A computational guide to the dichotomy of features and constraints[*]

Thomas Graf
Department of Linguistics
Stony Brook University
Stony Brook, NY 11794-4376, USA
mail@thomasgraf.net

Abstract A contentious issue in the Minimalist literature is whether certain phenomena are best described in terms of features or constraints. Building on recent work in mathematical linguistics, I argue that constraints and features are interchangeable in Minimalist syntax. This does not invalidate the feature-constraint debate, though. Rather, the interdefinability of the two points out an unexpected loop hole in the formalism that allows for massive overgeneration and produces incorrect typological predictions. At the same time, the feature-constraint equivalence can be helpful in plugging this loop hole.

Keywords: computational linguistics; generative capacity; model theoretic syntax; Minimalist grammars; subcategorization; syntactic categories; typology

1 Introduction

Features and constraints are essential components of the working syntactician's toolbox. There has always been a technical tension between the two, as some accounts couched in terms of one device can easily be rephrased using the other, thus prompting the question which one of the two is to

[*] I am greatly indebted to the three anonymous reviewers, whose extensive comments led to some major changes — and hopefully improvements — in the presentation of the formal results. Parts of this paper also profited from discussions at the 2015 workshops *What Drives Syntactic Computation? Alternatives to Formal Features* and *Generative Syntax in the Twenty-First Centure: The Road Ahead*, an extensive exchange on the blog *Faculty of Language*, and feedback from many friends and colleagues over the course of the last three years. I am particularly grateful to John Bailyn, Bob Berwick, Meaghan Fowlie, Jeffrey Heinz, Norbert Hornstein, Greg Kobele, Richard Larson, Gereon Müller, Gillian Ramchand, Henk van Riemsdijk, Ed Stabler, and Peter Svenonius.

be preferred. For example, the Phase Impenetrability Condition of Chomsky (2000) is recast by Abels (2003) in terms of the feature specifications of phase heads. Similarly, the constraints of classical binding theory are replaced by feature valuation in Reuland (2001) and Heinat (2006). In addition, the feature-constraint dichotomy is indirectly linked to the divide between derivational and representational formalisms. GPSG (Gazdar et al. 1985), for instance, endorses the purely derivational nature of context-free grammars and encodes all its constraints in terms of features and rewrite rules. GB (Chomsky 1981), on the other hand, treats constraints as first-class objects in the grammar that do not need to be reencoded in more primitive terms.

The advent of Minimalism (Chomsky 1993, 1995a) has pushed features and constraints into direct opposition. Features now act as the fuel driving the syntactic engine, while constraints have been reconceptualized as Bare Output Conditions imposed by the interfaces. This split is still present in recent incarnations of Minimalism (Chomsky 2007, 2008), and the Minimalist literature as a whole now spans a wide range of proposal, some based on features (Nevins 2007; Adger 2010; Müller 2010, 2014; Preminger 2011), others on constraints (Fox & Pesetsky 2005; Bruening 2014; Richards 2016). In light of these recent developments, it is crucial to ask (and answer!) whether the distinction between features and constraints has some linguistic substance or is but a matter of notation.

In this paper, I explore the feature-constraint dichotomy from a computational perspective. I take as my vantage point Graf's (2013a) proof that features and constraints are expressively equivalent. More precisely, Graf shows that the features involved in subcategorization are already powerful enough to encode any constraint that is definable in *monadic second-order logic* (MSO). As virtually all constraints in the syntactic literature satisfy this property, they can be precompiled into the lexicon as selectional restrictions. In the other direction, all types of Minimalist feature systems (cf. Adger 2006, 2010; Adger & Harbour 2008) have equivalent MSO-constraints. Put together, this yields the mathematical theorem that features and constraints are interchangeable in Minimalism.

The relevance of mathematical theorems, however, hinges on the validity of their technical assumptions. Graf's theorem is stated with respect to specific notions of constraints and features, and it uses *Minimalist grammars* (MGs; Stabler 1997) as its formalization of Minimalist syntax. The main thrust of this paper is that these details are ultimately inessential and the feature-constraint equivalence applies to every formalism with subcategorization, including mainstream Minimalism. This does not mean, though, that investigating the nature of features and constraints is a fruitless endeavor and linguists should move on to new pastures. The exact opposite is the case. The fact that subcategorization can express every MSO-definable constraint (rather than just those that happen to be syntactically natural) entails that our grammar formalisms overgenerate on an enormous scale and make stunningly incorrect typological predictions. As in any other case of overgeneration, we need a principled theory to rein in the power of the formalism. The upside of the feature-constraint equivalence is that both types of approach — feature-based and constraint-based — can contribute to this goal. In the future, we may be able to fully unify features and constraints into a more abstract concept, but at the moment their interdefinability is enough to make both perspectives equally worth exploring in the search for a more restricted theory of syntax.

The paper moves through these points as follows: Section 2 familiarizes the reader with the basics of MGs (2.1) and MSO (2.2) so that they can follow the informal presentation of the feature-constraint equivalence proof in Sec. 3. The proof proceeds in two steps. The first one is a translation from standard MGs to feature-free MGs definable in MSO that still produce the same output structures and are thus expressively equivalent. The second step then goes in the reverse direction and reduces MSO-definable constraints to subcategorization requirements. Together, these two translations establish that features and constraints are freely interchangeable. Section 4 argues that this formal theorem is of relevance to linguistics even though syntacticians may initially object to peculiarities of MGs (4.1), the use of category and subcategorization features (4.2), the assumed application domain of constraints (4.3), or the exclusive focus on generative capacity (4.4). With the relevance of the theorem reasserted, I then take an in-depth look at its implications in Sec. 5. I show that MSO-definable constraints are too powerful for syntax, make wrong typological predictions, and bleed existing constraints of their restrictive force (5.1). The negative effects are shown to be pervasive and difficult to fix in a principled manner given our current understanding. Section 5.2 subsequently outlines current attempts to address the problem and how empirical research can contribute to this formal enterprise.

When all these points are taken into consideration, the feature-constraint equivalence turns out to be both a deeply concerning problem and a great opportunity. While it highlights serious shortcomings in our current ability to limit the power of syntactic theories, it also reveals that the dichotomy of features and constraints in the literature is not about right and wrong, true and false, but rather about different perspectives on one and the same abstract concept. In some cases one perspective may be easier to grasp than the other, but both further our understanding of syntax.

2 Mathematical models of constraints and Minimalist syntax

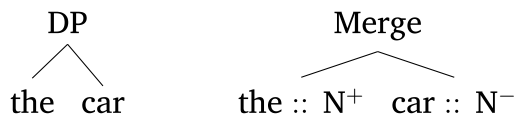
Graf's (2013a) theorem that features and constraints are interchangeable is obtained with respect to a specific mathematical foundation: MGs as the syntactic formalism (2.1), and logical formulas as the abstract counterpart to syntactic constraints (2.2). MGs are closely modeled after Minimalist syntax, but differ in some of their assumptions about the feature calculus (Sec. 4.1, though, explains how these differences can be done away with). The view of syntactic constraints as logical formulas is very intuitive once one realizes that both are means of picking out sets of well-formed structures.

2.1 Minimalist Grammars as a formalization of Minimalism

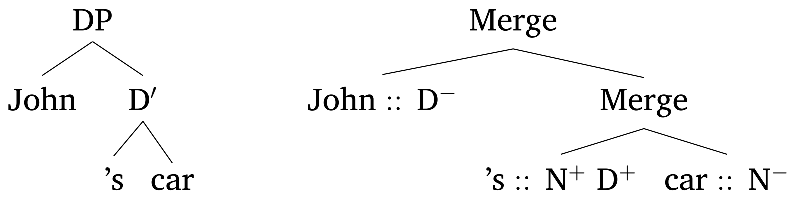
MGs were introduced in Stabler (1997) as a model of Minimalist syntax that provides a rigorous basis for mathematical proofs. Consequently, MGs include only the most basic concepts of Minimalism: structures are built via Merge and Move, and both operations are triggered by features. Many extensions and modifications of MGs have been studied over the years, but standard MGs are still preferred for proofs because it is easier to establish a property in the simpler formalism and then show that it carries over to more elaborate versions. The same holds true for the feature-constraint equivalence proof. For this reason I only discuss standard MGs here and postpone discussion of linguistically more faithful variants until Sec. 4.1.

The structure-building process in MGs is not too different from Minimalist syntax. Suppose we want to build the DP *the car*. All we have to do is merge the lexical items (LIs) *the* and *car*, but this is allowed only if the two have matching features. Assume, then, that *the* has a *subcategorization feature* N^+ (also called *selector feature* in the MG literature), and *car* has a matching *category feature* N^-. This is expressed more succinctly by writing the :: N^+ and car :: N^-. The LI with the positive feature acts as the head, the one with the negative feature as its dependent. This means that headedness is an intrinsic lexical property in MGs (in contrast to Chomsky 2013). For example, the positive feature N^+ on *the* indicates that the determiner is the head of the projected phrase, rather than the argument *car* with the negative feature N^-.

The full DP is shown below, together with the equivalent *derivation tree*, which encodes how the expression was built from the LIs. Note that since the features were checked and deleted during the construction of the phrase structure tree, only the derivation tree lists them explicitly.



A slightly more complicated case is the construction of *John's car*. If one adopts the standard DP-analysis of possessive *'s*, then the possessive marker takes an NP-complement and a DP-specifier, so we have John :: D^-, car :: N^-, and *'s* must have the features D^+ and N^+. But how do we encode the fact that the NP must be the first argument and the DP the second one? MGs solve this by explicitly arranging features in the order they must be checked in, a solution that has been independently invoked for Minimalism in Müller 2010 and Georgi & Salzmann 2011, among others. Hence the possessive LI has the entry 's :: N^+ D^+, which allows it to participate in two Merge operations as depicted below.



One point we have ignored so far is that neither *the* nor *'s* were assigned any category feature, so strictly speaking they cannot head a phrase because they have no category to project. We fix this by adding the category feature D^- at the end of their respective feature specifications, yielding the entries the :: N^+ D^- and 's :: N^+ D^+ D^-. The category features have to follow the subcategorization features because the DP must not be selected by, say, a verb before it has merged with all its arguments.

Adding movement to this system is straight-forward. One simply introduces a new type of features that trigger Move instead of Merge and allows for these features to be checked as long as the LI with the positive feature c-commands the LI with the negative feature. Fig. 1 on page 7 shows a simplified phrase structure tree for *John's car Mary likes* and the corresponding derivation tree. Once again the derivation tree lists the features for all LIs, which are no longer present in the phrase structure tree. In addition, moving items remain in the position where they were base-merged since that is the point where they enter the derivation. Finally, Move nodes are unary branching and consequently denote only the point at which movement takes place rather than the participating arguments. This is possible because the actual movement steps can be deduced from the feature calculus. The lower Move node is licensed by the feature case$^+$ on T and thus must involve *Mary*, which has the matching feature case$^-$. The same reasoning applies for the higher Move node, with the relevant features being top$^+$ and top$^-$.

The reader may wonder what would happen in a more faithful analysis where *John's car* has a case feature that it must check in Spec,vP before it can move on to Spec,CP. In this case the derivation would at some point contain two LIs whose first unchecked feature is case$^-$. One is *Mary*, as before, the other one *'s*. This configuration is depicted in Fig. 2 on page 8. Now it is no longer obvious from the feature specification which one of the two phrases undergoes movement. There are many ways one could deal with this situation, but for the sake of a maximally simple model standard MGs completely ban such underspecified configurations via the so-called *Shortest Move Constraint* (SMC). The SMC — which has little to do with its Minimalist namesake — is controversial as it conflicts with standard linguistic assumptions. There are many technical reasons to stick with the SMC, and its impact on linguistic practice is much more marginal than one might think. In particular, the SMC can be relaxed and generalized to an extent where it no longer rules out the usual Minimalist analyses (cf. Sec. 4.1 of this paper and Sec. 2.3.4 of Graf 2013a). Most importantly for our purposes, the central technical results reported in this paper are independent of the SMC. Only one formal detail hinges on the SMC, which will be discussed in greater detail later on (page 31). At this point it only matters that Move in MGs is deterministic once the feature components of all LIs are known, which allows us to keep the current format of derivation trees.

Derivation trees are of particular interest in MGs because they act as a blueprint for building phrase structure trees. In fact, phrase structure trees are redundant in MGs as all necessary information is already encoded in the derivation trees. This makes it possible to define each MG via two components:

i. A finite set of feature-annotated LIs, from which one can compute the set of all well-formed derivations, also known as the grammar's *Minimalist derivation tree language* (MDTL).

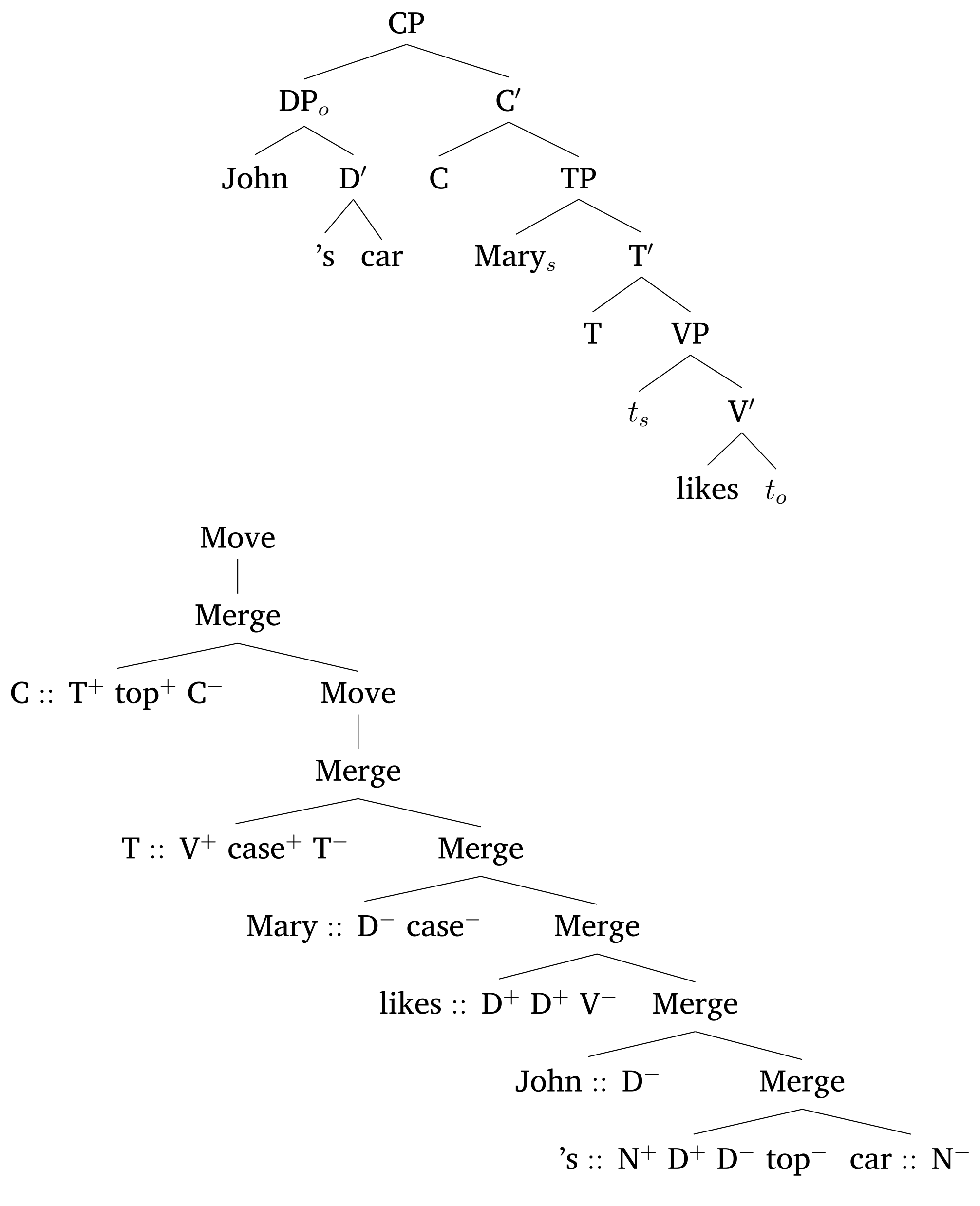


Figure 1 Phrase structure tree and MG derivation tree for *John’s car Mary likes*

