the "Extended Projection Principle" (EPP) feature of *it* should have been cancelled against the finite Tense node in its clause. But if they were, then nothing would be left to block the movements of *to whom* and *John*, respectively.

Problems also arise in dealing with successive cyclic movement.

(1.3) We are likely t_3 to be asked t_2 to t_1 build airplanes.

The problem is that the same feature of *we* (again, the EPP feature) is being cancelled in several places. For example, the pronoun *we* must be so constituted that all of the following sentences are grammatical.

(1.4) We build airplanes.

(1.5) We have been asked to build airplanes.

(1.6) We are likely to be asked to build airplanes.

(1.7) We seem to be likely to be asked to build airplanes.

We assume, pursuing a standard transformational account of such examples, that raising is to be analyzed as the combination of *we* as an argument with a number of distinct functors. We assume as well, pursuing this account within a resource-sensitive, type-logical framework, that this is to be accomplished by assigning *we* some sort of product type, or *chain* type [3], each of whose members is a resource that must be used up during the derivation. To deal with cyclic movement (and island effects, as we will see) this product must contain a number of instances of the same feature, worse, a number which cannot be determined in advance and recorded permanently in the lexicon.

2 Chomsky's Solution

Chomsky's analysis of both island effects and successive cyclic movement in [2] is based on the same essential idea: that a certain class of features—Chomsky identifies them as the *interpretable* features—do not delete during a derivation, even if they have been checked, and, as a result, they can be used over again. In fact, as a result, they do not even ever have to be checked. So for example there is now no syntactic motivation for covert Wh-movement in multiple Wh questions. In a sentence like the following,

(1.8) Who gave the lucky couple which coffee maker?

there is evidently no necessity for *which coffee maker* to raise to Comp to check its Wh feature: it is an interpretable feature and as such is tolerated at LF, so it does not have to be checked, and the Wh feature of Comp itself has already been checked by *who*. These ideas about the syntactic treatment of "interpretable" features make it appear that the minimalist program, which elsewhere seems to fall squarely in the camp of linearly resource sensitive approaches to grammar, radically departs from a program of strict resource sensitivity.

The Wh feature of the Comp node itself still does have to be checked by something, so Wh in situ is still ruled out (in English) for simple Wh questions. This offers a key to "having one's resources and consuming them too." Chomsky's claim is that the features of the functional categories that attract moving expressions are formally [−interpretable], and therefore, in general, they will delete once checked. So, in effect, the resources which a numeration makes available for syntactic construction remain fixed on one side of the ledger, which in turn strictly limits the degree to which the computational system can access a single resource multiple times on the other side of the ledger.

This use of interpretable features yields a solution to the problem of island effects as well. The claim is that the features of the island-offender which are being attracted belong to the interpretable class (Wh in the case of $\bar{A}$-movement, D in the case of A-movement). Therefore these features of the island-inducer have not deleted after having been checked. Furthermore, they are technically speaking closer to the attracting functional head than are the respective island-violators. Accordingly, movement out of the island is blocked by the Minimal Link Condition.

3 Chomsky's Proposals in Type Logical Terms

Like much of the minimalist program, Chomsky's proposals here can be straightforwardly embedded within a type-logical deductive framework. However, the relative insensitivity to resources evident in these proposals clearly forces the embedding outside the "categorial" type logics, i.e., outside the bounds of multiplicative intuitionistic linear logic. Clearly, we will need to give certain resources (corresponding to the "interpretable" features) access to the structural rule of Contraction,

$$\frac{\Gamma[(\Delta, \Delta)] \vdash B}{\Gamma[\Delta] \vdash B}\ LC$$

which, read bottom-up, in the direction of proof search and grammatical derivation, allows for the multiplication of the resource structure Δ.

Furthermore, we will always need one more copy of a given interpretable feature than we can actually cancel, so that it remains available to induce island effects. Since it cannot be removed by a left slash rule (alias slash elimination), it must either be removed by Weakening,

$$\frac{\Gamma[\Delta_i] \vdash B}{\Gamma[(\Delta_1, \Delta_2)] \vdash B}\ LW_i$$

or else we need to tolerate axioms of the form

$$\Gamma[A] \vdash A$$

where Γ is required to contain only interpretable features, apart from A.

Devices to control access to these rules are already built into Linear Logic: this is what the exponentials $!A$ and $?A$ are used for. Essentially, our proposals amount to a more contextually restricted variant of an exponential couched within the logic of unary residuation rather than the basically S4 logic of the exponentials.

3.1 Minimalist derivations with strict resource sensitivity

Setting aside for a moment the problems introduced by Chomsky's treatment of interpretable features, we assume an implementation of the minimalist program based on [4] (and see also [3]) in type logical terms as follows.

