University of California

Los Angeles

Local and Transderivational Constraints in Syntax and Semantics

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Linguistics

by

Thomas Graf

2013

ABSTRACT OF THE DISSERTATION

Local and Transderivational Constraints in Syntax and Semantics

by

Thomas Graf

Doctor of Philosophy in Linguistics

University of California, Los Angeles, 2013

Professor Edward P. Stabler, Chair

A long-standing tension in Minimalist syntax is that between the structure-building operations Merge and Move on the one hand and the constraints restricting the shape of the structures built by said operations on the other. Proposals differ vastly in how much weight they attribute to each component, partially because there seems to be no principled connection between the two — constraints can easily be grafted onto any syntactic theory, and the choice of constraints is apparently independent of the types of posited operations. As a consequence, many foundational questions still lack satisfying answers: What kinds of constraints are there? What is their respective power, and are there any factors that could limit it? Are there phenomena that can only be explained via constraints? Why would syntax have both operations and constraints?

My thesis explores these and related questions from a mathematically informed perspective. The central result is that Minimalist syntax can express a constraint purely via the operation Merge iff computing said constraint requires only a finitely bounded amount of working memory iff the constraint can be defined by an extension of first-order logic known as monadic second-order logic. A peculiar lexicalization procedure is used to establish the equivalence between Merge and constraints.

The procedure pushes the working memory configurations that emerge during the computation of a constraint directly into the lexical categories. Refining categories in this fashion allows the selectional restrictions of lexical items to act as a proxy through which constraints are expressed via Merge.

Merge-expressible constraints are very powerful and can capture dependencies between nodes as well as the domains that such dependencies are usually relativized to. Hence almost all conditions proposed in the syntactic literature belong to this class, including transderivational constraints and economy conditions. Surprisingly, the power of Merge-expressible constraints does not vary with respect to the type of syntactic tree structure they are stated over (phrase structure tree, multidominance tree, derivation tree), nor is it affected by locality restrictions. With respect to the issues raised above, then, the emerging picture is that the kind of constraints entertained by syntacticians belong to a uniform class of structural conditions that is computationally well-behaved and tightly linked to the foundations of the framework. The existence of these constraints in language is a natural consequence of the core component of Minimalist syntax: feature-driven Merge.

Even though all claims are stated and proved with mathematical rigor, little knowledge is presupposed beyond a basic familiarity with generative syntax. Minimalist grammars are used as a formalization of Minimalist syntax, complemented with ideas from formal language theory and mathematical logic. All concepts are carefully explained and illustrated with numerous examples, including some advanced notions that so far have only been covered in highly technical papers. Thus the thesis should be of interest not only to theoretically minded syntacticians, but also to computational linguistics and everybody on their way to becoming one.

The dissertation of Thomas Graf is approved.

David Kaplan

Edward L. Keenan

Dominique Sportiche

Edward P. Stabler, Committee Chair

University of California, Los Angeles

2013

Don't you use your fancy mathematics to muddy the issue!

Applejack, My Little Pony: Friendship is Magic

Table of Contents

List of Examples

List of Figures

Φ_{gr}	mapping from derivations to multidominance trees
Φ_{tr}	mapping from derivations to phrase structure trees
ζ	mapping from lexical items to slices
CFG	context-free (string) grammar
EPP	Extended Projection Principle
FSL	free slice language
INC	Identity of Numerations Condition
(l)butt	(linear) bottom-up tree transducer
LI	lexical item
(l)tdtt	(linear) top-down tree transducer
MCFG	multiple context-free grammar
MCFL	multiple context-free language
MDTL	Minimalist derivation tree language
MG	Minimalist grammar
MGRC	Minimalist grammar with rational constraints
MOM	Merge-over-Move constraint
MRTG	multiple regular tree grammar
MRTL	multiple regular tree language
MSO	monadic second-order logic
PBC	Proper Binding Condition
PMCFG	parallel multiple context-free grammar
PMCFL	parallel multiple context-free language
RC	reference-set constraint
SDP	Shortest Derivation Principle
SMC	Shortest Move Constraint
SPIC	Specifier Island Constraint

TAG Tree adjoining grammar

TAL Tree adjoining language

Acknowledgments

If PHD comics is to be believed, grad school is a never-ending plight, a soul-crushing purgatory where young, idealistic minds get to toil away in solitude while their cohorts are already transitioning into the proper adult lifestyle of family, money, careers. The thesis is a beacon of hope, the promise of salvation that will allow the devout grad student to ascend to the heavens of tenure track — or so they say. Frankly, I have no idea what all the fuss is about.

Writing this thesis wasn't a defining experience in my life, let alone a dreadful one. After all, it would not be an academic lifestyle without spending hours in front of a computer deprived of any social contact. Moreover, I had an excellent thesis committee, and I am grateful to all the members for making the gestation process an enjoyable experience. But whereas writing the thesis was only an enjoyable experience, not a defining one, the great time I had at UCLA definitely meets both criteria. Looking back, it is hard to believe how much of an impact the last five years have had one me (strictly positive, I assure you).

Of course nobody has shaped my recent thinking more than Ed Stabler. In fact, Ed is the reason I am a mathematical linguist today. As an undergraduate I found myself disillusioned with both computational linguistics and generative syntax. But then, one lucky day, I stumbled upon a certain paper called "Derivational Minimalism" and even though I didn't understand a word — those pesky symbol salads sure were hard to decipher back then — I immediately felt that this is the way to make sense of language, that this is what I had to do. A lesser man might have failed to sustain my interest, but every new project Ed starts gets me even more excited. He is a great teacher to boot, a brilliant advisor, and has well-reasoned opinions on the two most important aspects of life: Linux and Star Trek.

I am also indebted to Ed Keenan for his extraordinary support from the very beginning, for opening my eyes to the beauty of Boolean algebra, and for being the

most reliable supplier of funny yet insightful anecdotes; if only I had taped them all. Besides the two Eds, many other faculty have helped me in one way or the other over the years. Dominique Sportiche and Kie Zuraw in particular always had an open ear for my ideas about syntax and phonology, respectively.

Many more people deserve a shout-out, be it for personal or professional reasons. Without going into further details, let it just be said that the following people have earned the highly prestigious Thomas Graf seal of approval™: Natasha Abner, Mel Bervoets, Joe Buffington, Heather Burnett, Karen Campbell, Alex Drummond, Jenn Fischer, Meaghan Fowlie, Michael Freedman, Hans-Martin Gärtner, Mattyas Huggard, Tim Hunter, Dana Kleifield, Greg Kobele, Hilda Koopman, Natasha Korotkova & Vanya Kapitonov, Marcus Kracht, Shalom Lappin, Winnie Lechner, Ingvar Löfstedt, Jens Michaelis, Uwe Mönnich, Gereon Müller, Anna Pagé & Bernhard Koller, James Pannacciulli, Martin Prinzhorn, Jim Rogers, Uli Sauerland, Michael Tseng, Tamara Vardomskaya, Sarah van Wagenen, Martin Walkow, Alexis Wellwood, Kristine Yu, and Sarah Zobel.