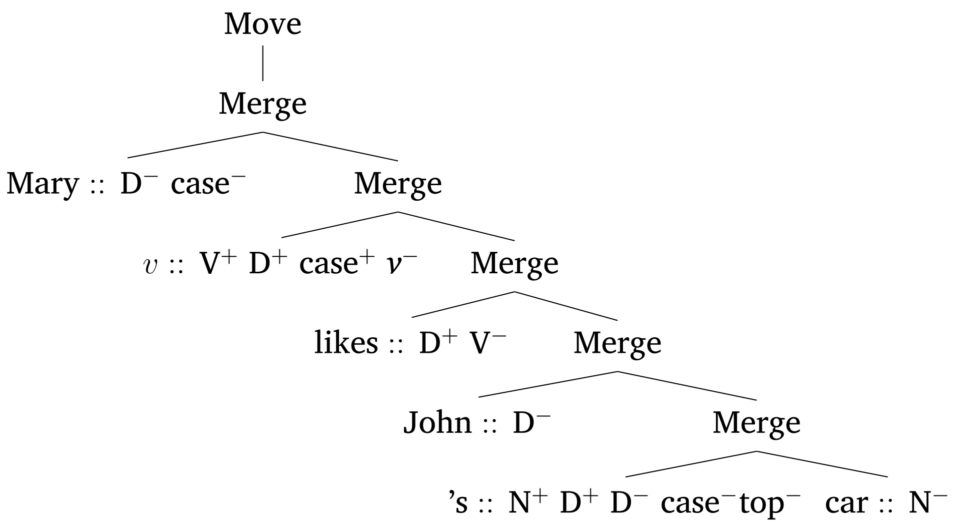


Figure 2 This derivation with two potential case movers violates the SMC.

ii. An *output mapping* (akin to Spell-Out) that translates each derivation tree into the corresponding output structure.

The output mapping can be freely chosen depending on what kind of structure is desired: bare phrase structure sets, X$'$ trees, prosodic trees, LF representations, strings, and so on. The choice of output structure is mostly irrelevant for the feature-constraint equivalence, which will be established at the level of derivation trees. It plays a minor role in Sec. 3.1 (translation from features to constraints) and 4.3 (the power of interface constraints). But as we will see there, the output structures commonly posited in syntax fit within the required mathematical parameters.

In sum, standard MGs are a highly feature-driven formalism where the feature calculus completely determines the set of well-formed derivation trees, and those in turn encode all the necessary information to build the desired output structures.

2.2 Formalizing constraints as logical formulas

Since MGs are purely feature-driven, they seemingly leave no room for syntactic constraints. Yet the syntactic literature has produced a number of constraints belong to a variety of types: at the very least constraints can apply to representations, operations, or derivations, and they can be local or non-local (cf. Müller & Sternefeld 2000). Irrespective of their exact

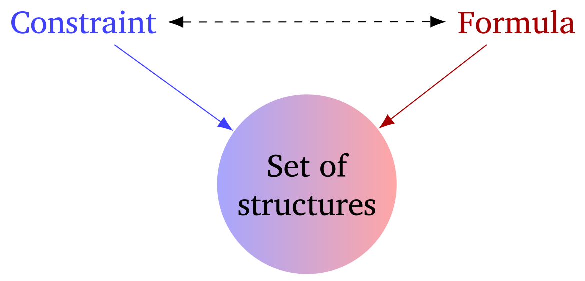


Figure 3 Constraints and logical formulas are in a one-to-one correspondence via the structures they license.

domain, though, all constraints share the property that they define a set of well-formed structures. Consequently, we may identify each constraint with a set of structures as its extension.

This is a coarse metric as very different constraints may define exactly the same set of structures, but it is abstract and general enough to be easy to work with. It is also remarkably close to the central idea of model theory in mathematics, where each logical formula is identified with the set of structures in which it is true. Daisy chaining these two definitions together, we can identify constraints with logical formulas through their extensions. So a constraint c is definable in logic L iff there is a formula ϕ of L such that ϕ is true in all structures, and only those, that are well-formed with respect to c (cf. Fig. 3).

This view of constraints has been explored in great detail in a subbranch of mathematical linguistics called *model-theoretic syntax* (Blackburn et al. 1993; Backofen et al. 1995; Morawietz & Cornell 1999; Cornell & Rogers 1998; Rogers 1998, 2003; Pullum 2007; Kepser 2008). The more powerful a logic, the more complex the constraints it can express. A particularly common logic in this domain is monadic second-order logic (MSO), which extends first-order logic with quantification over sets. MSO's main appeal is that it is powerful enough to allow for direct translation of syntactic constraints while still being well-behaved mathematically.

Let me give a brief illustration of MSO with a series of examples of increasing complexity. The reader need not pay close attention to every technical detail of the presentation; the important insight is that MSO is a very powerful language that can easily handle the kind of feature systems and structural notions encountered in syntactic constraints. It is for this very reason that virtually all syntactic constraints can be translated into MSO.

First we must pick a *signature*, i.e. a set of relational symbols to be used in the formulas. Usually the signature contains at least the following relations over trees (other structures such as bare phrase structure sets can be derived from underlying tree structures, see the end of this section).

- immediate dominance $\triangleleft$
- proper dominance $\triangleleft^+$
- reflexive dominance $\triangleleft^*$
- identity $\approx$

In addition, we may write $L(x)$ to indicate that node x has label L, and $X(x)$ if node x is contained in set X (the formal difference between the two is just that labels are part of the signature). Here X may be any suitably defined set of nodes, it need not correspond to a linguistically natural object such as a constituent. We also use the standard logical connectives $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), and $\leftrightarrow$ (iff). Note that we are not claiming that any of these are syntactic primitives. Rather, they are simply part of our MSO-based description language that we use to define certain constraints, irrespective of how they are actually specified in narrow syntax or at the interfaces.

As a first step, we verify that all the feature types that are considered suitable for Minimalism in Adger (2010) can be expressed in MSO. Adger distinguishes between privative, binary-valued, and multi-valued features. All three can be treated the same way in MSO, so I only cover multi-valued features here. Suppose the multi-valued feature Cat:N occurs only on the LIs *dog*, *car*, and *water* (it is a standard assumption that the lexicon, albeit unbounded, is always finite, so we can safely assume that there are only a finite number of LIs with a given feature). Rather than adding Cat:N to our signature, we define it through an equivalence such that a node x has this feature iff it is labeled *dog*, *car*, or *water*:

$$\mathsf{Cat{:}N}(x) \leftrightarrow \textit{dog}(x) \vee \textit{car}(x) \vee \textit{water}(x)$$

This formula will usually need to be refined to account for homonymy. For instance, *water* could have the category feature Cat:N or Cat:V depending on its structural position, but never both.

It is also important to keep in mind that this MSO treatment of features does not capture the technical differences between the three feature systems and, intuitively speaking, reduces them all to privative features. As a result, the distinctions between the three feature systems has nothing to do with the range of permissible feature values. What matters is not the feature values but the behavior of the whole feature calculus, e.g. that the absence of a feature in a privative system cannot trigger operations like a negative feature might in a binary-valued one. These things are encoded in separate MSO formulas that describe the feature calculus rather than the distribution of features.

Now that we have an MSO template for distributing features, let us see how we can formalize a simplified version of Principle A, stated over X′-style phrase structure trees.

Principle A (simplified) Every anaphor must be c-commanded within its binding domain by a DP with matching ϕ-features.

If all terms in this definition already had corresponding MSO-predicates, we could translate it almost word by word.

$$\forall x\Big[\textit{anaphor}(x) \rightarrow \\ \exists y\big[\textit{DP}(y) \wedge \textit{c-com}(y,x) \wedge \phi\textit{-match}(y,x) \wedge \exists X[\textit{bind-dom}(X,x) \wedge X(y)]\big]\Big]$$

This formula may look intimidating at first, but it only restates all the parts in the definition above. Let us go through it piece by piece and define predicates where needed.

The universal quantifier at the beginning of the formula ensures that the constraint holds for every node x that satisfies the antecedent condition, which in this case is the property of being an anaphor. We can define *anaphor* just like Cat:N above by listing all LIs that should be considered anaphors.

$$\textit{anaphor}(x) \leftrightarrow \textit{himself}(x) \vee \textit{herself}(x) \vee \textit{itself}(x) \vee \textit{themselves}(x)$$

The consequent of the implication starts with $\exists y$, which asserts the existence of some node. We then see that this node y is required to be labeled DP, c-command x, and ϕ-match x. All labels, including DP, are already part of our signature and thus need not be defined. Which nodes in particular may be labeled DP is regulated by other formulas that capture the rules of headedness and projection. I skip these rules for the sake of brevity. C-command and ϕ-matching, on the other hand, need to be supplied with suitable interpretations. But this is straight-forward. The former is specified via dominance in the usual fashion:

$$\textit{c-com}(x,y) \leftrightarrow \neg(x \triangleleft^* y) \wedge \forall z[z \triangleleft^+ x \rightarrow z \triangleleft^+ y]$$

In other words, x c-commands y iff x does not reflexively dominate y and every z that properly dominates x also properly dominates y.

For ϕ-matching, we assume that the set of ϕ-features corresponds to the predicates f_1, f_2, ..., f_n. There will usually be more predicates than features because Gen:m and Gen:f, for instance, are formalized as two distinct predicates.

$$\textit{\textbf{ϕ-match}}(x,y) \leftrightarrow (f_1(x) \leftrightarrow f_1(y)) \wedge (f_2(x) \leftrightarrow f_2(y)) \wedge \ldots \wedge (f_n(x) \leftrightarrow f_n(y))$$

This formula simply states that two nodes ϕ-match iff exactly the same ϕ-feature predicates are true of them.

A less strict definition of ϕ-matching may allow for the features of x to be a subset of y's features, or it may only require that x and y do not disagree on the values of the features they share. The former weakens equivalence to implication:

$$\begin{aligned}\textit{\textbf{ϕ-match}}(x,y) \leftrightarrow{} & \big((f_1(x) \rightarrow f_1(y)) \wedge (f_2(x) \rightarrow f_2(y)) \wedge \ldots \wedge (f_n(x) \rightarrow f_n(y))\big) \\ & \vee \big((f_1(y) \rightarrow f_1(x)) \wedge (f_2(y) \rightarrow f_2(x)) \wedge \ldots \wedge (f_n(y) \rightarrow f_n(x))\big)\end{aligned}$$

The latter changes each implication statement from $f_i(x) \rightarrow f_i(y)$ to $f_i(x) \rightarrow \neg f_{i+1}(y) \wedge \cdots \wedge \neg f_j(y)$, where f_i, f_{i+1}, ..., f_n are all the predicates for a specific feature, for example Gen:m, Gen:f, and Gen:n.

The last part of the MSO formula for Principle A is the restriction that the DP y must occur within a set of nodes that is the binding domain of the anaphor x. According to canonical binding theory, the binding domain of an anaphor is the smallest TP containing an abstract subject. For the sake of exposition, we will instead just use the smallest TP. So the set X should contain all nodes that are reflexively dominated by the closest dominating TP. Two predicates are defined for this purpose. First, *closest-TP*(x,y) picks out the closest TP node x that dominates y. Then *bind-dom*(X,y) constructs the set of all the nodes that are reflexively dominated by this TP node, which is the binding domain of y.

$$\textit{\textbf{closest-TP}}(x,y) \leftrightarrow \textit{\textbf{TP}}(x) \wedge x \triangleleft^* y \wedge \neg\exists z[\textit{\textbf{TP}}(z) \wedge x \triangleleft^+ z \wedge z \triangleleft^* y]$$

$$\textit{\textbf{bind-dom}}(X,y) \leftrightarrow \exists x[\textit{\textbf{closest-TP}}(x,y) \wedge \forall z[X(z) \leftrightarrow x \triangleleft^* z]]$$

Strictly speaking *bind-dom*(X,y) is a superset of the binding domain of y because it also includes TPs contained in the binding domain of y. This is irrelevant for Principle A, but a more accurate definition that excludes these TPs may be useful for Principle B.

$$\begin{aligned}\textit{\textbf{bind-dom}}(X,y) \leftrightarrow{} & \exists x[\textit{\textbf{closest-TP}}(x,y) \wedge \\ & \forall z[X(z) \leftrightarrow x \triangleleft^* z \wedge \neg\exists z'[\textit{\textbf{TP}}(z') \wedge x \triangleleft^+ z' \wedge z' \triangleleft^* z]]]\end{aligned}$$

With all predicates reduced to definitions that only use symbols from our original signature, we have a complete MSO-definition of (a simplified version of) Principle A.

As evidenced by this example, MSO — when equipped with predicates for dominance and all labels — is capable of defining very complex linguistic concepts. Features can be distributed across nodes in the tree; local and non-local dependencies are easily enforced; and locality domains are elegantly captured via the set quantification powers of MSO. Since these are the essential ingredients of syntactic constraints, virtually all of them are MSO-definable. This includes even transderivational ones (cf. Graf 2013a:Ch. 4), but the procedure is much more complicated in this case and cannot always be done by hand. The only constraints that are beyond the reach of MSO are semantically enriched principles like Rule I (Reinhart 1983; Heim 1998) with its identity of meaning requirement. But the details of the formulation can make quite a difference: while the strong version of Scope Economy (Fox 2000) — which bans QR if it does not affect the meaning of the sentence — is not MSO-definable, MSO can handle the weak version, which only bans those instances of QR that never affect the interpretation in any sentence. Overall, it is a safe bet that MSO can handle the overwhelming majority of syntactic constraints.

In addition to formalizing constraints, MSO can also define translations from one structure into another (Bloem & Engelfriet 2000; Engelfriet & Maneth 2003). The translations from MG derivation trees to phrase structures, bare phrase structure sets, and pronounced strings are all MSO-definable. The MG derivation tree languages themselves can also be fully specified via a small number of MSO-constraints. The interested reader is referred to Chapter 2 and Appendix B of Graf 2013a.

Recall from Sec. 2.1 that MGs are specified purely in terms of their derivation tree language and the output mapping, both of which are MSO-definable. Consequently, MGs can be described purely via MSO. This fact will be helpful in the construction of feature-free MGs in the next section.