1. We treat feature checking as type cancellation, i.e., via the use of left slash rules.

2. We treat minimalist derivations as proof searches (i.e., proofs read bottom-up) in a multimodal Gentzen sequent calculus. Goal sequents are to be read as "the given numeration (multiset of lexical formulas) derives the succedent type."

3. Movement is to be forced by the resource sensitivity of the system, according to Chomsky's Last Resort Principle. Therefore, lexical formulas which must have features checked in several places are implemented as product types. Here "formal features" are first class resources.

4. Contra Chomsky, we include categorial selection features in the syntax. (We allow category features, and selection features are their obvious logical dual [5].)

We assume the lexicon contains only first order formulas, that is: basic types, tuples of basic types and functions from basic types to tuples of basic types. We think of the basic types as being types of chain positions and product types as being the types of (XP) chains. A functional type takes several arguments to build up a phrase, which is then associated with the chain positions which are the type of the function's ultimate value.

We assume the base multimodal logic of residuation as presented, for example, by Moortgat [8]. This is essentially a multimodal version of the non-associative Lambek calculus [6], with the addition of "unary products", i.e., unary modal operators subject to the appropriate residuation law. The logical rules we employ are thus absolutely standard and we present here only the packages of structural rules we have superimposed on that logic.

The core operators that we will use will be the basic **NL** product $\bullet$ and two unary modals $\Diamond_0$ and $\Diamond_1$, together with their residuals. The $\bullet$ product will be used to glue features (i.e., chain positions) together and the two unary modals will be used to distinguish the use of a feature as a categorial selection feature ($\Diamond_0$) or as a formal feature ($\Diamond_1$). We will also make use of the following binary control modals: $\otimes, >, <, \lhd, \rhd$, whose residuals we will typically not make use of.[1] These control modals supply crucial context in the definitions of various packages of structural rules.

We will use four distinct packages of structural rules. The first, the Numeration package (rules N1–N2), simply defines the numeration as a multiset, i.e., a sequent whose resources have free access to both permutation and association.[2] Another package, also simple, defines Merge, the operation which removes from numeration resources their freedom of movement (rule Mg). The Move package (rules Mv1–Mv3) implements the transformational component, allowing limited access to permutation and association to appropriately marked formulas. Finally, the Feature Checking package (rules F0c–F2) defines the checking configurations for formal and selectional features.

The $\otimes$ product is defined by the Numeration package to be the associative commutative product of **LP**.

$$N1: \ A \otimes B \longrightarrow B \otimes A \tag{1.9}$$

$$N2: \ A \otimes (B \otimes C) \longleftrightarrow (A \otimes B) \otimes C \tag{1.10}$$

The main sequents of interest are sequents whose antecedents are pure numeration structures and whose consequents are basic categories, in particular c, the category of sentences.[3]

The numeration product can be "stiffened" by the Merge transformation, which converts substructures of the numeration into headed tree structures. We use $>$ and $<$ for building these "syntactic objects", with the narrow end taken to be pointing towards the head. (This phrase structure notation is borrowed from Stabler's work, e.g., [10]). That is, $A < B$ indicates that B is the complement of A and $A > B$ indicates that A is the specifier of B. It is only in these tree structures—the hypothesis goes—that movement

[1] We assume that they are subject to the same logical rules as other residuals, i.e., they are present in the logic, but not in the lexicon. Since they do not appear in lexical type formulas, they will also not appear in grammatical derivations, and so we ignore them from now on.

[2] In [2], Chomsky distinguishes the numeration, which is and remains a multiset of lexical items, from the collection of *syntactic objects* which is essentially the working memory of the computational component. It is also a multiset. If one thinks of lexical items as trivial syntactic objects then there is no need to distinguish the numeration (proper) from the collection of syntactic objects. Here we refer to the combination of these two multisets simply as the numeration.