A 6000 miles long-distance Thank You note goes to my parents. They have had to put up with a lot of things due to me, some of them even worse than constantly being asked by their friends what this weird mathematical linguistics thing is all about.

Ein 9000 Kilometer Dankeschön-Ferntelegramm geht an meine Eltern. Sie haben sich wegen mir mit so einigem abfinden müssen, manches sogar schlimmer als die unentwegten Fragen von Bekannten worum es denn bei dieser komischen Mathematischen Linguistik überhaupt geht.

VITA

2007	Mag.phil. (≈ M.A.) in Linguistics, *summa cum laude*, University of Vienna.
2009	Springer Prize for Best Student Paper at ESSLLI2009
2010	M.A. in Linguistics, UCLA.
2010 & 2011	DOC-Fellowship of the Austrian Academy of Sciences

PUBLICATIONS

Graf, Thomas. 2012. Concealed Reference-Set Computation: How Syntax Escapes the Parser's Clutches. In Anna Maria Di Sciullo (ed.), *Towards a Biolinguistic Understanding of Grammar. Essays on Interfaces*, 339–362. John Benjamins: Amsterdam.

Graf, Thomas. 2012. Movement-Generalized Minimalist Grammars. In Denis Béchet and Alexander Dikovsky (eds.), *Proceedings of LACL 2012*, Lectures Notes in Computer Science 7351, 58–73. Springer: Heidelberg.

Graf, Thomas. 2011. Closure Properties of Minimalist Derivation Tree Languages. In Sylvain Pogodalla and Jean-Philippe Prost (eds.), *Proceedings of LACL 2011*, Lectures Notes in Artifical Intelligence 6736, 96–111. Springer: Heidelberg.

Graf, Thomas. 2010. Formal Parameters of Phonology: From Government Phonology to SPE. In Thomas Icard and Reinhard Muskens (eds.), *Interfaces: Explorations in*

Logic, Language and Computation, Lecture Notes in Computer Science 6211, 72–86. Springer: Berlin.

Graf, Thomas. 2007. Agreement with Hybrid Nouns in Icelandic. *Snippets* 16, 7f, 2007

INTRODUCTION

In this thesis I try to clarify the role that constraints may play in a syntactic theory. The questions I address include:

- Why should there be constraints in syntax?
- How do constraints relate to operations? Are they more/less powerful?
- How can the power of constraints be restricted? Are they affected by locality restrictions?
- Are there subclasses of constraints with particularly interesting properties? Should only specific types of constraints be used?
- Can constraints be efficiently computed? If so, how?
- Should semantic notions such as logical entailment be part of syntactic constraints?

These and related issues have been explored in the generative literature for many years. What sets this thesis apart is its mathematical and computational grounding. While I approach the topic from a linguistically informed perspective, I do not rely on empirical case studies or arguments of conceptual simplicity. Instead, I use mathematical models that allow me to establish the fundamental properties of constraints and prove them in a rigorous manner.

The mathematically disinclined need not worry, though, as I have made a deliberate attempt to present the material in as approachable a manner as possible. Proofs and formal definitions can be skipped, and all important concepts and results are illustrated with examples. Advanced material that is not essential for understanding the main ideas is explicitly marked with the symbol ☠. Only some set-theoretic notation and the basics of first-order logic go unexplained in the main text, but these

are carefully explained in Appendix A. Readers who wish to explore every nook and cranny of the thesis should also have some basic familiarity with notions from formal language theory such as regular and context-free string grammars and finite-state automata.

By carefully working through the thesis, the reader will get to see Minimalist syntax and constraints from a new perspective. They will understand the importance of derivation trees and their close relation to the Minimalist feature calculus. It will also become clear why syntactic constraints can be computed with a finitely bounded amount of working memory and why this fact entails that these constraints can be expressed purely via Merge. On a technical level, the reader will acquire an intuitive understanding of many advanced topics from mathematical linguistics such as Minimalist grammars, mild context-sensitivity, weak and strong generative capacity and their role in evaluating grammar formalisms, tree languages and tree transductions, monadic second-order logic, and Thatcher's theorem.

The thesis is designed in a modular way so that readers can skip parts that they are not interested in. The first two chapters are dedicated to Minimalist grammars, which provide the formal model of Minimalist syntax used throughout this thesis. Chapter 1 provides both an intuitive presentation of the formalism and a separate, more rigorous treatment. Either section provides the necessary background to follow my claims, but only the latter provides the details that are necessary to follow the proofs. The reader may freely pick one of the two depending on his mathematical comfort level and prior familiarity with Minimalist grammars. Chapter 2 also comes in two parts. The first one introduces the reader to the computational properties of MGs, whereas the second one discusses to what extent Minimalist grammars are an adequate model of Minimalist syntax — an issue that is crucial to evaluating the relevance of my results to syntacticians. Neither section is absolutely necessary for the later chapters, but in particular the first one introduces various points that will be referenced later on, so they are recommended reading.

The second part of the thesis is dedicated to the mathematical exploration of constraints in Minimalist grammars. Chapter 3 covers the central results of this thesis: monadic second-order logic as a description logic of constraints, the expressive equivalence of Merge and constraints definable in this logic, the equivalence of constraints over representations and derivations, and the fact that local constraints are just as powerful as locally unbounded ones. The first equivalence result follows from the fact that monadic second-order definable constraints can be computed by finite-state tree automata. The states of these automata can be pushed directly into the category and selector features checked by Merge, so that Merge implicitly enforces these constraints during the structure building process. Chapter 4 then extends this equivalence result to transderivational constraints by treating them not as filters, but as rewriting rules that turn suboptimal derivations into optimal ones. This is formally captured by modeling them as linear tree transducers.

For syntacticians, the shortest path through this thesis consists of Sec. 1.1, 3.1, and 3.2.2, optionally supplemented by 2.1.1, 2.3, the rest of 3.2, and 3.3. If their primary interest is transderivational constraints they may skip 3.1 and move directly to Chap. 4 after Sec. 3.2.2. However, I highly recommend working through 2.1.1 in this case.

Formally minded readers can follow the recommendations for linguists except that Sec. 1.1 is replaced by 1.2. They might particularly appreciate that the definition of Minimalist grammars uses a two-step format that treats a grammar as a regular derivation tree language plus a mapping from derivations to phrase structure trees. This approach has yielded a lot of interesting insights in recent years but has not been discussed in a widely accessible manner yet. Similarly, Sec. 2.1 covers many technical aspects of MGs only found in specialized literature such as the relation of Minimalist derivation tree languages to bottom-up tree automata and context-free string grammars, the complexity of the mapping from derivations to phrase structure trees, their strong generative capacity, and the importance of remnant movement for

pushing MGs beyond the realm of context-free string languages.