3 Translating between features and constraints

With the technical preliminaries out of the way, I turn to the feature-constraint equivalence proof in Graf (2013a). I first explain how features can be replaced by constraints (3.1) and then look at the translation in the other direction (3.2). The translation from features to constraints as presented here improves on Graf (2013a) in that it also ensures that the mapping to output structures is feature-free. It depends on a greater number of technical properties than the opposite translation from constraints to features, though; details will be discussed in Sec. 3.1.3.

3.1 From features to constraints

As discussed in the previous section, the set of well-formed derivation trees of a given MG, which is called its MDTL, is definable in MSO — and so is the mapping from derivation trees to output structures. In fact, one can remove all features from the grammar without altering the set of licit derivation trees or the generated output structures. The result is a kind of feature-free MG where all work is done by constraints on derivation trees and suitably modified mappings to output structures.

3.1.1 Translation procedure

The basic strategy for making MGs feature-free is to first remove all features from the derivation trees and then use several mathematical tricks to design a new output mapping based on the old one. Given a standard MG G, two steps suffice to construct a feature-free MG G' that generates exactly the same derivation trees and output structures:

i. The feature-free MDTL of G' is the image of the MDTL of G under the relabeling *remove features*.
ii. The output mapping of G' is obtained by composing the inverse of *remove features* with the output mapping of G.

The process is depicted in Fig. 4. There we see that *remove features* keeps the derivation trees of the original MG exactly the same except that all LIs lose their feature annotation. These feature-free derivation trees can no longer be translated into phrase structure trees via the standard output mapping of MGs because the latter assumes that head-argument relations and movement dependencies are explicitly encoded by features. However, one can construct a new mapping that behaves exactly like the standard output mapping after all features have been reinserted. So the switch to feature-free MGs requires only the removal of all features from the derivation trees and a modification of the output mapping.

The reader may doubt the feasibility and adequacy of this feature removal procedure, and rightfully so. It is far from obvious that the feature-free MDTL and the new mapping to output structures are well-defined objects, let alone computable ones. More importantly, removing features from

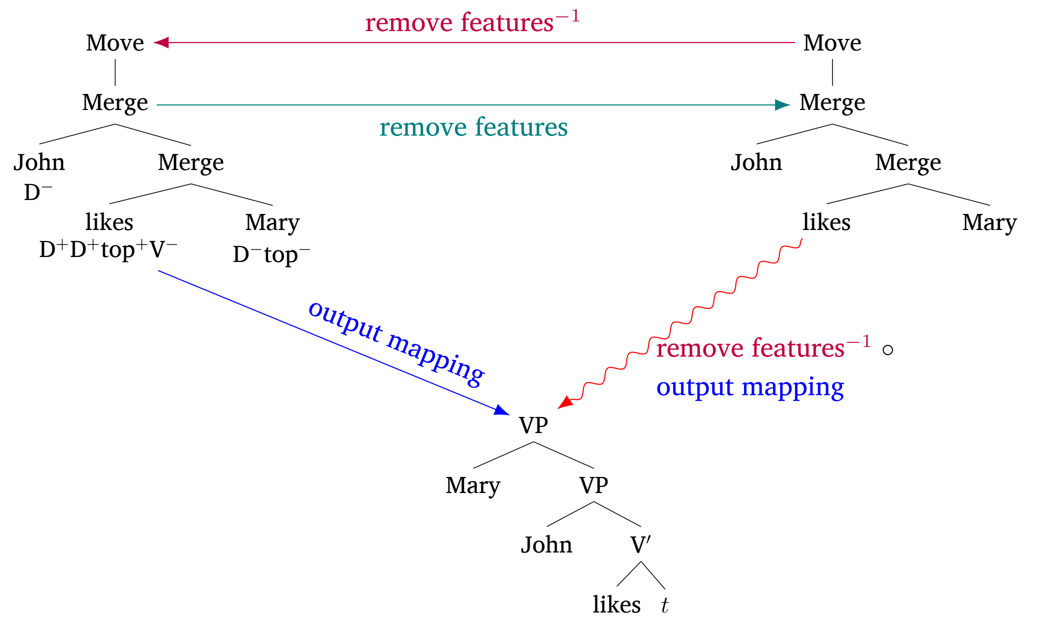


Figure 4 General procedure for constructing a feature-free MG

derivation trees only to secretly reinstate them in the output mapping seems like little more than a technical trick. How does it take us closer to the goal of completely removing features from MGs? I will address the second issue first since its answer leads towards an understanding of (relational) composition that is a prerequisite for tackling the more mathematical questions of computability.

3.1.2 Composition and the status of intermediate features

In mathematical terms, the output mapping of feature-free MGs is defined as the composition of the inverse of *remove features* with the standard output mapping, also written *remove features*$^{-1}$ $\circ$ *output mapping*. Let us break this expression into its subparts, starting with *remove features* and its inverse.

A binary relation R can be viewed as a set of pairs $\langle a, b\rangle$ such that a is related to b by R. The inverse of R is denoted R^{-1} and contains $\langle a, b\rangle$ iff R contains $\langle b, a\rangle$. In other words, R^{-1} simply switches the direction of R — output becomes input and input becomes output. *Remove features* is a binary relation that maps each feature-annotated derivation tree to the equivalent derivation tree without features. Consequently, *remove features*$^{-1}$ relates each feature-free derivation tree to the feature-annotated derivation trees it can be obtained from via *remove features*. For most grammars *remove features*$^{-1}$ is one-to-many as different feature-annotated derivation trees may be related to the same feature-free derivation tree. For example, a variant of the feature-annotated derivation tree in Fig. 4 where the topicalization feature is instantiated on *John* rather than on *Mary* would still be mapped to the same feature-free derivation tree by *remove features*.

Given two relations R and S, their composition $R \circ S$ contains a pair $\langle a, c \rangle$ iff R contains $\langle a, b \rangle$ and S contains $\langle b, c \rangle$. Intuitively, one may think of $R \circ S$ as the result of feeding the output of R into S. But this analogy is not quite correct because it suggests that $R \circ S$ is computed by first running R and then S. This is not necessarily the case. Consider the following example. Let R map each integer n to $n + 1$, whereas S maps each integer n to $n - 1$. In this case, $R \circ S$ maps every integer to itself. Now of course one could compute this relation by first adding 1 to the input and the subtracting 1 from it, but a much more efficient solution is to simply return the input as the output. In this case, the output of R no longer exists in the computation of $R \circ S$. So while R and S might be convenient for specifying $R \circ S$, they are not necessarily part of computing this relation.

For this reason, *remove features*$^{-1}$ $\circ$ *output mapping* does not necessarily involve features at all. Even though features appear in the output of *remove features*$^{-1}$ they might not be part of the computation of the composed output mapping. The question, then, is whether there is a way of computing the composed output mapping without ever computing the output of *remove features*. The answer is affirmative, and it leads us back to the original question about the computability of the feature removal procedure.

3.1.3 The complexity of feature-free MGs

At the end of the introduction to MSO in Sec. 2.2 it was mentioned that every MDTL is MSO-definable and that the same holds for the mapping from derivation trees to output structures. These two properties jointly ensure that *remove features*$^{-1}$$\circ$*output mapping* is also MSO-definable and thus can be computed directly without first constructing the output of *remove features*$^{-1}$. The proof is a simple corollary of several well-known properties (see Engelfriet 1975, Engelfriet & Maneth 2003, Graf 2013a, and references therein).

i. MDTLs are MSO-definable.
ii. Since *remove features* is a relabeling, it preserves MSO-definability.
iii. Since *remove features* is a relabeling of an MSO-definable tree language, its inverse is MSO-definable.

iv. The output mapping is MSO-definable for the usual choices of output structures (bare phrase structure trees/sets, X′ trees, multi-dominance trees, strings).
v. Given two MSO-definable transductions, their composition is also MSO-definable.
vi. It follows that *remove features*$^{-1}$ $\circ$ *output mapping* is MSO-definable.

This confirms that the output mapping can be computed directly from feature-free derivations without positing features at some point during the process. The details for how one might construct an algorithm for this mapping will be skipped here as they are highly technical in nature and hardly insightful. For our purposes, it suffices to know that it can be done fully automatically with no human ingenuity required. Consequently, the right algorithm is guaranteed to be constructible.

An interesting property of the proof above is that it is valid as long as (i) and (iv) are satisfied. In other words, the feature removal procedure can be used for any feature-based formalism where an MSO-definable tree language is translated into a set of output structures by an MSO-definable mapping. This isn't a distinguishing property of MGs, it also holds of Tree Adjoining Grammar (Mönnich 2012), GPSG (Rogers 1996, 1997), and certain variants of Dependency Grammar (Boston et al. 2010). Grammar formalisms with recursive feature systems like HPSG and LFG, on the other hand, do not fit these requirements. The same holds for MGs without the SMC, which blocks two LIs from being eligible to move to the same position (cf. the discussion on page 6). MGs without the SMC still can be made feature-free but the process requires a more powerful logic than MSO, as will be briefly discussed in Sec. 4.1.

3.1.4 An intuitive view of feature removal in MSO

Considering how features are implemented in MSO, it is actually not surprising that they can be removed from Minimalist derivation trees. An MSO formalization of MGs could start out by defining a predicate for each feature string. If x is the LI likes :: $\mathrm{D^+D^+V^-}$, one would say that the predicate $\mathrm{D^+D^+top^+V^-}$ holds of x. But remember that such feature predicates are not part of the MSO signature, they are derived from more basic predicates. Just like we previously defined Cat:N(x) as a shorthand for *dog*$(x) \vee$ *car*$(x) \vee$ *water*(x), we can define $\mathrm{D^+D^+V^-}$ as a disjunction of all LIs that may have this feature string.

$$\mathrm{D^+D^+V^-}(x) \leftrightarrow \textit{like}(x) \vee \textit{kick}(x) \vee \textit{hate}(x) \vee \textit{devour}(x) \vee \ldots$$

As soon as MGs are defined in this fashion, they are already feature-free. Whenever features are mentioned in some formula, that is merely a notational shorthand for disjunctions of the kind above. Hence the MSO formulas never speak about features as such, just about specific LIs. Quite simply, features can be done away with because the computational apparatus can reason with lists of LIs instead.

One complicating factor for this neat picture is lexical ambiguity. Assuming that subjects are VP-arguments rather than vP-arguments, the verb *eat* has the feature string $\mathrm{D}^{+}\mathrm{D}^{+}\mathrm{V}^{-}$ when used transitively and $\mathrm{D}^{+}\mathrm{V}^{-}$ when used intransitively. Similarly, *water* has the feature strings N^{-} as a noun and $\mathrm{D}^{+}\mathrm{V}^{-}$ as a verb. The definition of feature strings sketched above entails that multiple feature predicates simultaneously hold of every ambiguous LI such as *eat* or *water*. In that case it is not always safe to replace, say, the predicate N^{+} by a disjunction containing *water* because the specific instance of *water* in the derivation may actually be a verb. Additional technical tricks are required to correctly handle such cases. But since the MSO formulas for a feature-free MDTL are computed by an algorithm that operates in a more sophisticated manner than the approach sketched here, this does not matter for the validity of the feature removal result.

It is interesting to note, though, that lexical ambiguity is much less of an issue if one takes seriously Chomsky's (1995b) assertion that an LI is a triple consisting of a phonetic exponent, a set of formal features, and a semantic representation. Given an LI's phonetic exponent and semantic representation (e.g. a lambda term), it is much easier to determine the LI's feature string. The only source of uncertainty in this case is the distribution of movement features. If those can also be predicted, e.g. from other properties of the derivation, the construction of a feature-free MSO formalization of MGs would be remarkably simple. Grammars that make it possible to deterministically infer the correct feature strings for LIs also have great advantages for learnability and parsing (Kanazawa 1998; Kobele et al. 2002; Hale & Stabler 2005). So there are independent reasons to assume that lexical feature assignment is heavily restricted in natural languages, which in turn would further simplify the conversion to feature-free MGs.

3.2 From constraints to features

It has now been established that features can be completely replaced by constraints. The role features play in picking out well-formed derivation trees and determining their spell out is sufficiently restricted that it can be handled by MSO-definable constraints and an MSO-definable output mapping instead. More succinctly, the information encoded in features can be "delexicalized". Of course the opposite is also possible: constraints can be lexicalized by reencoding them as feature valuations of LIs. This insight is pretty much part of linguistic folklore, and has been extensively used in GPSG. But this section presents a much stronger and much more surprising finding.

MSO-Constraints ≡ Subcategorization The class of constraints that are definable in MSO is exactly the class of constraints that can be lexicalized via subcategorization requirements.

As a result, even highly complex constraints of contemporary Minimalism can be reencoded via the most basic mechanism shared by virtually all syntactic formalisms: selection/subcategorization.

3.2.1 From constraints to tree automata

Once again our strategy hinges on a formal translation procedure, but this time the procedure converts an MG with constraints over derivation trees into one without these constraints (this will be further generalized to interface constraints in Sec. 4.3). The translation hinges on a long-known equivalence between MSO formulas and *bottom-up tree automata* (Büchi 1960; Doner 1970; Thatcher & Wright 1968). This equivalence is used to decompose the MSO-constraint into a collection of local constraints between heads and arguments which is then reencoded as selectional restrictions of the feature calculus (Kobele 2011; Graf 2011; Graf 2013a:Chap. 3).[1] The spirit of the translation is similar to GPSG's slash feature percolation for movement

[1] An anonymous reviewer wonders whether this decomposition is at odds with well-known weak generative capacity results. Natural languages, when viewed as string languages, are at least Tree-Adjoining languages (Culy 1985; Huybregts 1984; Shieber 1985), and possibly even more powerful (Kobele 2006; Michaelis & Kracht 1997; Radzinski 1991). Bottom-up tree automata, on the other hand, can be viewed as an extension of context-free grammars that still generates only context-free languages. Those, in turn, form a proper subclass of the Tree-Adjoining languages. How then can every syntactic constraint be reduced to a bottom-up tree automaton?

The answer goes back to the end of Sec. 3.1.3, where it was pointed out that grammar formalisms can be decomposed into two MSO-components: the set of well-formed derivation trees, and a mapping from derivation trees to the desired output structure. Among other things, this implies that syntax is actually context-free at the level of derivation trees. It is only the mapping to the output structures that pushes the complexity into a more complex class of string languages. So bottom-up tree automata can encode all syntactic constraints because syntax is context-free over derivation trees. Section 4.3 explains how interface constraints fit into this picture.

dependencies, but is general enough to work for all MSO-constraints and with every formalism that assumes subcategorization requirements.

The conversion from MSO formulas to an equivalent bottom-up tree automaton is well known in formal language theory, and the reader is referred to Morawietz (2003:64–72) for a detailed discussion. That said, a brief look at bottom-up tree automata will help the reader understand why MSO-definable constraints are reducible to selectional restrictions.

A bottom-up tree automaton is a mechanism to verify the well-formedness of a tree in a purely local fashion. The automaton has a set of *states*, which are arbitrary symbols without any specific meaning attached to them. Crucially, the state set of bottom-up tree automata must be finite; this restriction keeps their power in check, and it is also the linchpin on which the reduction of MSO-constraints to selection rests. The states are used by the automaton to mark-up nodes in the tree. More precisely, each node is assigned a state depending on its label and the states assigned to its daughters. A tree is well-formed iff its root has been assigned a distinguished state called a *final state*. Readers familiar with automata theory will notice that bottom-up tree automata are the natural generalization of finite-state automata from strings to trees.