[3] Observe that this means we do not get a language model of the sort made familiar in [9], in which the "prosodic" dimension of linguistic signs is supplied by a set of strings (**L**) or trees (**NL**) corresponding to the antecedents of the main grammar theorems. We still can get logical forms via the Curry-Howard morphism, but the "prosodic" dimension is harder to get, since we have not done anything to distinguish overt movement from covert movement.

transformations can be adequately controlled.

$$Mg: \ A \otimes B \longrightarrow \begin{cases} (A < B) & \text{, if } A \text{ is of the form } C/D; \\ (B > A) & \text{, if } A \text{ is of the form } A' < A'' \text{ or } A' > A''. \end{cases} \tag{1.11}$$

We use unary modals to distinguish two types of features: those involved in categorial selection, marked with $\diamond_0$, and those involved in movement—Chomsky's "formal features"—marked with $\diamond_1$. Only features marked with a $\diamond_1$ are visible to the attraction system, i.e., only they can move. Note that if our functor types look like A/B for some atomic B, and our argument types looked like $\diamond_i B$, then we would be in trouble since we have no way to get rid of the diamond. If we want to use unary modals to license access to structural rules then we must provide a way of removing them, either by making all of our functors look like $A/\diamond_i B$ or by making our chain position types look like $\diamond_i \Box_i^{\downarrow} B$. This latter is what we will assume, i.e., that all diamonds are ultimately "self-cancelling" by the use of the rules $L\diamond$ and $L\Box^{\downarrow}$. This self-cancellation may seem vacous, but it is not. Formally,

$$\diamond \Box^{\downarrow} A \longrightarrow A$$

by the residuation rule for unary operators, so the type denoted by $\diamond\Box^{\downarrow}A$ must be a subset of the type denoted by A: in our case it is the proper subset which is subject to the various structural postulates that refer to the $\diamond$ operator.

We also take it that in a formula $\diamond_1 A \bullet B$, that is, a chain type whose first member is marked with a movable diamond, A can "pied pipe" B. In the following postulates we take $A^{\diamond}$ to be a formula either whose main functor is $\diamond_1$ or else whose main functor is $\bullet$ and whose leftmost proper subformula has main functor $\diamond_1$. In the Move package we make use of a binary control modal $\lhd$ which connects the moving type to the rest of the structure. Transformational theory would have the movement accomplished in a single step, but here we must define it as the succession of very small steps. Accordingly the intermediate structures have no status in the theory, so we so not want to use a product that has theoretical status.

$$Mv1: \ (B < A^{\diamond}) \longrightarrow (A^{\diamond} \lhd B) \tag{1.12}$$
$$Mv2: \ (A^{\diamond} > B) \longrightarrow (A^{\diamond} \lhd B) \tag{1.13}$$
$$Mv3: \ (C > (A^{\diamond} \lhd B)) \longrightarrow (A^{\diamond} \lhd (C > B)) \tag{1.14}$$
$$Mv4: \ (C < (A^{\diamond} \lhd B)) \longrightarrow (A^{\diamond} \lhd (C < B)) \tag{1.15}$$
$$Mv5: \ (C \otimes (A^{\diamond} \lhd B)) \longrightarrow (C \otimes (A^{\diamond} > B)) \tag{1.16}$$

Rule $Mv5$ implements the condition that all movement must be to the root (must "extend its target"). We can recognize the root of a syntactic object as the point at which we make contact with the numeration. For simplicity we subsume under this case the case where the numeration contains a single syntactic object, allowing C to be empty, in effect.

In the Feature Checking package, we must maneouver a single feature into a $\bullet$ configuration with the head of its structure. We have to consider two cases: for categorial selection features, complement position is a checking position, so a $\Diamond_0$ marked feature in complement position can simply reassociate with the head to be in the appropriate configuration for the application of $L/$ ($F6$, $F7$). Specifier positions are checking configurations for both types of features. Specifiers require more complex handling, especially since Chomsky allows for the possibility of multiple specifiers in some languages. As a result we require a second "auxiliary product" $\rhd$, for building up structures that arise in intermediate stages of the feature cancellation process.

Rules $F1$ and $F2$ allow a feature (either singleton or part of a feature bundle) in any specifier position to cancel by placing it in the $\rhd$ relation with the head term. From there it is simply a matter of finding the ultimate head of the structure, either via rule $F4$ or $F5$.

$$F1: \ (\Diamond_i A > B) \longrightarrow (\Diamond_i A \rhd B) \tag{1.17}$$

$$F2: \ ((\Diamond_i A \bullet B) > C) \longrightarrow (B > (\Diamond_i A \rhd C)) \tag{1.18}$$

$$F3: \ (A \rhd (C > B)) \longrightarrow (C > (A \rhd B)) \tag{1.19}$$

$$F4: \ (A \rhd (B < C)) \longrightarrow ((A \bullet B) < C) \tag{1.20}$$

$$F5: \ (A \rhd B) \longrightarrow (A \bullet B),\ B \text{ unheaded.} \tag{1.21}$$

$$F6: \ (B < (\Diamond_0 A \bullet C)) \longrightarrow ((B \bullet \Diamond_0 A) < C) \tag{1.22}$$

$$F7: \ (B < \Diamond_0 A) \longrightarrow (B \bullet \Diamond_0 A) \tag{1.23}$$

3.2 The Minimal Link Condition

At the core of the theory of locality in the minimalist program lies the Minimal Link Condition (MLC). Let K range over functors (heads) and α and β over chain types.