Part I

Setting the Stage

CHAPTER 1

Minimalist Grammars

Contents

Minimalist grammars (MGs) provide the rigorous foundation on which all results of this thesis rest. A good intuitive grasp of the formalism is indispensable, and basic familiarity with the underlying mathematics is helpful in appreciating some of the finer points discussed in Chap. 3 and 4. This chapter provides as accessible an introduction to these issues as possible.

Section 1.1 presents MGs in an intuitive fashion that avoids mathematical notation without sacrificing any details of the formalism. Extensive use is made of

practical examples to illustrate abstract ideas. Differences between MGs and Minimalist syntax that are common sources of confusion are also pointed out prominently.

Section 1.2, on the other hand, defines MGs with all the mathematical rigor necessary to support the results proved in the later chapters. Mathematically inclined readers may prefer to start with this section right away, consulting the previous two only where clarification is needed. This section might also be of interest to MG researchers because it presents the fairly recent two-step approach to MGs (cf. Kobele et al. 2007; Mönnich 2006, 2007; Graf 2012b,c). That is to say, MGs are defined in terms of their derivation tree languages and a mapping from derivations to multi-dominance phrase structure trees.

This chapter is followed by further information on the formal properties of MGs as well as a detailed evaluation of the linguistic faithfulness of MGs in Chap. 2. Impatient readers may choose to proceed to Chap. 3 right away, where I show that MGs can be enriched with constraints on derivation trees and phrase structure trees without increasing their generative capacity.

1.1 Minimalist Grammars: The Intuition

MGs were originally defined in Stabler (1997) as a formalization of Chomsky's (1995c) early version of Minimalism. As is often done in these cases, Stabler implements only the core aspects of Chomsky's highly detailed proposal, in the hope that mathematical results will be easier to establish this way but can later be shown to carry over to more faithful extensions. Whether this strategy has been successful will be discussed in the next chapter (Sec. 2.3). For now, let us set the question of faithfulness aside and focus just on what Stabler's formalization of Minimalism looks like.

MGs can be described in broad strokes as Minimalist syntax with a more stringent feature calculus:

1. Lexical items (LIs) consist of a (possibly null) phonetic exponent and 1 or more features.

2. Features come in two polarities, and both Merge and Move delete exactly two features that differ only in their polarity.

3. Each LI's features are linearly ordered and must be checked in that order.

4. Besides checking features, Merge and Move also build phrase structure trees in the familiar way.

5. The Shortest Move Constraint blocks every configuration where more than one phrase can be moved in order to check a given feature.

The following subsections expand on these points, starting with 1–3, which jointly make up the MG feature calculus. After that, structure building and the Shortest Move Constraint are discussed in greater detail. I deliberately proceed at a leisurely pace, making use of examples and highlighting subtle differences between MGs and Minimalist syntax. The examples consist mostly of single trees or short toy grammars; a fully developed MG grammar for a fragment of English can be found in Chap. 2 of Kobele (2006), though. Readers who prefer a brisker pace are advised to skip ahead to Sec. 1.2 and consult the examples provided here for clarification.

1.1.1 Feature Calculus

MGs require a modest amount of book-keeping regarding which features trigger certain operations at a given point in the derivation. In the spirit of the Borer-Chomsky Conjecture (Borer 1984; Chomsky 1995c), features are the fuel for the entire MG machinery, and variation between grammars are merely variations in feature specifications of LIs. Consequently, every MG can be given as a list of feature-annotated LIs.

While their reliance on features endows MGs with sufficient rigor to support mathematical reasoning, it is also a common source of estrangement for practicing Minimalists, who are used to a high-level style of analysis where not every feature licensing a particular step in the derivation is given explicitly. As a matter of fact, many questions about features are seldom discussed in the literature, e.g. whether an LI may have the same feature multiple times (cf. Nunes 2000) and how many possible values there are for a given feature (cf. Adger 2006, 2010). Due to their feature-driven nature, MGs have to make specific assumptions about these issues; I refrain from defending them here, for once the reader has a better understanding of the formalism, they will see that little hinges on these details.

Early Minimalism postulates a bifurcation into interpretable and uninterpretable features, only the latter of which of are actually deleted by feature checking. MGs, on the other hand, assign each feature one of two polarities — *positive* and *negative* — and feature checking is tantamount to the deletion of two features of opposite polarity.

Example 1.1 Basic feature checking

According to Adger (2003), a transitive verb like *kiss* has an interpretable V feature and an uninterpretable D feature, while a noun like *pigs* has an interpretable D feature. When *kiss* is merged with *pigs*, the two enter into a checking relation. They only agree on their D features, so these are the only ones that can be checked. The uninterpretable feature is subsequently deleted, but the interpretable feature is still present because interpretable features are never erased from the structure. In the corresponding MG analysis, *kiss* has a negative feature V^- and a positive feature D^+, and *pigs* carries a matching D^-. But when the two are merged, both D^+ and D^- are deleted, not just one of them.

As another example, consider the checking of abstract Case, which involves two

uninterpretable features rather than just one. In Minimalist syntax, the head T hosts the overt subject position and has an uninterpretable Case feature. At the same time, there is also an uninterpretable Case feature on the subject DP, which has been introduced at a lower point in the structure. Since feature checking cannot take place at a distance, the subject is first moved into Spec,TP, after which both Case features can be checked and deleted. In MGs, the features triggering Move are still of opposite polarity: T has a positive Case feature, the DP a negative one.

In a sense, MGs treat all features as uninterpretable and thus subject to deletion. Their distinction between positive and negative polarity features adds another dimension that roughly corresponds to the directionality of the dependency. For instance, a verb licenses the presence of its DP arguments, so its D features are all of positive polarity. Since feature checking involves features of opposite polarities, it follows that each DP argument has a negative D feature. Polarity therefore encodes the distinction between licensor and licensee and is not related to the notions of interpretability or valuation entertained in Minimalist syntax.

Features are further divided into Merge features on the one hand and Move feature on the other. No feature can be both a Merge feature and a Move feature, so there is no uncertainty as to which operation will be invoked in order to check a feature. This is not the case for Minimalist syntax, where D-features may be checked via Merge in the case of a DP being selected by a verb, or via Move when a DP has to check the so-called EPP-feature of a T head. MGs do not allow for this kind of overlap and partition the set of features into four subclasses according to their polarity and operation type:

	Merge	Move
-	category feature f	licensee feature $-$f
$+$	selector feature $=$f	licensor feature $+$f

This basic system makes it very easy to specify the appropriate feature make-up for various LIs. Note that an LI may contain several occurrences of the same feature, e.g. two selector features $=$D. As pointed out by Kobele (2005), this means that one should not think of MG features in terms of properties, since it makes little sense for an LI to have the same property twice. Rather, features are the building blocks from which LIs are assembled.