One may think of bottom-up tree automata as a specific kind of phrase structure grammar where the state symbols act as an ancillary, invisible alphabet.[2] Consider for instance the (linguistically implausible) constraint that the symbol A must occur an odd number of times in each tree. For the sake of simplicity, assume that all trees are strictly binary branching. Then the corresponding tree automaton can be specified via a set of rewrite rules.

$$
\begin{array}{lllclll}
\mathrm{X}^e & \to & \_^o\ \_^o & \qquad & \mathrm{A}^o & \to & \_^o\ \_^o \\
\mathrm{X}^o & \to & \_^o\ \_^e & & \mathrm{A}^e & \to & \_^o\ \_^e \\
\mathrm{X}^o & \to & \_^e\ \_^o & & \mathrm{A}^e & \to & \_^e\ \_^o \\
\mathrm{X}^e & \to & \_^e\ \_^e & & \mathrm{A}^o & \to & \_^e\ \_^e \\
\mathrm{X}^e & \to & & & \mathrm{A}^o & \to &
\end{array}
$$

Here we use X as a placeholder for any label that is not A and the underscore for all possible node labels. The rewrite rules without a right-hand side are used for leaf nodes. The superscripts o and e represent the two states of the automaton, which we may think of as abbreviations for odd and even. The rule $\mathrm{X}^e \to \_^o\ \_^o$, for example, postulates that whenever both daughters have

[2] In contrast to finite-state automata, bottom-up tree automata have no appealing representation in terms of graphs. Since multiple states may form the input for a single state transition, one would need hypergraphs with labeled hyperedges to depict these transitions. Such graphs would quickly become indecipherable for all but the smallest automata.

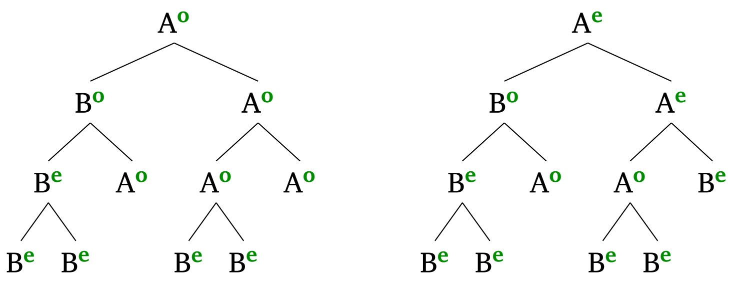


Figure 5 State assignments of the odd/even automaton for licit and illicit trees

state o and the node is not labeled A, it is assigned state e. On the other hand $\text{A}^o \rightarrow \_^o\ \_^o$ tells us that if the node were labeled A, it would get state o instead. The strategy used by the automaton is to assign o to those nodes that are the root of a subtree containing an odd number of As. So if both daughters contain an odd number of As, an even number of As is properly dominated by the current node. Depending on the label of the current node, the whole subtree thus contains an odd or an even number of nodes labeled A. As long as o is the only final state, the automaton will accept only trees with an odd number of As; see Fig. 5 for two examples.

3.2.2 An extended example: Principle A

While the automaton above provides a simple illustration of the technical machinery, a linguistically more faithful example gives a better impression of how information has to be carefully passed around between states. Principle A, although a fairly simple constraint, poses several challenges in this respect.

As in Sec. 2.2, only a simplified variant of Principle A will be considered for the sake of exposition. Assume, then, that Principle A applies over strictly binary-branching phrase structure trees and requires every anaphor to be c-commanded (at S-structure) by a DP with matching ϕ-features. In order to accomplish this, the desired automaton needs states that keep track of whether a reflexive has been encountered, and what its ϕ-features look like. For English, this requires four leaf node rules:

$$\begin{aligned} \text{himself}^{a,m,s} &\rightarrow \qquad\qquad & \text{themselves}^{a,p} &\rightarrow \\ \text{herself}^{a,f,s} &\rightarrow & & \\ \text{itself}^{a,n,s} &\rightarrow & & \end{aligned}$$

A state like a, m, s indicates that an anaphor has been encountered that is masculine and singular, whereas a, p represents a plural anaphor. In languages where plural anaphors also display gender differences, one would have states such as a, m, p instead. For the sake of simplicity, I just write a, ϕ whenever the specific ϕ-feature values do not matter.

But assigning states to the anaphors is not enough, this information also has to be passed up to the subtrees that contain these anaphors. Let X be a shorthand for any non-terminal label except TP (which will be treated in a special way later on). Furthermore, $*$ is a state that is assigned to all elements that are not involved in anaphor licensing, e.g. verbs. A single rule suffices to pass on this state to interior nodes, e.g. VPs that contain neither DPs nor anaphors.

$$\mathrm{X}^{*} \quad \rightarrow \quad \_^{*}\ \_^{*}$$

Two very similar rules are used to percolate anaphor states.

$$\mathrm{X}^{a,\phi} \quad \rightarrow \quad \_^{a,\phi}\ \_^{*} \qquad\qquad \mathrm{X}^{a,\phi} \quad \rightarrow \quad \_^{*}\ \_^{a,\phi}$$

Since a subtree may contain multiple anaphors, we also need to be able to percolate multiple anaphor states at once. Given two feature bundles ϕ and ϕ', let $\phi \oplus \phi' = \phi, \phi'$ if $\phi \neq \phi'$, and just ϕ otherwise. The $\oplus$ operator is generalized to sequences $\phi_1, \ldots, \phi_n$ and $\phi'_1, \ldots, \phi'_n$ so that it returns the sequence of all feature bundles in these two sequences, but with duplicates removed. Since there's only finitely many distinct ϕ-feature bundles, the $\oplus$ operator can only produce finitely many distinct sequences of feature bundles. Consequently, the following template encodes a finite number of automaton rules.

$$\mathrm{X}^{a,\phi} \quad \rightarrow \quad \_^{a,\phi_1,\ldots,\phi_m}\ \_^{a,\phi'_1,\ldots,\phi'_n} \quad \text{where } \phi = \phi_1, \ldots, \phi_m \oplus \phi'_1, \ldots, \phi'_n$$

Intuitively, the states now store a list of all the ϕ-feature bundles that must be matched by c-commanding DPs.

Licensing of one or several anaphors by a DP amounts to removal of the corresponding ϕ-feature bundles from the projected state. Suppose that each DP is already annotated with its ϕ-feature bundle as its state. The mechanism for percolating these states from, say, *John* or *women* to the DPs they head is mostly analogous to the procedure for the anaphor states. I leave it as an exercise to the reader (who should take care that the correct ϕ-feature state gets percolated in phrases like *Mary's parents*). In order to save ink, the operator $\ominus$ handles the removal of ϕ-features: $\phi_1, \ldots \phi_n \ominus$

$\phi'_1, \ldots, \phi'_n$ contains all elements of $\phi_1, \ldots, \phi_n$ that do no match any elements of $\phi'_1, \ldots, \phi'_n$.

$$\begin{array}{rcll}
\mathrm{X}^{a,\phi} & \to & \_^{\phi_1,\ldots,\phi_m}\ \_^{a,\phi'_1,\ldots,\phi'_n} & \text{where } \phi = \phi'_1, \ldots, \phi'_n \ominus \phi_1, \ldots, \phi_m \\
\mathrm{X}^{a,\phi} & \to & \_^{a,\phi_1,\ldots,\phi_m}\ \_^{\phi'_1,\ldots,\phi'_n} & \text{where } \phi = \phi_1, \ldots, \phi_m \ominus \phi'_1, \ldots, \phi'_n \\
\mathrm{X}^{*} & \to & \_^{\phi_1,\ldots,\phi_m}\ \_^{a,\phi'_1,\ldots,\phi'_n} & \text{where } \phi'_1, \ldots, \phi'_n \ominus \phi_1, \ldots, \phi_m \text{ is empty} \\
\mathrm{X}^{*} & \to & \_^{a,\phi_1,\ldots,\phi_m}\ \_^{\phi'_1,\ldots,\phi'_n} & \text{where } \phi_1, \ldots, \phi_m \ominus \phi'_1, \ldots, \phi'_n \text{ is empty}
\end{array}$$

The last two rules are crucial as they ensure that a subtree that only contains licensed anaphors has the same status with respect to Principle A as a subtree without any anaphors. Therefore both receive the state $*$.

Remember that TP is not covered by the placeholder X, it still requires some rules to be defined. These rules must be carefully chosen so that TP receives a state only if it contains no unlicensed anaphors. In other words, the only valid state for TP is $*$. We also make $*$ the only final state of the automaton so that only trees without unlicensed anaphors are accepted.

$$\mathrm{TP}^{a,*} \to \_^{*}\ \_^{*} \qquad \mathrm{TP}^{a,*} \to \_^{\phi}\ \_^{*} \qquad \mathrm{TP}^{a,*} \to \_^{*}\ \_^{\phi}$$

$$\begin{array}{rcll}
\mathrm{TP}^{a,*} & \to & \_^{\phi_1,\ldots,\phi_m}\ \_^{a,\phi'_1,\ldots,\phi'_n} & \text{where } \phi'_1, \ldots, \phi'_n \ominus \phi_1, \ldots, \phi_m \text{ is empty} \\
\mathrm{TP}^{a,*} & \to & \_^{a,\phi_1,\ldots,\phi_m}\ \_^{\phi'_1,\ldots,\phi'_n} & \text{where } \phi_1, \ldots, \phi_m \ominus \phi'_1, \ldots, \phi'_n \text{ is empty}
\end{array}$$

The inability to assign a state to TP unless all anaphors it contains are licensed takes care of two properties at once. Since anaphor states cannot percolate out of TP, all anaphors must be licensed within the smallest containing TP. And if a TP contains unlicensed anaphors, it cannot receive a state and thus the whole tree is rejected (cf. Fig. 6).

As the reader can see, writing tree automata by hand is a laborious process for even the simplest constraints. The treatment here omitted several bookkeeping rules to percolate the states for licensing DPs, and we completely omitted complicating factors such as anaphors contained within DPs or A′movement feeding Principle A. But fortunately the automata need not be defined manually, there is an algorithm to automatically convert MSO-constraints into equivalent bottom-up tree automata. The example shows, however, that the automata quickly approach surprising degrees of complexity. I return to this point at the end of Sec. 5.2 (page 37), where I suggest that the constraints that are analyzed in terms of features in the literature are those whose corresponding automata are fairly simple.

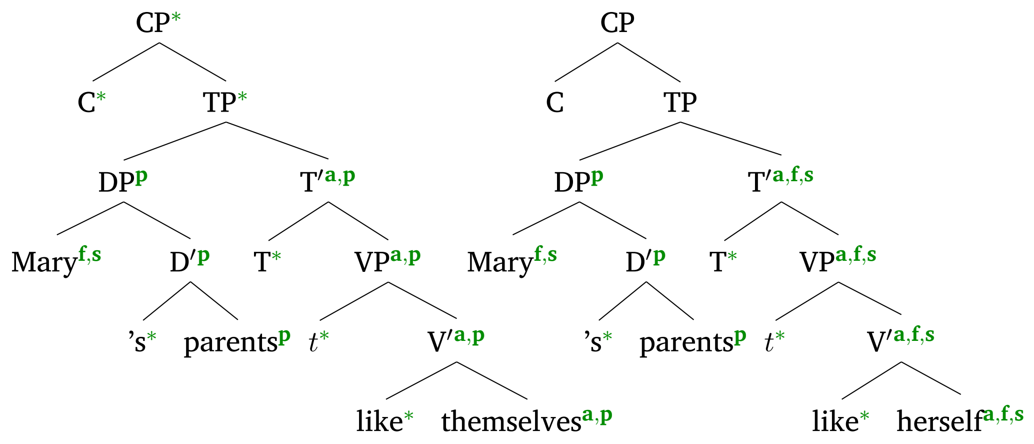


Figure 6 The well-formed tree on the left is assigned a final state by the Principle A automaton, whereas state assignment halts at TP in the illicit one on the right.

3.2.3 From automata to refined subcategorization requirements

After an MSO-constraint has been converted into a bottom-up tree automaton, it can be used to induce a state assignment over the set of derivation trees. If the MSO-constraint was originally stated over phrase structure trees, as our example of Principle A, it must first be converted back to a derivation tree. This is always possible, and fairly easy in the case of MGs. A full example is given in Sec. 4.3 during the discussion of interface constraints.

With the help of the constructed tree automaton, every derivation tree has a state attached to each one of its nodes. Most of these states are actually redundant, and this is what allows for conversion of the automaton into subcategorization requirements. In a grammar formalism like MGs, the status of a phrase depends only on its head and the shape of the arguments selected by that head. The shape of an argument, to the extent that it matters for a given MSO-constraint, is fully represented by the state assigned to its root node. For instance, the states o and e of the first example automaton do not encode the full structure of a given subtree, but they do keep track of the information that matters for the constraint — namely whether the number of nodes in the subtree with label A is even or odd. So if we have a head X selecting a YP and ZP and the corresponding maximal projections are actually labeled XP^p, YP^q, and ZP^r after state assignment, then we can reinterpret this as a statement that X selects a q-YP and an r-ZP to produce a p-XP (which implies that X is actually p-X). This is tantamount to treating the automaton's state assignments as a refinement of the grammar's parts of speech, which is illustrated in Fig. 7.

Category refinement can greatly increase the size of the lexicon but cannot render it infinite, which is an essential requirement for all MGs. Consider an LI that selects two arguments, e.g. like :: D^{+} D^{+} V^{-}. The odd/even automaton from our first example can assign two different states to each DP and the VP itself, which means that category refinement might split the single LI into eight minimally different variants.

like :: D^{o+} D^{o+} V^{o-}	like :: D^{e+} D^{o+} V^{o-}
like :: D^{o+} D^{o+} V^{e-}	like :: D^{e+} D^{o+} V^{e-}
like :: D^{o+} D^{e+} V^{o-}	like :: D^{e+} D^{e+} V^{o-}
like :: D^{o+} D^{e+} V^{e-}	like :: D^{e+} D^{e+} V^{e-}

Of course some of these state assignments might never be realized in a well-formed derivation. In the case at hand, we can eliminate the first column of LIs since no node projected by *like* is labeled A, wherefore the number of As depends only on the shape of the arguments. If the grammar contains not a single LI labeled A, then the automaton will assign state e to every node so that like :: D^{+} D^{+} V^{-} would be refined into the single entry like :: D^{e+} D^{e+} V^{e-}. Irrespective of how big the actual blow-up, each LI will be refined into only a finite number of LIs (the upper bound is $Q \times (k+1)$, where Q is the number of states of the automaton and k the number of subcategorization features on the LI).

3.2.4 Why category refinement works for MSO-constraints

Expressing MSO-constraints via category refinement is possible only because of MSO's equivalence to bottom-up tree automata and their restriction to a finite number of distinct states. In fact, one can show that MSO-constraints are the most powerful constraints that can still be pushed into the feature system via category refinement. The reader might be wondering, though, how MSO-constraints on the one hand can be powerful enough to accommodate virtually all constraints from the syntactic literature but on the other hand are restricted enough that they can be broken down into a finite set of states with a particular assignment pattern resembling simple phrase structure rules. The answer reveals a crucial property of linguistic constraints: even though they may involve very complex structural configurations and hold across much larger distances than selection, only a finite amount of information about the tree ever needs to be kept in memory.