> **The Minimal Link Condition.** K attracts α only if there is no β, β closer to K than α, such that K attracts β. ([2], p. 311)
>
> **Closeness.** If β c-commands α and τ is the target of raising, then β is closer to K than α unless β is in the same minimal domain as (a) τ or (b) α. ([2], p. 356)

Here "minimal domain" of a head essentially just means the expressions attached to a projection of that head, i.e., the "off-spine" members of its phrase. This definition entails that arguments in the target domain and the source domain cannot interfere with movement. Only members of domains properly between the source position and the target position can intervene.

Therefore, we must take some pains to distinguish for a moving expression whether it is still in its source domain, in some intermediate domain, or in its target domain. We therefore propose the following refinement of the movement package to allow a precise implementation of the MLC. We will split the auxiliary product $\lhd$ in three: $\lhd_1$, $\lhd_2$ and $\lhd_3$. Rules $Mv1$ and $Mv2$ must be modified so that they introduce $\lhd_1$, the initial "state" of this product. Movement rule $Mv4$ represents the configuration in which the moving expression $A^\diamond$ leaves one domain (B) and enters the domain of a new head (C), so $Mv4$ is where we change state:

$$Mv4: \ (C < (A^\diamond \lhd_i B)) \longrightarrow (A^\diamond \lhd_j (C < B)) \tag{1.24}$$

where, if $i = 1$ then $j = 2$ and if $i = 2$ then j can be either 2 or 3. Note that if $i = 3$ then the derivation crashes, since we are supposed only to enter state 3 in the target domain. $Mv5$ must then be modified to require $\lhd_3$.

That leaves us with $Mv3$, which is where we actually impose the MLC.

$$Mv3: \ (C > (A^\diamond \lhd_i B)) \longrightarrow (A^\diamond \lhd_i (C > B)) \tag{1.25}$$

Recall that any legitimate intervener must be found in a strictly intermediate domain, so while we are in state 2. Rule $Mv3$ will therefore operate for all $\lhd_i$, but when $i = 2$ it will be subject to the side condition that C not count as an intervener. That is, the leftmost feature of C must not be the same as the leftmost feature of $A^\diamond$. Recall that it is the leftmost feature of $A^\diamond$ that will be cancelled at the end of the movement step, so this is the feature that counts as what is being attracted. C could only be attracted by K if its leftmost feature could be cancelled by K as well.

Note that this condition rests crucially on the assumption that, at the point of the derivation at which $A^\diamond$ crosses C, C still has the relevant blocking feature. If that feature *can* be cancelled (by the head of B) then there is sure to be a derivation in which it *has* been cancelled at the point at which $A^\diamond$ passes by, and so is no longer available to block movement. We turn to this problem in the next subsection.

3.3 Controlling Access to Contraction and Weakening

Now we return to considering the difference, within formal features, between "[+interpretable]" and "[–interpretable]" features, and we introduce a further Interpretable Features package for handling controlled access to contraction and weakening. There are now essentially three types of features we must deal with: features which are not allowed to move but must be cancelled where they are merged into a structure, features which are allowed to move but not to multiply, and features which are allowed both to move and to multiply. We will introduce a new control modal, $\diamond_2$, to mark these [+interpretable] features. We redefine the movement rules so that $A^\diamond$

stands for a formula with a diamond of whichever movable sort—the movement rules themselves do not distinguish between the two types of formal features.

The two moveable diamonds differ in their behavior at the end of a movement transformation, however. Once we have placed a $\diamond_2$ formula in a specifier position, instead of cancelling it, we cancel a *copy* of it.

$$I1: \ (\diamond_2 A > B) \longrightarrow (\diamond_2 A > (\diamond_2 A \rhd B)) \tag{1.26}$$

$$I2: \ ((\diamond_2 A \bullet C) > B) \longrightarrow ((\diamond_2 A \bullet C) > (\diamond_2 A \rhd B)) \tag{1.27}$$

In the case if $I2$, there is a further complication to consider: either we intend to move the formula to cancel A again (the case of successive cyclic movement), or we are done with A and wish to keep it around only to induce islands for other formulas' movements. The Feature Cancellation package assumes that the feature to be cancelled is left-most in its $\bullet$-tuple, so features that we are done with have to be gotten out of the way. We can do this by using the $\diamond_2$ to allow permutation and association within a chain type.

$$I3: \ (\diamond_2 A \bullet B) \longrightarrow (B \bullet \diamond_2 A) \tag{1.28}$$

$$I4: \ ((B \bullet C) \bullet \diamond_2 A) \longrightarrow (B \bullet (C \bullet \diamond_2 A)) \tag{1.29}$$

The purpose of the $I3$ and $I4$ rules, once again, is to get used-up interpretable features out of the way of any remaining syntactically active features.