Example 1.2 Features as building blocks for lexical items

The transitive verb *kiss* has a category feature V and selector feature $=$D, whereas ditransitive *owe* has two such selector features. The determiner *the*, in turn, has a category feature D and selector feature $=$N, while its wh-counterpart *which* also has a licensee feature $-$wh, which must be checked via Move by a matching selector feature $+$wh on some C-head.

Simply annotating LIs with features in this way is insufficient, though, if it matters in which order arguments are selected. Consider ditransitive verbs that take a CP as one of their arguments, as in *John told Bill that Mary had left him*. Saying that this instance of *tell* carries the features V, $=$D and $=$C is insufficient, because the grammar also needs a way to decide whether the CP or the DP should be selected first. Some kind of order needs to be established between the two selector features.

For cases where a head selects two arguments of different categories in a specific order, syntacticians might be inclined to infer said order from the thematic roles of the arguments. But this requires at the very least a basic Θ-role system, which MGs do not provide. A mathematically simpler solution to the problem is to linearly order

all the features of an LI. Only the first (i.e. leftmost) feature of an LI is *active* and may enter a checking relation with a matching feature of opposite polarity. Once it has been deleted via feature checking, the feature immediately following it becomes active. So if an XP should be selected by LI l before a YP, then $=$X precedes $=$Y in the specification of l. For the MG apparatus, then, an LI consists of a phonetic exponent (denoted ε if null) and a finite string of features as depicted in Tab. 1.1 (the double colon :: visually separates the two components).

the :: $=$N D	pigs :: N	ε :: $=$V $=$D v
the :: $=$N D $-$nom	sleep :: V	ε :: $=v$ $+$nom T
which :: $=$N D $-$wh	kiss :: $=$D V	that :: $=$T C
which :: $=$N D $-$nom $-$wh	owe :: $=$D $=$D V	ε :: $=$T C
's :: $=$D $=$D D	tell :: $=$C $=$D V	ε :: $=$T $+$wh C

Table 1.1: Lexical Items as Combinations of Phonetic Exponents and Feature Strings

The features on each LI regulate the structure-building process and serve in determining grammaticality: in order for a tree to be well-formed, all features must have been checked on all LIs except the category feature of the highest head, which must be a C feature. This is similar to the requirement in Minimalist syntax that grammatical sentences are CPs from which all uninterpretable features have been erased.

1.1.2 Derivations

Although the MG feature calculus is simple, it is often helpful to have a pictorial representation of how features are discharged, and at what point. While there are several ways this can be accomplished, the most intuitive one for linguists is provided by *augmented derivation trees* (proof nets are a viable alternative, see Stabler 1999 and Salvati 2011).

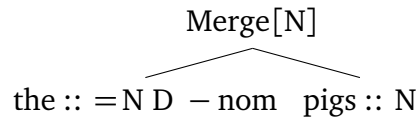
An augmented derivation tree provides a record of the steps taken during the derivation and their relative order: its leafs are annotated with the LIs a given tree

is constructed from, and interior nodes are labeled with the name of the operation that takes place at this point and the name of the two features being checked. The daughters of an interior node are the elements that are combined by the operation the node represents. Note that an augmented derivation tree is actually a strictly binary branching multi-dominance tree because a given subtree might be involved in both Merge and Move.

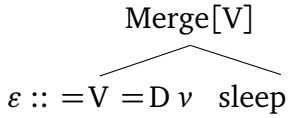
Example 1.3 Augmented derivations

Let us verify that *the pigs sleep* is well-formed with respect to the feature calculus of the MG defined by the lexicon in Tab. 1.1. That is to say, is there some way we can combine LIs in our lexicon to get some structure for *the pigs sleep* that only contains an unchecked C feature on the highest head?

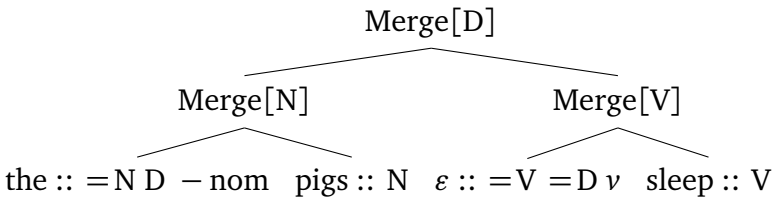
For *pigs* and *sleep*, we have the LIs pigs :: N and sleep :: V, respectively. For *the*, there are two options, the :: =N D and the :: =N D −nom. In either case *the* can select *pigs*, and the features N and =N on *pigs* and *the*, respectively, are subsequently deleted. Consequently there are no features left on *pigs*, while the first unchecked feature on *the* is now its category feature D. The corresponding derivation tree is given below for the case where *the* also carries a licensee feature.



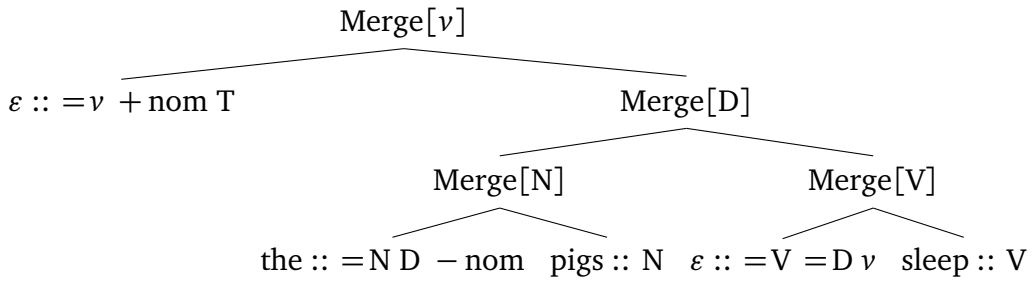
None of the LIs we have considered so far have a matching selector feature, so let us turn our attention to *sleep* now. It only has a category feature, so it, too, needs to be selected by some other LI. Only the empty *v* head may select a verb, so we merge it with *sleep*. This gives us *ε sleep*, which is string equivalent to *sleep*. The derivation tree is similar to the one for *the pigs*.