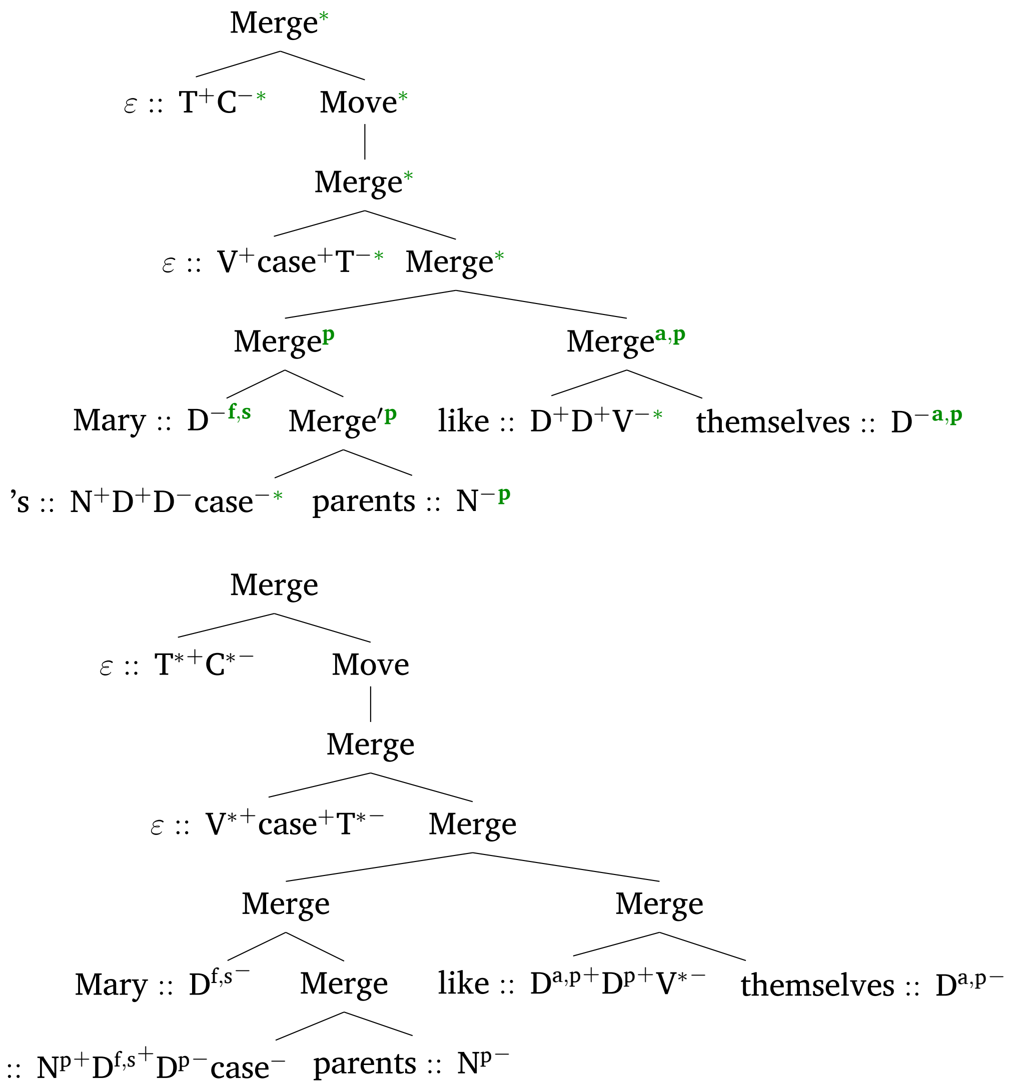


Figure 7 Converting a state assignment for a derivational version of Principle A into refined categories

For instance, Principle A requires much more limited knowledge of the shape of the tree than one might expect. As discussed in Sec. 3.2.2, a bottom-up tree automaton only needs to remember that I) an anaphor with a specific set of ϕ-features has been encountered, and II) it has not reached the root of the anaphor's binding domain yet. If no c-commanding, ϕ-matching DP is found before the automaton leaves the binding domain, the tree is rejected. But if a c-commanding DP is found, the automaton need not even keep track of that. Instead, it simply forgets about the anaphor altogether — its dependency has been satisfied, and thus there is no reason to pay further attention to it. In a fully worked out grammar, where Principle A interacts with other constraints, things are not quite as straight-forward, but the ability to forget about parts of the tree as soon as they can no longer participate in any dependencies still holds.

What makes syntactic constraints so well-behaved, then, is that one can always make do with a finitely bounded amount of memory if one discards all information as soon as possible. In a certain sense this can be likened to a radical version of phases (Chomsky 2001, 2004) where subtrees are closed off at every point in the derivation and even those subtrees that still participate in dependencies have been reduced to a simple node with a record of their licensing properties (this strategy is also employed in the definition of MGs in Stabler & Keenan 2003, and Stabler 2011 shows that it is unaffected by the addition of phases in Chomsky's original sense). These records are then passed along from node to node, with specific licensing properties being added or removed as the structural configurations change. This is exactly what bottom-up tree automata do with their state assignments and thereby reduce non-local licensing configurations into a local transferral of licensing properties that can be emulated by subcategorization.

4 Linguistic applicability of the formal results

Up to this point we were only concerned with establishing a mathematical theorem: a specific type of feature system is expressively equivalent to a specific class of constraints. As so often in mathematical linguistics, though, the real question is not whether a theorem holds but how it relates to the generative enterprise. A result that hinges on dubious assumptions is of little relevance to linguistic research. The feature-constraint equivalence is embedded in a broad network of assumptions that can be split into three macro groups:

i. Minimalist grammars provide a suitable model of syntax, in particular, the syntactic feature calculus.
ii. The class of constraints is identified with the set of MSO formulas over MG derivation trees.
iii. Equivalence is defined as the ability to license exactly the same sets of derivations and output structures.

In this section, I argue that these differences between the formal model and standard linguistic practice are not pronounced enough to invalidate the equivalence result. Minimalist grammars are sufficiently close to Minimalist syntax (4.1), there are no empirically sound reasons to reject category refinement (4.2), comparable results can be obtained for interface constraints (4.3), and the focus on generative capacity is not a severe limitation (4.4). The next section then continues the thread by exploring its ramifications for generative syntax.

4.1 Minimalist Grammars and Minimalist syntax

Standard MGs are in many respects a stripped down version of early Minimalism as defined in Chomsky (1995a). They do not incorporate later additions like Agree (Chomsky 2000), phases (Chomsky 2001, 2004), sidewards movement (Hornstein 2001; Nunes 2004), Late Merge (Lebeaux 1988; Ochi 1999; Stepanov 2001; Takahashi & Hulsey 2009), or even adjunction. Yet MGs also stipulate details of the feature calculus that are not shared by the Minimalist mainstream. The divide between interpretable and uninterpretable features (Chomsky 1995a) or valued and unvalued features (Chomsky 2001) is replaced by a positive/negative bifurcation that behaves more like a system where all features are uninterpretable and thus must be deleted. There is also an explicit divide between Merge features and Move features, and features are ordered to reflect the derivational sequence in which they are checked. Finally, the SMC puts severe restrictions on movement by rejecting all derivations in which two LIs are both eligible to move to a specific target site. In sum, standard MGs lack important aspects of modern Minimalism while also including a non-trivial amount of "non-Minimalist" machinery.

The crucial point, though, is that the absence of modern Minimalist trappings holds only of standard MGs. Since their inception 20 years ago in Stabler (1997), MGs have been extended and generalized in various directions. The literature is vast, and only a small sample can be given here. Agree can be implemented with the hypothetical reasoning mechanism of

Kobele (2010), adjunction has been formalized in various ways (Frey & Gärtner 2002; Fowlie 2013; Hunter 2015; see also Graf 2014c), head movement was studied in Kobele (2002) and Stabler (2003), and sideward movement in Stabler (2006). Graf (2012b) even defines a general system for adding new movement types to MGs without altering certain formal properties, and Graf (2014b) shows that this system can also be used to incorporate Late Merge. What all these variants have in common is that they preserve two core properties of MGs: both their MDTLs and the mapping to output structures are MSO-definable. But as we saw in Sec. 3.1 these are exactly the properties that allow for features to be replaced by MSO-constraints. As a consequence, features are inessential not only in standard MGs but in any grammar formalism that stays within the realm of MSO-definability.

The MG feature calculus has also been altered in various ways. Three modifications are particularly noteworthy. The first one is the addition of interpretable (also called *persistent*) features in Kobele (2006) and Stabler (2011). These features may participate in an unbounded number of checking relations. Just like standard MG features and the related adjunction feature calculus of Frey & Gärtner (2002), these features can be handled via MSO. The other two theorems about MG features can be considered part of the MG folklore, but had not been published until Graf (2013a).

Following up on Chomsky's assertion that Move is just a special case of Merge, Graf (2013a:2.3.2) explains how the distinction between Merge and Move can safely be eliminated from MGs (see also Hunter 2011). Remember that derivation trees are the central data structure of MGs, and over those trees the difference between Merge and Move boils down to whether an operation checks Merge features or Move features. Without a distinction between Merge and Move features, then, Merge and Move behave the same in derivation trees and differ only in how they affect the output structure.

Without the Merge/Move distinction, MG features are ambiguous as to which operation they trigger in the derivation. Instead of a four-way split between positive and negative Merge and Move features, there is only an opposition between positive and negative features. However, one can still uniquely identify licensee features, i.e. the features that allow an LI and the phrase it projects to undergo movement. Since an LI may only move after it has entered the derivation via selection, licensee features must always follow the category feature. The category feature, in turn, must be the first negative feature, right after all positive features. This is because an LI can be selected only after all its argument positions and specifiers have been filled, which is tantamount to checking all its positive features. Consequently, the licensee features of an LI are exactly those after the first negative feature. Where necessary, one can then reinstantiate the strict division between Merge and Move features by using different feature names for category features and licensee features. Due to the symmetry of feature checking, this also bifurcates the positive features into selector and licensor features. So the split between Merge and Move features is redundant because it can be implicitly encoded via the feature orderings — merger of an already merged element is necessarily re-merge.

In the same section, Graf (2013a) also shows that feature order is irrelevant as long as heads can distinguish between complements and specifiers. First, every MG can be brought into a strongly equivalent normal form where every LI has at most one licensee feature (Graf et al. 2016). For an LI with at most two arguments, that is enough to render feature order superfluous. Such an LI has at most four features that aren't licensor features: two selector features, a category feature, and a licensee feature. The negative features must follow the positive ones, and given our previous discussion we can always infer which one of the two is a category feature and hence must precede the other negative feature. The selector features are ordered by the initial assumption that heads can distinguish between arguments and complements. Licensor features in Minimalist syntax always follow selector features, so their order is also known. So all the features of the LI are linearly ordered. For LIs with more than two arguments, one creates an extended projection of unpronounced functional heads, each one of which hosts one of these arguments. If desired, these extended projections can also be given licensee features to more accurately emulate intermediate movement of the LI. Even then each one of these heads still has an implicit feature order as described before. Overall, then, feature ordering can be removed from MGs because it is readily deduced from the logic of the feature calculus in combination with extended projections.

Now recall from Sec. 3.2 that constraints are done away with via category refinement, which acts on the features of every LI and annotates them with the states of a specific bottom-up tree automaton. This procedure is correct as long as one can discern which features of an LI are subcategorization features and match them to the correct argument. That is obviously the case if Merge and Move features are distinct and all features are ordered. Given the previous discussion, the refinement also works without an explicit Merge/Move split. And if one furthermore assumes that every head selects at most two arguments — as is commonly done in Minimalism — then the features can also be unordered, provided one indicates for each selector feature whether it takes a complement or a specifier. These considerations show that the specifics of the MG feature calculus also have little bearing on whether the interdefinability proof goes through.

There are two aspects of the feature calculus, though, that do matter for the feature-constraint equivalence. One — the SMC — is mostly of technical interest, whereas the other has some genuine linguistic bite to it. The SMC is an essential prerequisite for the MSO-definability of MDTLs and the MG output mapping. Removing the SMC thus breaks the equivalence between MSO-constraints and subcategorization, but only in one direction. In MGs without the SMC, MSO-constraints can still be compiled into the lexicon via category refinement, but removing features is no longer possible with MSO because the absence of the SMC allows for much more complicated movement patterns. That said, there is little reason to drop the SMC (see Sec. 2.3.4 of Graf 2013a for a detailed discussion), and it is still possible to employ the removal template with a more powerful logic than MSO (cf. Salvati 2011). Therefore feature removal is still an option for MGs without the SMC, just not while staying within the realm of MSO.

The linguistically more interesting aspect is that category refinement yields the right results only if subcategorization requirements are exact. That is to say, a verb that subcategorizes for a DP can only select a DP as an argument and not, say, an NP. An alternative system would be one where categories are organized hierarchically such that if $X > Y$, then every head that selects an XP can also select a YP. Such a system has been proposed by Fowlie (2013) as a means to reconcile ordering effects among adjectives with their optionality, without positing a cartographic spine of mostly empty projections. If it is a universal of natural languages that subcategorization must involve hierarchies of this kind such that one can never require an exact category match, then category refinement would no longer be a valid substitute for MSO-constraints. Depending on the properties of the hierarchies the procedure might still be salvageable, of course, but without further work we would no longer be in a position to posit the equivalence of features and constraints. However, establishing the empirical validity of such a category ordering universal will prove difficult in the absence of a worked out theory of what should count as a category. The theoretical status of categories also ties into my next point: does category refinement conflict with properties of UG?

4.2 The validity of category refinement

An implicit assumption of most generative research is that the set of categories is fixed by UG. Hence the CP node in a tree for English represents exactly the same category as the CP nodes in trees for Chinese, Tongan, or Inuktitut. All grammars share the substantive universal that categories must be sampled from a set containing V, N, D, P, Adj, Adv, v, T, C and possibly a few more, depending on the granularity of one's analysis. From this perspective, category refinement cannot be a valid strategy because it creates categories that are not furnished by UG and thus no grammar using such categories can be a natural language grammar. As a result, constraints cannot be reduced to features without moving beyond the class of natural languages, wherefore the two are demonstrably different mechanisms of the language faculty.

There are at least two problems with this line of reasoning. First, even if one accepts that the set of categories is fixed across all languages — which is far from an empirical truism — it might still be the case that this set is very large and each language uses only a fraction of the available categories. For example, it has been claimed that Navajo has no adjectives, in which case the corresponding category presumably goes unused in the grammar. But if for each language there is a large set of unused categories, then category refinement is still possible. Going back to the odd/even example from Sec. 3.2, the lexical item like :: D^{o+} D^{e+} V^{e-} could also be written as like :: A^{+} D^{+} Asp^{-} in a language that lacks adjectives and dedicated aspect heads. The name of a feature is completely irrelevant, what matters is what role it serves in the feature calculus. The feature A^{+} can fill exactly the same role as D^{o+} if the category A is not needed anywhere else in the grammar. This means that assuming a fixed set of categories is not enough to invalidate the category refinement procedure, one also has to show that there aren't enough unused categories in any given grammar to lexicalize any linguistically adequate constraints (or that such reusal is illicit for independent reasons).