The rules $I1$ and $I2$ give us our controlled access to Contraction. Note that every application of Contraction introduces one feature which must be cancelled and one "generator" feature, from which more copies can be made. As a result we will always end up with one uncancelled—and uncancellable—generator feature, which we will have to account for by other means than a logical cancellation rule. We propose the following restricted weakening rules.

$$I5: \ (B < \diamond_2 A) \longrightarrow B \tag{1.30}$$

$$I6: \ (B < (\diamond_2 A \bullet C)) \longrightarrow (B < C) \tag{1.31}$$

$$I7: \ (\diamond_2 A > B) \longrightarrow B, \ B \text{ unheaded.} \tag{1.32}$$

$$I8: \ ((\diamond_2 A \bullet C) > B) \longrightarrow (C > B), \ B \text{ unheaded.} \tag{1.33}$$

The side condition on the second pair of weakening rules is meant to ensure a condition which we wish to hold in general, namely that we can only throw away a $\diamond_2$ formula when it no longer c-commands anything. In that case we can be assured that it induces only a vacuous island, since there is nothing that could possibly move past it.

Note that it is possible to remove $\diamond_2$ formulas too soon: $\diamond_2 A$ in $I5$–$I8$ could conceivably be attracted by some higher head from such a configuration. However, if there is no closer instance of A to the attracting head (i.e., movement is possible in the first place) then there is no other formula in the structure which could cancel against the attracting head, and no derivation would be possible.

4 Concluding Remarks

Chomsky's theory of [+interpretable] features is one of the most fundamental points at which the minimalist program seems—at least at first glance—*not* to be like categorial grammar. These ideas heavily disrupt the perceived parallelism between feature checking in minimalist derivations and logical elmination rules in categorial derivations, in particular since it appears that most features of argument categories turn out to be [+interpretable] and therefore strictly outside of this neat, highly resource conscious derivation scheme. We hope to have shown here that the standard mechanisms of multimodal type logical grammar can be employed to rein in the excesses of this class of features. In particular, the introduction of (forms of) contraction does not threaten the decidability of the system, since they may only be applied to chain types, i.e., logically negative atoms of the sequent antecedent.

Nonetheless an explanatory debt is incurred which will be hard to pay off. We have here treated Chomsky's label "[+interpretable]" as nothing more than a label, but Chomsky clearly intends to offer an explanation for the reusability of this class of features. Namely, these are supposed to be exactly those features which are needed in the derivation in order to construct an LF. But our derivations are just type checking computations. There is no sense in which an LF is constructed directly out of the types in the goal sequent. Such a derivation produces an LF implicitly: it is to be read off the structure of the proof. For example, one could easily enough use something like the Curry-Howard morphism to construct a GB-style LF structure rather than a lambda-term, if that was desirable. But then the "interpretability" of a given type has to do with whether or not the chain position it types is used in that LF proof term. This is an altogether different relation than that which holds in the proof machinery itself among type formulas subject to structural inference rules.

Bibliography

[1] Robert C. Berwick and Samuel D. Epstein. Computational minimalism: The convergence of the minimalist syntactic program and categorial grammar. In *TWLT-10: Algebraic Methods in Language Processing*, Enschede, the Netherlands, 1995. AMAST, Faculty of Computer Science, University of Twente.

[2] Noam Chomsky. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.

[3] Thomas L. Cornell. A type-logical perspective on minimalist derivations. In *Proceedings of Formal Grammar '97*, Aix-en-Provence, France, August 1997.

[4] Thomas L. Cornell. A calculus for categories and transformations. Ms. in prep., University of Tübingen, 1998.

[5] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[6] J. Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Language and Its Mathematical Aspects*, Providence, Rhode Island, 1961. Proceedings of the Twelfth Symposium in Applied Mathematics.

[7] Alain Lecomte. POM-nets and minimalism. In Claudia Casadio, editor, *Proceedings of the IV Roma Workshop: Dynamic Perspectives in Logic and Linguistics*. SILPS Group in Logic and Natural Languages, 1997.

[8] Michael Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1996.

[9] Glynn Morrill. *Type Logical Grammar*. Kluwer, Dordrecht, 1994.

[10] Edward Stabler. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 68–95, Berlin, 1997. Springer. LNAI 1328.