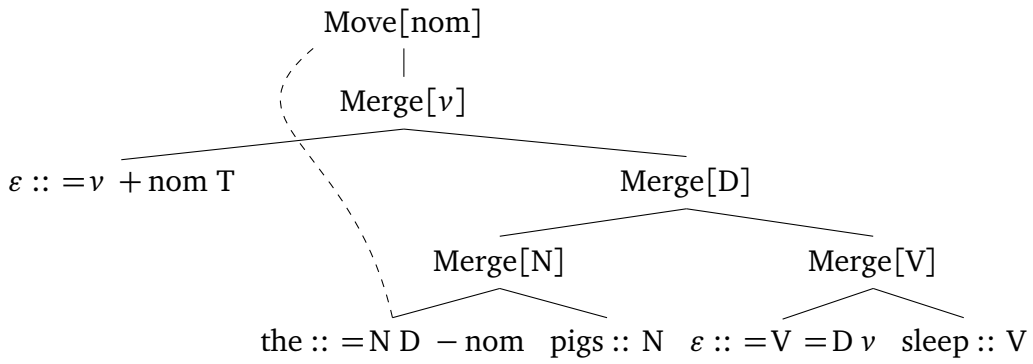
Note that the first unchecked feature the v-head is now $=$D, which matches the first unchecked feature of *the*. So Merge can be applied once more.



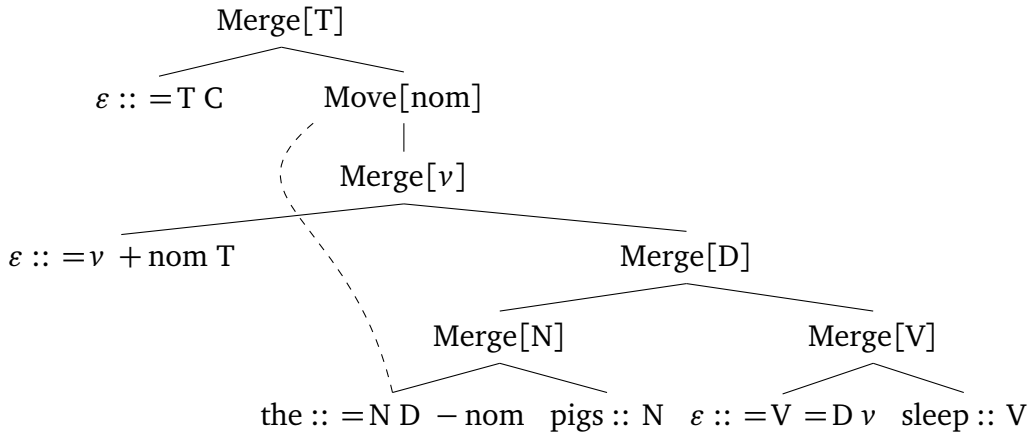
At this point all features have been discharged except for the category feature v and the licensee feature $-$nom on *the*. The LI $\varepsilon :: =v \; +$nom T is the only available selector for v, so it is merged in the familiar way.



The first feature of the T-head is now $+$nom. If none of the LIs we have merged so far had a matching $-$nom feature as its first currently active feature, the grammar would have to abort: Move is not applicable because there is no matching feature for $+$nom, and no further Merge steps are possible because no LI has a category or selector feature as its first active feature. Fortunately we picked an LI for *the* earlier on that carries a $-$nom feature, and this feature is currently active. So Move applies, the nominative features are erased, and T's category feature becomes active. Note that we draw a movement branch in the augmented derivation tree directly to *the* in order to make fully explicit which head hosts the relevant licensee feature.



The last step consists of merging the LI ε :: =T C. The only unchecked feature at this point is C, so the assembled structure is well-formed.



We have not yet talked about how one determines the actual phrase structure tree built by this derivation, but it should be intuitively clear the resulting tree yields the string *the pigs sleep*. Hence there is at least one way of generating this sentence so that all requirements of the MG feature calculus are obeyed.

So far, then, we have seen that MGs build on a symmetric feature calculus — each checking operation deletes two features of opposite polarity — in which each LI is annotated with a string of features that must be checked one after another. The

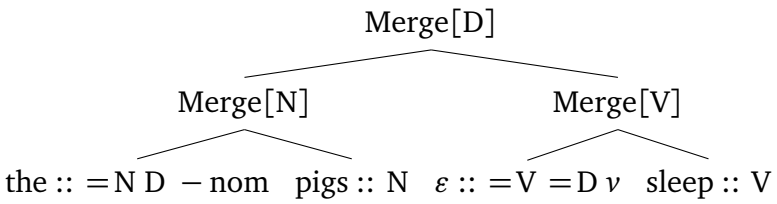
checking is accomplished through sequential applications of Merge and Move, and derivation trees provide an elegant means for keeping track of the order in which the operations are applied. But Merge and Move aren't merely responsible for checking features of a respective type, they crucially build structure in doing so. What, then, are the actual structures generated by MGs?

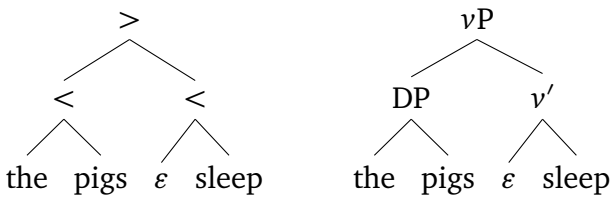
1.1.3 Building Structures

A great advantage of augmented derivation trees is that they make it very easy to define the phrase structure trees produced by Merge and Move. MGs generate a variant of the Bare Phrase Structure trees defined in Chomsky (1995a,c). Heads still project phrases similar to X′-syntax, except that interior nodes are omitted if they have only one daughter. The labeling mechanism used to indicate projection is kept as simple as possible, so that interior labels are merely arrows (< or >) pointing toward the branch along which the projecting head can be found. Ignoring Move and questions of linearization for now, it should be easy to see that the trees generated this way differ only marginally from the augmented derivation trees: LIs are reduced to their phonetic exponent, and interior nodes are relabeled < and > depending on where the projecting head is to be found.

Example 1.4 From derivations to derived trees: Merge

Applying Merge as indicated in the derivation tree below yields the phrase structure tree to the left, which could also be drawn as the tree to the right using a more elaborate labeling algorithm.