Such an argument is impossible to make at this point because of the second, much more pressing problem: the concept of a syntactic category is not well-defined. What counts as a category is a very fuzzy affair that is almost purely driven by intuition rather than formal criteria. Linguists usually rely on morphology, semantics, and syntactic distribution, but neither are these properties clearcut nor is there an agreed-upon standard of their relative ranking.

The morphological criterion posits that two LIs belong to the same class if they are subject to all the same morphological operations, e.g. case marking or deverbalization. This implication only works in one direction since both *tall* and *alleged* are considered adjectives yet only the former has comparative and superlative forms. Furthermore, one must require all morphological operations to be shared in order to prevent nouns and adjectives from being lumped together in those languages where both share case marking. But then definite and indefinite determiners should belong to different categories in Icelandic, where only the former is inflected for case (because the latter is unpronounced). In order to reconcile these facts, morphological criteria have to be decoupled from surface patterns, which makes many categorizations unfalsifiable, or one has to posit a suitably quantified threshold for morphological overlap. To the best of my knowledge, neither has been advocated for in the literature, so that morphology gives at best a rough approximation of the category system that must be clarified by recourse to syntax and semantics.

But these two criteria do not fare much better. Semantics is well-known to crosscut categorial boundaries, as is illustrated by the semantic identity of the verb *destroy* and the noun *destruction*. Syntactic criteria, on the other hand, are mostly about the syntactic distribution of words, which is also confounded in multiple ways. For example, all verbs are assigned the category V even though intransitive and transitive verbs have completely different distributions. This is usually motivated by the fact that the phrases projected by verbs have identical distribution. But this line of reasoning actually supports category refinement rather than debunking it. For if the category of a head depends on the distribution of the phrase it projects, then one and the same transitive verb must have multiple categories depending on whether the object it selects is a reflexive. A VP containing the reflexive object *himself* needs a suitable c-commanding antecedent that would not be required if the object were simply *Mary* instead, wherefore the two VPs have distinct distributions and their respective VP-heads must have different categories under a distributionalist interpretation of categories.

The reader might object that binding effects are deliberately factored out into independent mechanisms like Principle A, but this just begs the question. The equivalence of features and constraints shows that the workload of regulating the distribution of words and phrases can be shifted around between phrases and categories. Linguists opt for factoring out certain aspects such as agreement and binding while leaving a small remnant to the category system. But as we just saw, there is no well-defined cutoff point for how much work categories are allowed to do. Even if there were it would still be unclear why this is the only suitable cutoff point for linguistic theories. In the absence of a full-fledged theory of categories, there is no principled way to reject category refinement as illicit.

This point is further strengthened by the observation that the category refinement procedure operates on subcategorization features, which are not necessarily the same as syntactic categories. In Sec. 3.2 I equated a feature like D^{o+} with the requirement to select an o-DP. But this is just one convenient interpretation out of many, and we can easily adopt a linguistically more sophisticated one. Selection requirements evidently pay attention to more than just the category of the selected argument. For example, verbs may require PP-arguments to be headed by a specific preposition (*laugh at* but **laugh to*) and some determiners only occur with count nouns (*a car* but **a furniture*). In formalisms likes GPSG that heavily rely on attribute value matrices (AVMs), this is easily accounted for. For instance, the AVM of *laugh* contains a subcategorization feature that requires a phrase whose category feature is P and whose head is pronounced *at*. The AVM of *a*, on the other hand, has a subcategorization feature that seeks a phrase with category feature N and type *count*.

$$\begin{bmatrix} \mathsf{Phon} & \text{laugh} \\ \mathsf{Cat} & \text{v} \\ \mathsf{Subcat} & \begin{bmatrix} \mathsf{Cat} & \text{P} \\ \mathsf{Phon} & \text{at} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \mathsf{Phon} & \text{a} \\ \mathsf{Cat} & \text{D} \\ \mathsf{Subcat} & \begin{bmatrix} \mathsf{Cat} & \text{N} \\ \mathsf{Type} & \text{count} \end{bmatrix} \end{bmatrix}$$

In such a system, the MG feature D^{o+} is simply interpreted as a subcategorization requirement for a phrase with category feature D and type o. So category refinement does not invariably alter the set of lexical categories; rather, it constitutes a refinement of selectional restrictions, which can be encoded in a variety of ways depending on the choice of feature system.

4.3 Interface constraints

In the translation from constraints to features (Sec. 3.2), all MSO-constraints were assumed to be stated over MG derivation trees, which makes them derivational/global constraints in the terminology of Müller & Sternefeld (2000). While early Minimalism still had its fair share of constraints operating within narrow syntax — e.g. the Shortest Derivation Principle (Chomsky 1995a) or Merge-over-Move (Chomsky 2000) — the debate has shifted a lot since then, with constraints now construed exclusively as Bare Output Conditions, i.e. interface constraints. In fact, this lends a new facet to the feature constraint dichotomy, which can now be framed as a more technical variant of the question whether certain phenomena are better accounted for in syntax proper or rather at the interfaces. Against this backdrop it seems misguided to compare features to derivational constraints, neither one of which is situated at the interfaces.

From a mathematical perspective, though, it is fairly easy to shift the workload between narrow syntax and the interfaces, at least given a specific technical interpretation of these notions. Let us define *syntactic constraints* as MSO formulas over Minimalist derivation trees and *interface constraints* as MSO formulas over the set of structures produced by some suitable, MSO-definable output mapping. Given these definitions, interface constraints can be lexicalized via category refinement in the same fashion as syntactic constraints. This follows immediately from the fact that every interface constraint can be automatically translated into a syntactic constraint. We see, then, that shifting our focus from syntactic constraints to interface constraints does not affect the close relation between features and constraints.

In many cases, the switch from interface constraints to derivational constraints is surprisingly easy. The MSO-constraint for Principle A from Sec. 2.2 was stated as a licensing condition over phrase structure trees.

$$\forall x \Big[\textit{anaphor}(x) \rightarrow \\ \exists y \big[\textit{DP}(y) \wedge \textit{c-com}(y,x) \wedge \phi\textit{-match}(y,x) \wedge \exists X \big[\textit{bind-dom}(X,x) \wedge X(y) \big] \big] \Big]$$

To adapt it to derivation trees, we only need to change the definitions of some predicates. A predicate $\textit{XP}(y)$ no longer holds of nodes labeled XP, but rather nodes that represent an operation that checked the last positive feature of some LI with category feature X. The predicates $\phi\textit{match}(x,y)$ and $\textit{bind-dom}(X,x)$ remain exactly the same (except that the meaning of $\textit{TP}(x)$ in the definition of binding domain has now changed to the derivational format). The biggest change happens to c-command, which is now stated in terms of derivational prominence. For any given LI l, x is its final occurrence iff x denotes the operation that checked the last negative feature of l. Let this relation be denoted by $\textit{f-occ}(x,l)$ (a precise definition is given in Graf 2012a and Graf 2013a). Then c-command (at S-structure) reduces to dominance between final occurrences.

$$\textit{c-com}(x,y) \leftrightarrow \exists z, z' [\textit{f-occ}(z,x) \wedge \textit{f-occ}(z',y) \wedge z \triangleleft^{+} z']$$

This shows that the constraints are stated in the same way over derivations and output structures, the only thing that changes is the definition of the structural predicates they invoke.

One interesting detail is that the reverse of the translation does not always work: some MSO formulas over Minimalist derivation trees do not have an equivalent MSO formula over output structure. This is a consequence of the MSO-definable output mapping possibly increasing the complexity of the output structure such that a local dependency over derivation

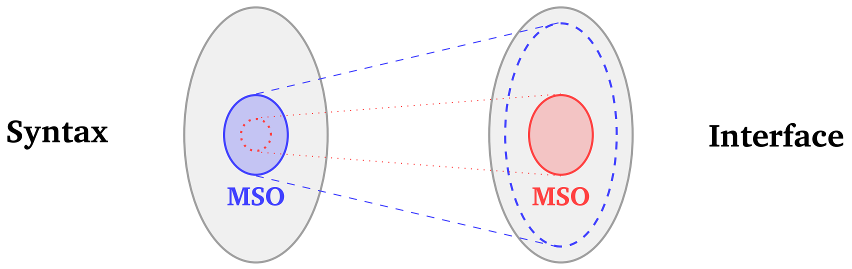


Figure 8 MSO-constraints that apply at the interfaces are weaker than those operating over derivation trees.

trees becomes a very intricate global dependency over output structures. The previous claim about interface constraints thus can be strengthened: every constraint over output structures, even if it is not MSO-definable, can be lexicalized via category refinement as long as it can be translated into an MSO-constraint over derivation trees (cf. Fig. 8).

Note that if one adopts the proposed definition of interface constraints as MSO formulas over output structures, then these new results may be seen as a formal analog of current claims that interface constraints cannot handle certain aspects of syntax (Preminger 2011). This would be particularly intriguing if it turned out that all the interface constraints commonly entertained by linguists are indeed MSO-definable, which might prove helpful in the search for linguistically more adequate constraint classes that will be outlined at the end of the paper in Sec. 5.2.

It might also be the case that some constraints that apply at the interfaces cannot be expressed in MSO even when they are stated directly over derivation trees. In particular constraints that depend on effects on meaning such as Rule I and Scope Economy are problematic in this respect (recall the discussion from Sec. 2.2, page 13). But these kinds of constraints are transderivational in nature and thus not commonly assumed in contemporary Minimalism (Collins 1996; Johnson & Lappin 1999; Müller & Sternefeld 2000). If such constraints turn out to be indispensable while syntax remains firmly within the bounds of MSO, this would constitute an argument that these constraints must apply at the interfaces. For the majority of constraints posited at the interfaces, though, a derivational, syntactic analogue is readily available.

4.4 Beyond generative capacity

All the results discussed so far approach the feature constraint dichotomy from a purely extensional perspective: can both mechanisms define the same sets of structures? This goes beyond weak or strong generative capacity in that it not only compares the sets of strings and phrase structure trees that can be generated. It also pays attention to whether the derivational process is carried out in the same fashion, i.e. whether the shape of derivation trees is affected by the choice between features and constraints. Since derivation trees directly represent the grammar and how it uses its operations, they are the closest any formalization can get to I-language (Chomsky 1986), which is commonly deemed the right level of linguistic reasoning in Minimalism.

Nonetheless many readers will indubitably feel that the results miss the mark because the linguistic debate is not about expressivity but rather explanatory adequacy. The question is not whether one encoding device is more powerful than the other, it is which one of the two provides an adequate explanation for the specific properties exhibited by natural languages. The renewed interest in constraints is due to the desire to derive specific patterns via plausible interface conditions rather than stipulative feature mechanisms, and thus it is a worthy enterprise irrespective of how the two devices line up with respect to generative capacity.

I fully share the desire to move away from stipulation and to find a natural characterization of syntactic mechanisms and dependencies, but I also have to caution against disregarding expressivity results simply because they do not capture the full picture. In the next section I will explain at length why the expressivity results in this paper have very far-reaching empirical consequences and carve out new research questions that have been neglected so far. The feature constraint equivalence may not provide a conclusive answer to the conceptual debate, but it nonetheless offers important linguistic insights.

In addition, contrasting features against constraints only makes sense if one posits that the grammar is a privileged cognitive module that is connected to interface systems. But one might just as well take inspiration from Marr & Poggio (1976) and Marr (1982) and view the grammar as a suitable abstraction of the performance system (cf. Neeleman & van de Koot 2010). In that case, it is conceivable that constraints are just a high-level description of mechanisms that are encoded by features at the performance level, and this idea — albeit highly speculative — has some intriguing implications.

MSO-constraints are much more succinct than the corresponding bottom-up tree automata that form the basis for category refinement (each quantifier alternation in a formula may induce an exponential blow-up in the number of states). As a result, category refinement can increase the difficulty of the parsing problem due to a sharp increase in the size of the grammar (see also Stabler 2013). Hence, if one assumes that the human parser operates with refined categories in place of a special mechanism for constraints, then all syntactic constraints must be simple enough that they can be lexicalized without making parsing intractable. In other words, feature encodings can be used as a window into what classes of constraints are efficiently processable. This lines up perfectly with syntacticians' desire to carve out a natural class of constraints, and as we will see soon such restrictions on constraints are sorely needed.

5 Implications for syntactic research

We have now firmly ascertained that the MG feature calculus is expressively equivalent to the class of MSO-constraints over derivation trees (Sec. 3) and that these results carry over even if the simplified mathematical model is brought more closely in line with standard linguistic practice (Sec. 4.1–4.3). Since virtually all syntactic constraints are MSO-constraints (Sec. 2.2), it follows that all syntactic constraints can be expressed via features, and the other way round.

This leaves us with the pressing question what this finding entails for syntactic research. One possible interpretation is that the equivalence of features of constraints renders all research in this area obsolete. Since one can be automatically translated into the other, the two cannot be distinguished by well-formedness judgments. In the absence of a principled theory of succinctness or explanatory adequacy, researchers should abstain from pitting one against the other and simply use whatever device they prefer. But this view neglects that theoretical debates in linguistics are usually linked to an empirical core, which is not at all voided by the findings reported here. Quite to the contrary, the results strengthen the importance of empirical considerations by carving out a well-defined set of research questions that require both computational and linguistic insight in order to be resolved in a satisfactory manner.

The equivalence of subcategorization and MSO-constraints highlights an as yet unnoticed loophole in linguistic formalisms that leads to massive overgeneration and predicts grossly inadequate language typologies.

In order to plug this loophole, we need a restricted theory of constraints, or equivalently, a restricted theory of features. The feature-constraint equivalence furnishes two attack vectors and it would be prudent for us to take advantage of both. Any proposal that restricts the power of our formalisms is progress, irrespective of whether it is stated via features or constraints.

5.1 Hidden overgeneration and false typologies

As explained in Sec. 2.2, MSO is a natural starting point for the formal study of constraints because it is mathematically well-understood and expressive enough to also accommodate very complicated syntactic constraints. The reducibility of MSO-constraints to subcategorization thus shows that pretty much every syntactic constraint in the literature can be lexicalized.

The downside, however, is that only a fraction of MSO-constraints look like reasonable constraints of natural language. For example, for every MSO-constraint there is a symmetric analog that just switches the direction of licensing requirements. As Principle A is an MSO-constraint, so must be *Reverse Principle A*, which requires every anaphor to locally c-command its antecedent rather than be c-commanded by it. Similarly there are $2^9 = 512$ PCC-like MSO-constraints on the distribution of pronominal elements in a system with three distinct persons, but only 4 are currently attested (Nevins 2007; Walkow 2012). It would also be trivial to write MSO-constraints to license the unattested A-B-A pattern of comparative formation (Bobaljik 2012) or to allow only those DP-internal word orders that are listed as unattested in Cinque (2005). MSO-constraints can also be sensitive to entirely non-linguistic notions like the number of nodes in a phrase such that, for instance, a CP is a phase iff it contains 11 pronounced LIs or its total number of nodes is a multiple of 17. All these unnatural constraints are expressible in every syntactic formalism with exact subcategorization (meaning the category of the argument must exactly match what the head selects for), and consequently none of these formalisms currently offer an explanation for their non-existence.