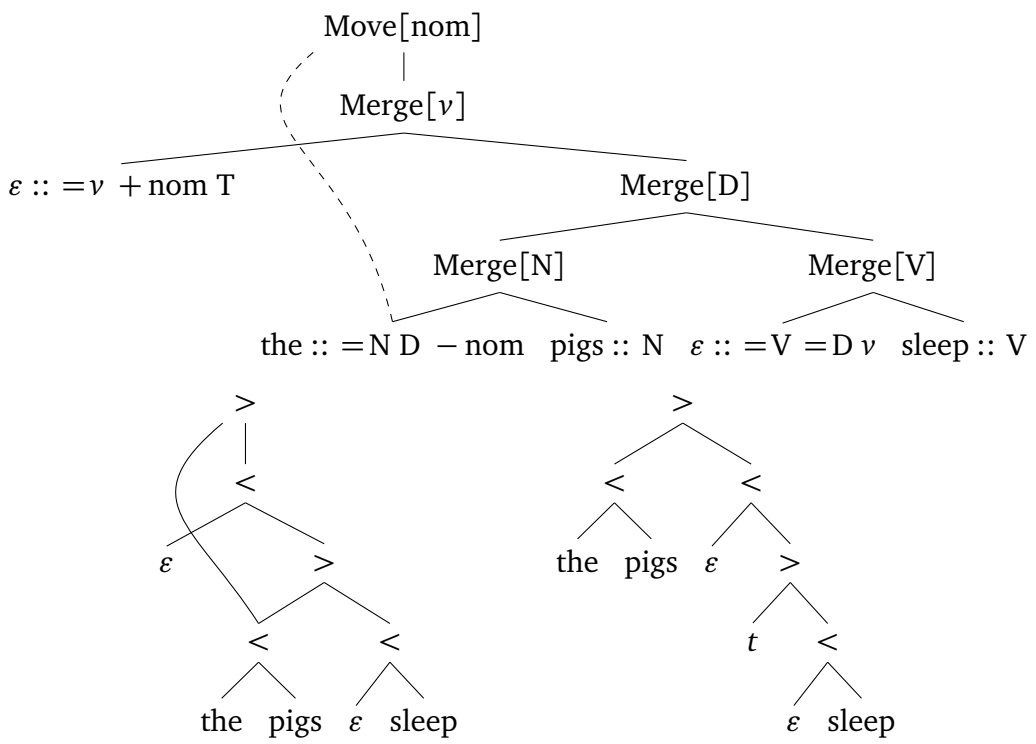


Computing the output of Move follows the same procedure, except that the tail end of the movement branch must be relocated from the LI hosting the licensee feature to the root of the phrase said LI projects — after all, we want the entire phrase to move, not just the LI. This yields the kind of multi-dominance tree commonly entertained in current Minimalist writing.

Multi-dominance trees can easily be converted into standard Bare Phrase Structure trees by detaching the moving subtree from all branches except the movement branch attached to the highest node and optionally attaching traces or copies to any dangling branches created this way (see Kobele 2006 for a rigorous treatment of copying movement).

Example 1.5 From derivations to derived trees: Move

In the derivation below, the movement step yields the multi-dominance tree to the left, or alternatively the phrase structure tree to the right.



This procedure even works in more complicated cases such as remnant movement and roll-up movement. In remnant movement (also known as diving or surfing), a phrase XP from which another phrase YP has been extracted is itself moved across the landing site of YP to some higher position. This yields a configuration in which YP no longer c-commands its original position inside XP. This is blocked by the Empty Category Principle in GB, but Minimalism freely allows for this kind of movement.

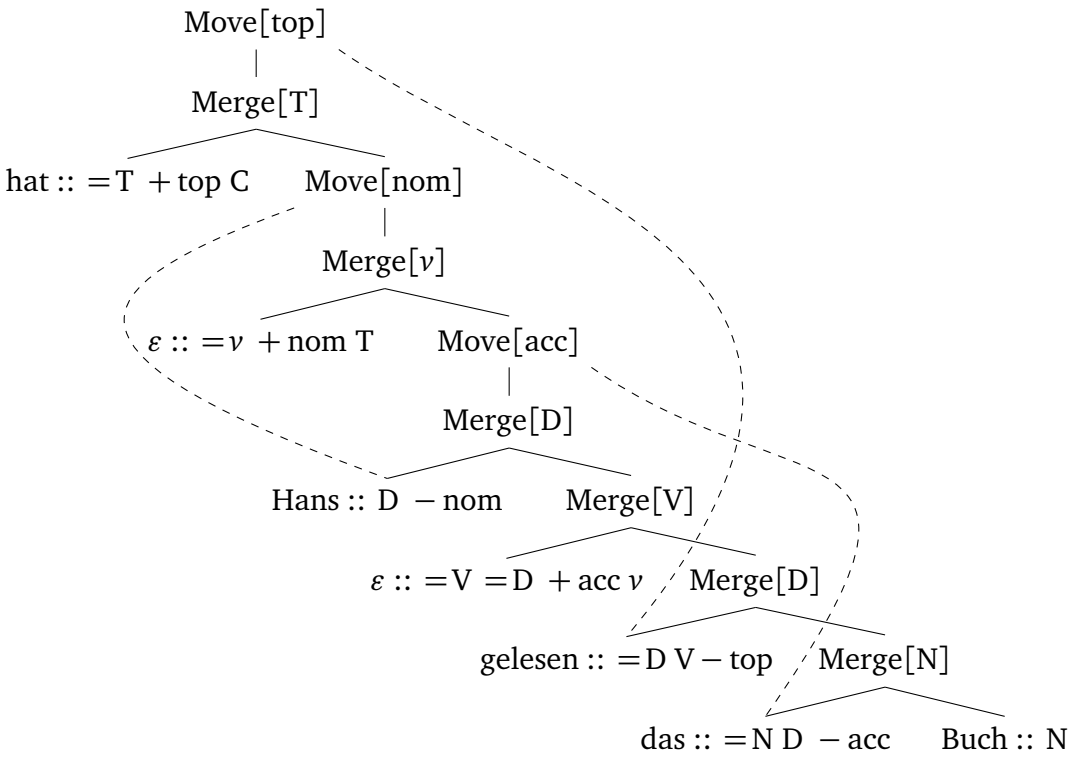
Example 1.6 Remnant movement

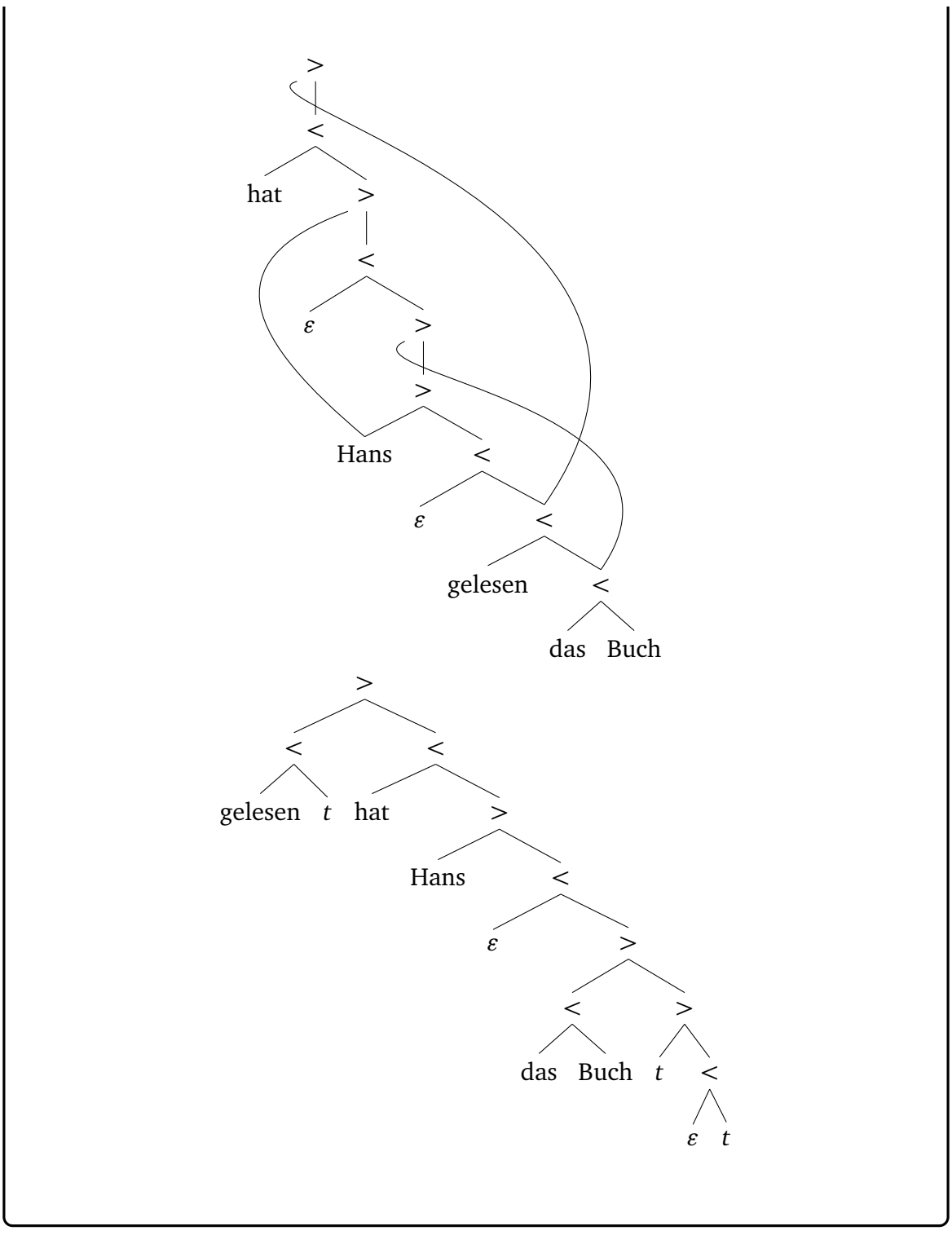
Remnant movement analyses are commonly invoked for cases of verb fronting in German (den Besten and Webelhuth 1990).

(1) $[t_{DP}$ gelesen$]_{VP}$ hat Hans [das Buch]$_{DP}$ t_{VP}.
$[t_{DP}$ read$]_{VP}$ has Hans [the book]$_{DP}$ t_{VP}.

'Hans **read** the book.'

It is commonly assumed that the finite verb in German resides in C^0, and everything preceding it in Spec,CP. By this reasoning, *gelesen* in *Gelesen hat Hans das Buch* must be a phrase despite appearance to the contrary. This is accounted for by having the object move out of the VP, which then moves into Spec,CP — an instance of remnant movement. The MG derivation for this analysis is given below (all usually posited instances of head movement are ignored in this example, as is the head-finality of TP and VP in German).



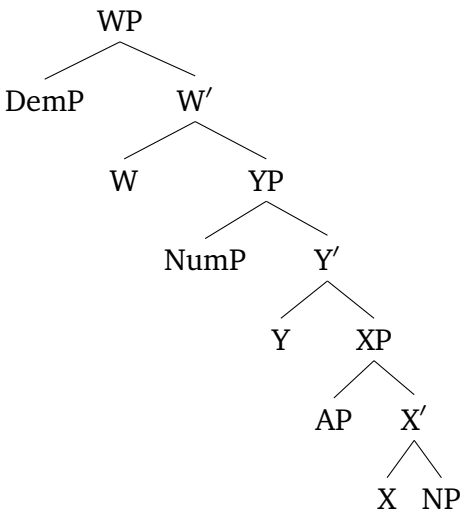


In roll-up movement (also known as snowballing), a phrase XP moves into the specifier of the next higher phrase, which then continues to move on its own,

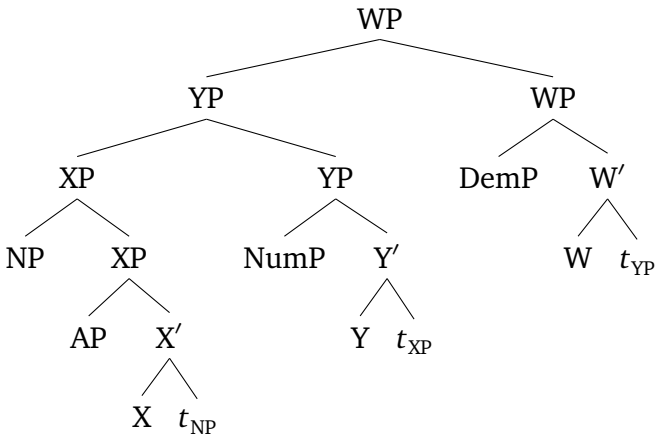
carrying XP along with it. This provides an easy way to invert the order of LIs in a given phrase.