MSO-constraints are furthermore a bad model of how syntactic constraints may be modified or combined. The negation of an MSO-constraint is still an MSO-constraint, while the same is in general not true of a syntactic constraint. The negation of the CED, for example, would require every grammatical sentence to contain at least one phrase that has been extracted from a constituent in specifier position. In addition, MSO-constraints can also be combined via logical *or*. So there should be a language where a sentence is grammatical as long as it obeys binding theory or subject-verb agreement, but not necessarily both. In this hypothetical language, subject-verb agreement would be mandatory only in sentences without pronominal elements. To the best of my knowledge, nothing of this sort has ever been described in the literature. At most, syntactic constraints may be freely conjoinable via logical *and* (assuming that the empty language is a natural language to accommodate conjunction of mutually incompatible constraints). But even this may conflict with some implicational universals of the form "no language with constraint A displays constraint B". Whatever the class of attested syntactic constraints may look like, it cannot be closed under negation and disjunction, and possibly not even conjunction — all of which argues against MSO as a viable model.

Even if one stipulates the current set of constraints as UG primitives and bans free combinations thereof, MSO still opens up major loopholes in the theory. Among other things, island constraints can be rendered null and void by replacing movement by base merger in the target position. So the ungrammatical *Which book did John invite Mary after reading* could be generated by simply merging *which book* in Spec,CP. The subcategorization feature of *reading* is then taken care of by an MSO-predicate that picks out *which book* as the desired argument. This sequence of base merger and long-distance selection produces the same output as movement but completely circumvents the standard island constraints. Of course the island constraints can be amended to plug this loophole, but this does not block other strategies such as an intransitive variant of *reading* that may only occur in these pseudo-extraction cases. The options for relating elements and establishing dependencies are infinite in MSO. A generalized notion of movement that encompasses all of them but precludes other attested dependencies like agreement and binding is difficult to envision. Recall also that all MSO dependencies can be reduced to subcategorization requirements, and at this point it is completely unclear how one might distinguish subcategorization emulation of a licit dependency from subcategorization emulation of an illicit dependency; both are just instances of category refinement.

In sum, there are plenty of reasons to be worried about the feature constraint equivalence. Not only does it bring about massive overgeneration and faulty typologies, it also deprives existing constraints of their empirical force. The linguistic enterprise of characterizing the class of possible languages is undermined by the humongous MSO-power of subcategorization. Generative syntax has always aimed for highly restricted theories, in particular in the wake of Peters & Ritchie (1973). But without restrictions on subcategorization, most of these restrictions are rendered vacuous by the hidden MSO-power of the formalism they apply to.

There are three ways of addressing this problem. The first one is to show that MSO-constraints cannot be reduced to subcategorization due to certain properties of the syntactic formalism. As discussed in Sec. 4.1, this route does not look promising unless one can completely do away with the exact matching requirement for subcategorization.

The second option is to invoke third factors such as learnability and processing requirements so that even though grammars may use the full range of MSO-constraints, most of them will never be acquired by humans. This strategy is definitely viable. While the paradigms of formal learning theory bear little resemblance to the actual acquisition problem, there is wide consensus that the full class of MSO-definable languages cannot be learned under any reasonable paradigm. At this point, however, it is largely unclear what might be a reasonably learnable subset of these languages. Clark (2010) and related work present some promising results, but we still do not know enough to translate their learnability claims into meaningful properties of constraints or clear typological predictions.

For now, we should look for restrictions in the domain we understand best, i.e. the grammar. This is the credo of the third option. We have to look for weaker yet sufficiently expressive constraint classes and explore properties of the feature calculus in specific areas like the PCC (cf. Graf 2014a), in the hope that the two can ultimately be interlinked. Fortunately, first steps in this direction are already being made.

5.2 Towards a restricted theory of features/constraints

MSO plays a central role in this paper because of its exact correspondence to subcategorization and its ability to express virtually all syntactic constraints. Focusing only on the latter for a moment, one might wonder whether there is a weaker class of constraints that includes all syntactic constraints. This does indeed seem to be the case.

Careful examination of the logical formalization of MGs in Graf (2013a:App. B) shows that set quantification is never used. The same holds for the argument in Graf & Abner (2012) that syntactic binding is MSO-definable: it never invokes set quantification. And even in the formalization of Principle A in Sec. 2.2 the binding domain could have been defined without set quantification by directly referencing the relevant TP node. This pattern seems to extend to all syntactic constraints that operate over a single tree: set quantification is not needed because locality domains are tied to specific nodes. Whether transderivational constraints require set quantification is less clear at this point. Focus Economy and Merge over Move as formalized in Graf (2013a:Ch.4) do not, but the Shortest Derivation Principle might. In general, though, it seems fairly safe to assume that no prominent constraint in the syntactic literature requires MSO's set quantification. But MSO without set quantification is just first-order logic, which is strictly weaker than MSO.

First-Order Restriction Every syntactic constraint is definable in first-order logic.

While the first-order restriction does have some interesting formal implications, it does little to limit overgeneration. Among all the unnatural constraints and typologies discussed in the previous section, only the constraint that CP is a phase if its total number of nodes is a multiple of 17 is now blocked. But first-order logic can still conjoin and negate constraints, make any constituent with more than 11 pronounced LIs an island, and emulate Move via Merge. Therefore, first-order definability is as safe a restriction on constraints as it is a weak one.

At this point, it is unclear what additional restrictions could be imposed on the class of constraints. MGs have been known for a while to be definable with much less powerful devices than monadic second-order logic (Mönnich 2006, 2007; Kobele et al. 2007; Graf 2012a), in particular if one modulates the structures that constraints apply over (Graf & Heinz 2016). But these findings are stated with respect to standard MGs — how many constraints from the syntactic literature would fit into this tighter corset is still an open question. It is promising, however, that the same line of inquiry has yielded very tight characterizations for phonology (see Chandlee 2014, Heinz 2015, and references therein), and is also looking promising for morphology (Aksënova et al. 2016).

Our understanding of how much power is required for syntactic constraints would be more developed if the link between subclasses of MSO-constraints and the feature calculus had been explored more carefully. Even the restriction to first-order logic still lacks a feature-based characterization. In the other direction, basic concepts like feature geometries (Harley & Ritter 2002, among others) have not been couched in constraint-based terms yet.

Nonetheless certain observations can be made about features that strongly suggest that syntactic constraints form a highly restricted subclass of MSO. While every MSO-constraint can be emulated by the feature calculus, there is in general no guarantee that this emulation will be simple. Recall from Sec. 3.2 that the translation from MSO-constraints to features hinges on first building a bottom-up tree automaton, which uses a finite set of states to keep track of specific pieces of information. The number of states grows at a rapid rate with the complexity of the constraint, and as a result it is easy to obtain automata with thousands or even millions of states. But when syntacticians present a feature-based account of a specific phenomenon, the number of features is usually very small and their interaction rather simple. Unless this simplicity comes at the cost of increased complexity somewhere else (e.g. a modified, much more elaborate probe-goal system), the small number of required features shows that the constraint encoded this way is not nearly as involved as MSO would allow it to be. In other words, the fact that feature emulation is a messy and convoluted affair for most MSO-constraints makes the simplicity of feature-based accounts in the literature even more remarkable.

On an intuitive level, then, there is ample evidence that syntactic constraints operate within the bounds of a small subset of the class of MSO-constrains, but we have not reached the point yet where this intuition can be articulated more precisely. The success of any such attempt hinges on a tight link between features and constraints. With translations between constraint classes and feature systems, it will be possible to convert linguistic findings into mathematical restrictions and, the other way round, formulate mathematically motivated language universals that can easily be tested by linguists. The equivalence of MSO-constraints and subcategorization is just the first step in this direction, but it has already brought a plethora of less prominent issues into the limelight:

i. What is a syntactic category/part of speech?
ii. How exact is subcategorization?
iii. Are there constructions that do not involve subcategorization, and do they behave differently with respect to constraints?
iv. How can constraints combine, and what does that entail for typology?
v. Which constraints, if any, have symmetric analogs?
vi. Is there any natural language system (e.g. gender agreement) where every MSO-definable pattern is attested in some language?
vii. If not, are the missing patterns due to substantive properties of language (e.g. a specific feature geometry) or formal ones (the relevant MSO-dependencies simply cannot be expressed)?

These are very rich questions that, although prompted by mathematical considerations, have tremendous empirical substance. Some of them are easier to answer than others. Consider question iii. Adjunction is often formalized as asymmetric feature checking in MGs (Frey & Gärtner 2002).

The adjunct needs to check one of its features against the category feature of the head it adjoins to, whereas the head does not have any of its features checked. In this system, not all constraints can be correctly emulated via category refinement and adjuncts thus should behave differently from arguments with respect to certain constraints. As is well-known, adjuncts are indeed special in various ways. For instance, pronominal elements inside adjuncts often do not obey standard binding theory. So the tentative answer to question iii is yes, adjuncts are best modeled with a different selection mechanism and also show special behavior with respect to constraints. Whether their special behavior can be completely derived from asymmetric feature checking remains to be seen, though (see Graf 2013b for a first attempt). Whatever the answer may be, it is clear that the interdefinability of constraints and features opens up an interesting new way of thinking, regarding not only the technical machinery but also the questions one may ask about syntax.

6 Conclusion

Graf (2013a) has shown that features and constraints are expressively equivalent in a specific formal setting, and I have argued at length in this paper that these mathematical findings carry over to linguistic practice. More precisely:

i. Every MSO-definable constraint can be expressed via subcategorization requirements.
ii. Feature calculi can be eliminated in every MSO-definable grammar formalism (assuming they are non-recursive).

While subcategorization is rarely talked about in Minimalism (notable exceptions include Svenonius 1994, Adger 2003 and Müller 2010), there currently seems to be no viable alternative to the standard system of category features and complex subcategorization requirements familiar from AVM-based formalisms. Hence Minimalism is subject to result i and must find a solution to the overgeneration problem created by MSO.

Fortunately, Minimalism can also be construed as an MSO-definable grammar formalism, which means that feature-based accounts and category-based accounts are interdefinable and thus equally useful in limiting excessive power. Contrasting what MSO (and hence our formalism) is capable of against the behavior of natural languages will be essential in the search for the right subclass of constraints/feature systems and will also bring forth new insights about syntactic categories, subcategorization, and typological generalizations.

References

Abels, Klaus. 2003. *Successive cyclicity, anti-locality, and adposition stranding*: University of Conneticut dissertation.

Adger, David. 2003. *Core syntax: A Minimalist approach*. Oxford: Oxford University Press.

Adger, David. 2006. Remarks on Minimalist feature theory and Move. *Journal of Linguistics* 42. 663–673. http://dx.doi.org/10.1017/S0022226706004221.

Adger, David. 2010. A Minimalist theory of feature structure. In Anna Kibort & Greville G. Corbett (eds.), *Features: Perspectives on a key notion in linguistics*, 185–218. Oxford: Oxford University Press. http://dx.doi.org/10.1093/acprof:oso/9780199577743.001.0001.

Adger, David & Daniel Harbour. 2008. Why phi? In Daniel Harbour, David Adger & Susana Béjar (eds.), *Phi theory: Phi features across interfaces and modules*, 1–34. Oxford: Oxford University Press.

Aksënova, Alëna, Thomas Graf & Sedigheh Moradi. 2016. Morphotactics as tier-based strictly local dependencies, To appear in *Proceedings of SIGMorPhon 2016*.

Backofen, Rolf, James Rogers & K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information* 4. 5–39. http://dx.doi.org/10.1007/BF01048403.

Blackburn, Patrick, Claire Gardent & Wilfried Meyer-Viol. 1993. Talking about trees. In *Proceedings of the sixth conference of the European chapter of the Association for Computational Linguistics*, 30–36. http://dx.doi.org/10.3115/976744.976748.

Bloem, Roderick & Joost Engelfriet. 2000. A comparison of tree transductions defined by monadic second-order logic and attribute grammars. *Journal of Computational System Science* 61. 1–50. http://dx.doi.org/10.1006/jcss.1999.1684.

Bobaljik, Jonathan D. 2012. *Universals in comparative morphology: Suppletion, superlatives, and the structure of words*. Cambridge, MA: MIT Press.

Boston, Marisa Ferrara, John T. Hale & Marco Kuhlmann. 2010. Dependency structures derived from Minimalist grammars. In Christian Ebert, Gerhard Jäger & Jens Michaelis (eds.), *Mol 10/11*, vol. 6149 Lecture Notes in Computer Science, 1–12. Berlin: Springer. http://dx.doi.org/

10.1007/978-3-642-14322-9_1.

Bruening, Benjamin. 2014. Precede-and-command revisited. *Language* 90. 342–388. http://dx.doi.org/10.1353/lan.2014.0037.

Büchi, J. Richard. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6. 66–92. http://dx.doi.org/10.1002/malq.19600060105.

Chandlee, Jane. 2014. *Strictly local phonological processes*: University of Delaware dissertation.

Chomsky, Noam. 1981. *Lectures on government and binding: The Pisa lectures*. Dordrecht: Foris.

Chomsky, Noam. 1986. *Knowledge of language: Its nature, origin, and use*. New York: Praeger.

Chomsky, Noam. 1993. A Minimalist program for linguistic theory. In Kenneth Hale & Samuel Jay Keyser (eds.), *The view from building 20*, 1–52. Cambridge, MA: MIT Press.

Chomsky, Noam. 1995a. Categories and transformations. In *The Minimalist program*, chap. 4, 219–394. Cambridge, MA: MIT Press. http://dx.doi.org/10.7551/mitpress/9780262527347.003.0004.

Chomsky, Noam. 1995b. *The Minimalist Program*. Cambridge, MA: MIT Press. http://dx.doi.org/10.7551/mitpress/9780262527347.003.0003.

Chomsky, Noam. 2000. Minimalist inquiries: The framework. In Roger Martin, David Michaels & Juan Uriagereka (eds.), *Step by step: Essays on Minimalist syntax in honor of Howard Lasnik*, 89–156. Cambridge, MA: MIT Press.

Chomsky, Noam. 2001. Derivation by phase. In Michael J. Kenstowicz (ed.), *Ken Hale: A life in language*, 1–52. Cambridge, MA: MIT Press.

Chomsky, Noam. 2004. Beyond explanatory adequacy. In Adriana Belletti (ed.), *Structures and beyond: The cartography of syntactic structures volume 3*, 104–131. Oxford: Oxford University Press.

Chomsky, Noam. 2007. Approaching UG from below. In Uli Sauerland & Hans-Martin Gärtner (eds.), *Interfaces + recursion = language?: Chomsky's minimalism and the view from syntax-semantics*, 1–30. Berlin: Mouton de Gruyter. http://dx.doi.org/10.1515/9783110207552-001.

Chomsky, Noam. 2008. On phases. In Robert Freidin, Carlos P. Otero & Maria Luisa Zubizarreta (eds.), *Foundational issues in linguistic theory: Essay in honor of Jean-Roger Vergnaud*, 133–166. Cambridge, MA: MIT Press. http://dx.doi.org/10.7551/mitpress/9780262062787.003.0007.

Chomsky, Noam. 2013. Problems of projection. *Lingua* 130. 33–49. http://dx.doi.org/10.1016/j.lingua.2012.12.003.

Cinque, Guglielmo. 2005. Deriving Greenberg's Universal 20 and its exceptions. *Linguistic Inquiry* 36. 315–332. http://dx.doi.org/10.1162/0024389054396917.

Clark, Alexander. 2010. Efficient, correct, unsupervised learning of context-sensitive languages. In *Proceedings of the fourteenth conference on computational natural language learning (CoNLL)*, 28–37. Stroudsburg, PA: Association for Computational Linguistics.

Collins, Chris. 1996. *Local economy*. Cambridge, MA: MIT Press.

Cornell, Thomas & James Rogers. 1998. Model theoretic syntax. In *The Glot international state of the article book*, vol. 1 Studies in Generative Grammar 48, 101–125. Mouton de Gruyter. http://dx.doi.org/10.1515/9783110822861.171.

Culy, Christopher. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy* 8. 345–351.

Doner, John. 1970. Tree acceptors and some of their applications. *Journal of Computer and System Sciences* 4. 406–451. http://dx.doi.org/10.1016/S0022-0000(70)80041-1.

Engelfriet, Joost. 1975. Bottom-up and top-down tree transformations — a comparison. *Mathematical Systems Theory* 9. 198–231. http://dx.doi.org/10.1007/BF01704020.

Engelfriet, Joost & Sebastian Maneth. 2003. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing* 32. 950–1006. http://dx.doi.org/10.1137/S0097539701394511.

Fowlie, Meaghan. 2013. Order and optionality: Minimalist grammars with adjunction. In András Kornai & Marco Kuhlmann (eds.), *Proceedings of the 13th meeting on the mathematics of language (MoL 13)*, 12–20.

Fox, Danny. 2000. *Economy and semantic interpretation*. Cambridge, MA: MIT Press.

Fox, Danny & David Pesetsky. 2005. Cyclic linearization of syntactic structure. *Theoretical Linguistics* 31. 1–45.

Frey, Werner & Hans-Martin Gärtner. 2002. On the treatment of scrambling and adjunction in Minimalist grammars. In Gerhard Jäger, Paola Monachesi, Gerald Penn & Shuly Wintner (eds.), *Proceedings of the conference on Formal Grammar*, 41–52.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum & Ivan A. Sag. 1985. *Generalized phrase structure grammar*. Oxford: Blackwell.

Georgi, Doreen & Martin Salzmann. 2011. DP-internal double agreement is not double Agree: Consequences of Agree-based case assignment within DP. *Lingua* 121. 2069–2088. http://dx.doi.org/10.1016/j.lingua.2011.07.010.

Graf, Thomas. 2011. Closure properties of Minimalist derivation tree languages. In Sylvain Pogodalla & Jean-Philippe Prost (eds.), *LACL 2011*, vol. 6736 Lecture Notes in Artificial Intelligence, 96–111. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-22221-4_7.

Graf, Thomas. 2012a. Locality and the complexity of Minimalist derivation tree languages. In Philippe de Groot & Mark-Jan Nederhof (eds.), *Formal grammar 2010/2011*, vol. 7395 Lecture Notes in Computer Science, 208–227. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-32024-8_14.

Graf, Thomas. 2012b. Movement-generalized Minimalist grammars. In Denis Béchet & Alexander J. Dikovsky (eds.), *LACL 2012*, vol. 7351 Lecture Notes in Computer Science, 58–73. http://dx.doi.org/10.1007/978-3-642-31262-5_4.

Graf, Thomas. 2013a. *Local and transderivational constraints in syntax and semantics*: UCLA dissertation.

Graf, Thomas. 2013b. The syntactic algebra of adjuncts. In *Proceedings of CLS 49*, To appear.

Graf, Thomas. 2014a. Feature geometry and the Person Case Constraint: An algebraic link. In *Proceedings of CLS 50*, To appear.

Graf, Thomas. 2014b. Late merge as lowering movement in Minimalist grammars. In Nicholas Asher & Sergei Soloviev (eds.), *LACL 2014*, vol. 8535 Lecture Notes in Computer Science, 107–121. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-662-43742-1_9.

Graf, Thomas. 2014c. Models of adjunction in Minimalist grammars. In Glynn Morrill, Reinhard Muskens, Rainer Osswald & Frank Richter (eds.), *Formal grammar 2014*, vol. 8612 Lecture Notes in Computer Science, 52–68. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-662-44121-3_4.

Graf, Thomas & Natasha Abner. 2012. Is syntactic binding rational? In *Proceedings of the 11th international workshop on Tree Adjoining Grammars and related formalisms (TAG+11)*, 189–197.

Graf, Thomas, Alëna Aksënova & Aniello De Santo. 2016. A single movement normal form for Minimalist grammars. To appear in *Proceedings of Formal Grammar 2016*.

Graf, Thomas & Jeffrey Heinz. 2016. Tier-based strict locality in phonology and syntax. Ms., Stony Brook University and University of Delaware.

Hale, John T. & Edward P. Stabler. 2005. Strict deterministic aspects of Minimalist grammars. In Philippe Blache, Edward P. Stabler, Joan Busquets & Richard Moot (eds.), *Logical aspects of computational linguistics: 5th international conference*, 162–176. Berlin, Heidelberg: Springer.

http://dx.doi.org/10.1007/11422532_11.

Harley, Heidi & Elizabeth Ritter. 2002. Person and number in pronouns: A feature-geometric analysis. *Language* 78. 482–526. http://dx.doi.org/10.1353/lan.2002.0158.

Heim, Irene. 1998. Anaphora and semantic interpretation: A reinterpretation of Reinhart's approach. In Uli Sauerland & O. Percus (eds.), *The interpretive tract*, vol. 25 MIT Working Papers in Linguistics, 205–246. Cambridge, MA: MIT Press.

Heinat, Fredrik. 2006. *Probes, pronouns and binding in the Minimalist program*: University of Lund dissertation.

Heinz, Jeffrey. 2015. The computational nature of phonological generalizations. Ms., University of Delaware.

Hornstein, Norbert. 2001. *Move! A Minimalist theory of construal*. Oxford: Blackwell.

Hunter, Tim. 2011. Insertion Minimalist grammars: Eliminating redundancies between merge and move. In Makoto Kanazawa, András Kornai, Marcus Kracht & Hiroyuki Seki (eds.), *The mathematics of language: 12th biennial conference*, 90–107. Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-23211-4_6.

Hunter, Tim. 2015. Deconstructing merge and move to make room for adjunction. *Syntax* 18. 266–319. http://dx.doi.org/10.1111/synt.12033.

Huybregts, M. A. C. 1984. The weak adequacy of context-free phrase structure grammar. In Ger J. de Haan, Mieke Trommelen & Wim Zonneveld (eds.), *Van periferie naar kern*, 81–99. Dordrecht: Foris.

Johnson, David & Shalom Lappin. 1999. *Local constraints vs. economy*. Stanford: CSLI.

Kanazawa, Makoto. 1998. *Learnable classes of categorial grammars*. Stanford: CSLI.

Kepser, Stephan. 2008. A landscape of logics for finite unordered unranked trees. In Philippe de Groote, Laura Kallmeyer, Gerald Penn & Giorgio Satta (eds.), *FG-2008*, 23–40.

Kobele, Gregory M. 2002. Formalizing mirror theory. *Grammars* 5. 177–221.

Kobele, Gregory M. 2006. *Generating copies: An investigation into structural identity in language and grammar*: UCLA dissertation.

Kobele, Gregory M. 2010. A formal foundation for A and A-bar movement. In Christan Ebert, Gerhard Jäger & Jens Michaelis (eds.), *The mathematics of language*, vol. 6149 Lecture Notes in Computer Science, 145–159. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-14322-9_12.

Kobele, Gregory M. 2011. Minimalist tree languages are closed under intersection with recognizable tree languages. In Sylvain Pogodalla & Jean-Philippe Prost (eds.), *LACL 2011*, vol. 6736 Lecture Notes in Artificial Intelligence, 129–144. http://dx.doi.org/10.1007/978-3-642-22221-4_9.

Kobele, Gregory M., Travis C. Collier, Charles E. Taylor & Edward P. Stabler. 2002. Learning mirror theory. In *Proceedings of the sixth international workshop on Tree Adjoining Grammars and related frameworks (TAG+6)*, Venice.

Kobele, Gregory M., Christian Retoré & Sylvain Salvati. 2007. An automata-theoretic approach to Minimalism. In James Rogers & Stephan Kepser (eds.), *Model theoretic syntax at 10*, 71–80.

Lebeaux, David. 1988. *Language acquisition and the form of the grammar*: University of Massachusetts, Amherst dissertation. http://dx.doi.org/10.1075/z.97. Reprinted in 2000 by John Benjamins.

Marr, David. 1982. *Vision: A computational investigation into the human representation and processing of visual information*. New York: Freeman. http://dx.doi.org/10.7551/mitpress/9780262514620.001.0001. Reprinted in 2010 by MIT Press.

Marr, David & Tomaso Poggio. 1976. From understanding computation to understanding neural circuitry. Tech. rep. Artifical Intelligence Laboratory, MIT, AIM-357.

Michaelis, Jens & Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In Christian Retoré (ed.), *Logical aspects of computational linguistics*, vol. 1328 Lecture Notes in Artifical Intelligence, 329–345. Springer. http://dx.doi.org/10.1007/BFb0052165.

Mönnich, Uwe. 2006. Grammar morphisms. Ms. University of Tübingen.

Mönnich, Uwe. 2007. Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. In James Rogers & Stephan Kepser (eds.), *Model theoretic syntax at 10*, 83–87.

Mönnich, Uwe. 2012. A logical characterization of extended TAGs. In *Proceedings of the 11th international workshop on Tree Adjoining Grammars and related formalisms (TAG+11)*, 37–45. Paris, France.

Morawietz, Frank. 2003. *Two-step approaches to natural language formalisms*. Berlin: Walter de Gruyter. http://dx.doi.org/10.1515/9783110197259.

Morawietz, Frank & Thomas Cornell. 1999. The MSO logic automaton connection in linguistics. In Alain Lecomte, François Lamarche & Guy Perrier (eds.), *Logical aspects of computational linguistics: Second international conference*, vol. 1582, 112–131. Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/3-540-48975-4_6.

Müller, Gereon. 2010. On deriving CED effects from the PIC. *Linguistic Inquiry* 41. 35–82. http://dx.doi.org/0.1162/ling.2010.41.1.35.

Müller, Gereon. 2014. *Syntactic buffers* Linguistische Arbeitsberichte. Leipzig, Germany: Universität Leipzig.

Müller, Gereon & Wolfgang Sternefeld. 2000. The rise of competition in syntax: A synopsis. In Wolfgang Sternefeld & Gereon Müller (eds.), *Competition in syntax*, 1–68. Berlin: Mouton de Gruyter. http://dx.doi.org/10.1515/9783110829068.1.

Neeleman, Ad & Hans van de Koot. 2010. Theoretical validity and psychological reality of grammatical code. In Martin Everaert, Tom Lentz, Hannah de Mulder, Oystein Nilsen & Arjen Zondervan (eds.), *The linguistic enterprise: From knowledge of language to knowledge of linguistics*, 183–212. Amsterdam: John Benjamins. http://dx.doi.org/10.1075/la.150.08nee.

Nevins, Andrew. 2007. The representation of third person and its consequences for person-case effects. *Natural Language and Linguistic Theory* 25. 273–313.

Nunes, Jairo. 2004. *Linearization of chains and sideward movement*. Cambridge, MA: MIT Press.

Ochi, Masao. 1999. Multiple spell-out and PF adjacency. In *Proceedings of the North Eastern Linguistic Society*, vol. 29, 193–206.

Peters, Stanley & Robert W. Ritchie. 1973. On the generative power of transformational grammars. *Information Sciences* 6. 49–83. http://dx.doi.org/10.1016/0020-0255(73)90027-3.

Preminger, Omer. 2011. *Agreement as a fallible operation*: MIT dissertation.

Pullum, Geoffrey K. 2007. The evolution of model-theoretic frameworks in linguistics. In James Rogers & Stephan Kepser (eds.), *Model-theoretic syntax @ 10*, 1–10.

Radzinski, Daniel. 1991. Chinese number names, tree adjoining languages, and mild context sensitivity. *Computational Linguistics* 17. 277–300.

Reinhart, Tanya. 1983. *Anaphora and semantic interpretation*. Chicago University Press: Croon-Helm.

Reuland, Eric. 2001. Anaphors, logophors and binding. In Peter Cole, C.T. James Huang & Gabriella Hermon (eds.), *Long-distance reflexives*, 343–370. Burlington, MA: Academic Press.

Richards, Norvin. 2016. *Contiguity theory*. Cambridge, MA: MIT Press.

Rogers, James. 1996. What does a grammar formalism say about a language? Tech. Rep. 96-10 Institue for Research in Cognitive Science University of Pennsylvania.

Rogers, James. 1997. “Grammarless” phrase structure grammar. *Linguistics and Philosophy* 20. 721–746.

Rogers, James. 1998. *A descriptive approach to language-theoretic complexity*. Stanford: CSLI.

Rogers, James. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science* 293. 291–320. http://dx.doi.org/10.1016/S0304-3975(01)00349-8.

Salvati, Sylvain. 2011. Minimalist grammars in the light of logic. In Sylvain Pogodalla, Myriam Quatrini & Christian Retoré (eds.), *Logic and grammar — essays dedicated to Alain Lecomte on the occasion of his 60th birthday* (Lecture Notes in Computer Science 6700), 81–117. Berlin: Springer. http://dx.doi.org/10.1007/978-3-642-21490-5_5.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3). 333–345. http://dx.doi.org/10.1007/BF00630917.

Stabler, Edward P. 1997. Derivational Minimalism. In Christian Retoré (ed.), *Logical aspects of computational linguistics*, vol. 1328 Lecture Notes in Computer Science, 68–95. Berlin: Springer. http://dx.doi.org/10.1007/BFb0052152.

Stabler, Edward P. 2003. Comparing 3 perspectives on head movement. In A. Mahajan (ed.), *Syntax at sunset 3: Head movement and syntactic theory*, vol. 10 UCLA Working Papers in Linguistics, 178–198. Los Angeles, CA: UCLA.

Stabler, Edward P. 2006. Sidewards without copying. In Gerald Penn, Giorgio Satta & Shuly Wintner (eds.), *Formal Grammar '06, proceedings of the conference*, 133–146. Stanford: CSLI.

Stabler, Edward P. 2011. Computational perspectives on Minimalism. In Cedric Boeckx (ed.), *Oxford handbook of linguistic Minimalism*, 617–643. Oxford: Oxford University Press. http://dx.doi.org/10.1093/oxfordhb/9780199549368.013.0027.

Stabler, Edward P. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science* 5. 611–633. http://dx.doi.org/10.1111/tops.12031.

Stabler, Edward P. & Edward Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science* 293. 345–363. http://dx.doi.org/10.1016/S0304-3975(01)00351-6.

Stepanov, Arthur. 2001. Late adjunction and Minimalist phrase structure. *Syntax* 4. 94–125. http://dx.doi.org/10.1111/1467-9612.00038.

Svenonius, Peter. 1994. C-selection as feature-checking. *Studia Linguistica* 48. 133–155. http://dx.doi.org/10.1111/j.1467-9582.1994.tb00853.x.

Takahashi, Shoichi & Sarah Hulsey. 2009. Wholesale Late Merger: Beyond the A/$\overline{\text{A}}$ distinction. *Linguistc Inquiry* 40. 387–426.

Thatcher, James W. & J. B. Wright. 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory* 2(1). 57–81. http://dx.doi.org/10.1007/BF01691346.

Walkow, Martin. 2012. *Goals, big and small*: University of Massachusetts Amherst dissertation.