Example 1.7 Roll-up movement

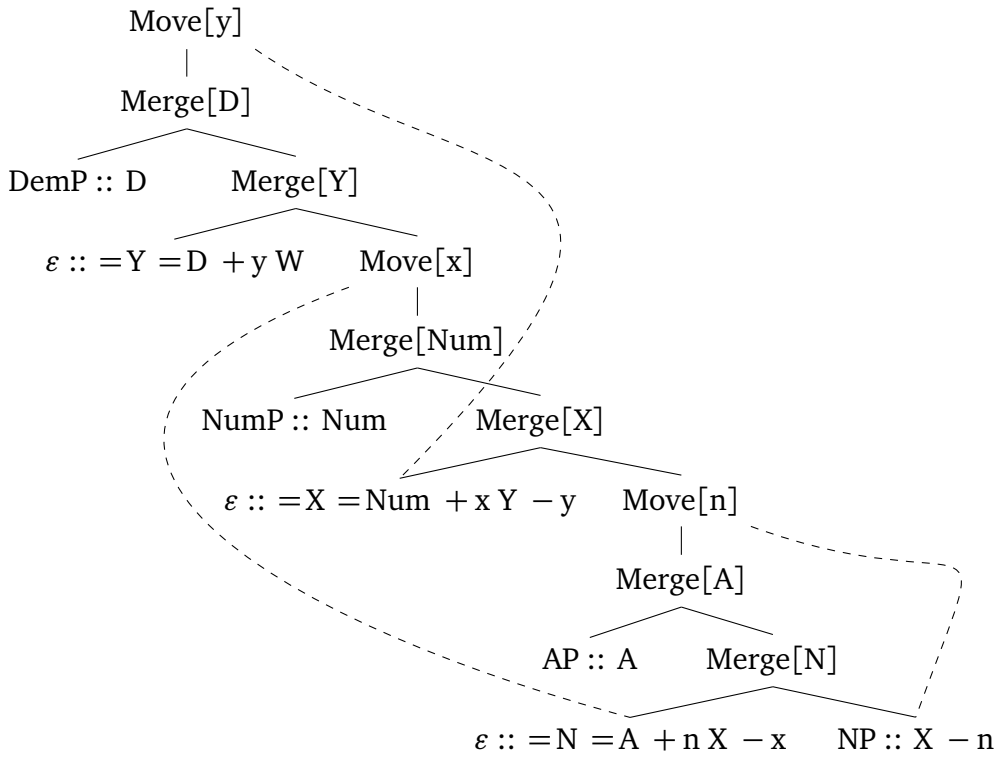
According to Cinque (2005), the underlying order of demonstratives, numerals, adjectives and nouns in a noun phrase is universally fixed by a single template. There are three (possibly empty) functional heads W, Y, and X such that each of them optionally hosts a demonstrative phrase, numeral phrase, or adjectival phrase in its specifier, respectively. Moreover, the noun phrase is the complement of X, which is the complement of Y, which is the complement of W.

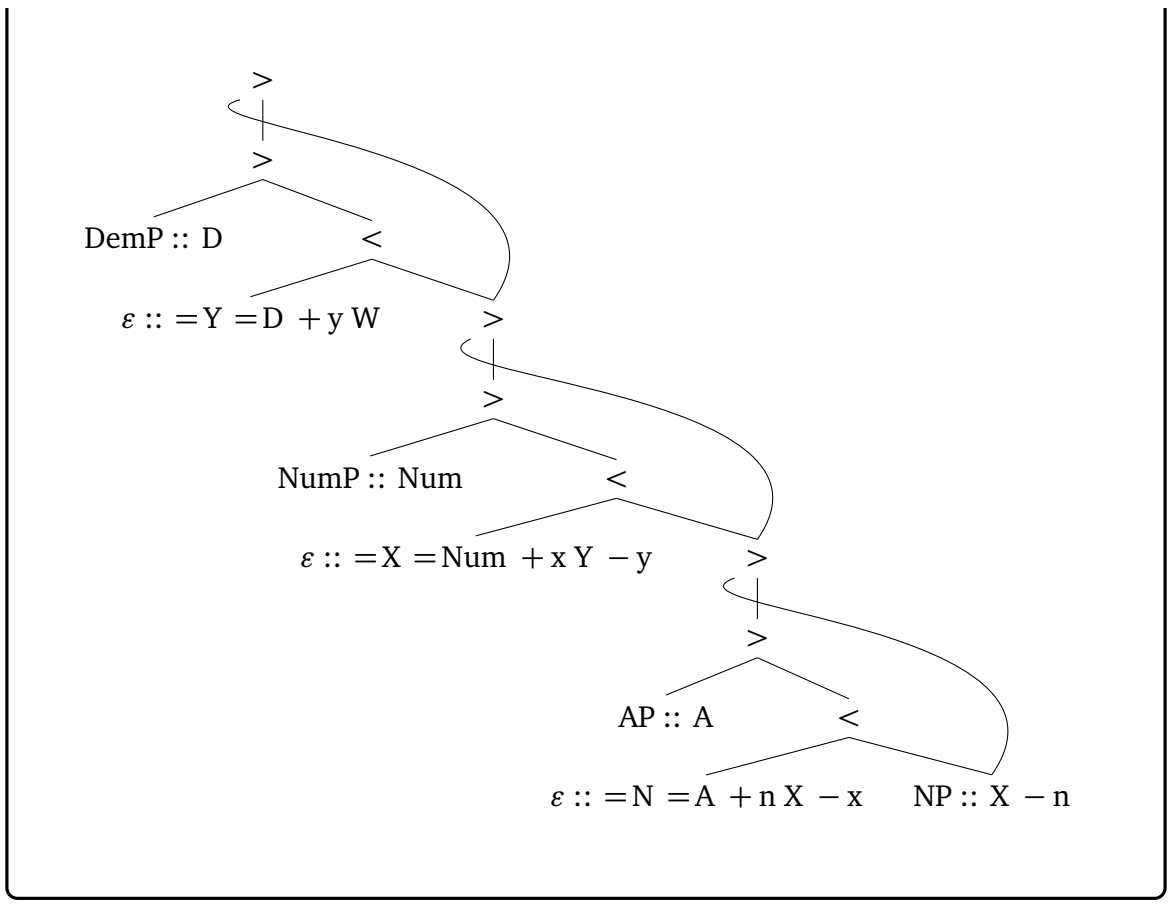


This structure yields the basic Dem-Num-A-N pattern found in English and many other Indo-European languages, cf. *the three bright students*. All empirically attested permutations of the base order are created via Move. The mirror image N-A-Num-Dem is generated via roll-up movement. First NP moves into a second specifier of X, then XP into a second specifier of YP, and finally YP into a second specifier of WP, yielding the phrase structure tree depicted below.



The corresponding MG derivation yields the multi-dominance representation of the same structure.





The examples above demonstrate that it does not matter for our translation from derivation trees to phrase structure trees how many elements move and what their respective timing is — reducing LIs to their phonetic exponent, relabeling interior nodes by < and > to indicate projection, and shifting the tail of movement paths jointly yield the desired multi-dominance tree.

The astute reader might be wondering, though, how < and > can be used to pick out the projecting head because phrase structure trees are commonly taken to be unordered following Kayne (1994). If one cannot distinguish left from right, one cannot use arrows to point in the direction of the head. The answer to this minor conundrum is that the trees generated by MGs are in fact ordered. This is once again done for the sake of convenience because unordered trees are not as well-behaved mathematically (cf. Kepser 2008). However, the assigned order is identical to the

one obtained by the c-command algorithm of Kayne (1994): the first argument of a head is put to its right, all other arguments to its left.

1.1.4 The Shortest Move Constraint

Minimalist syntax is ripe with principles that militate against specific instances of movement if several options exist. For instance, if XP and YP both carry a feature that would allow them to check an uninterpretable feature on ZP via movement, then XP may move to Spec,ZP only if it is not c-commanded by YP. This is supposed to explain why English wh-subjects can always be moved to Spec,CP, but wh-objects only if there is no wh-subject.

(2) a. Who *t* bought the book?

b. Who *t* bought what?

c. What did she buy *t*?

d. ?? What did who buy *t*?

Stabler (1997) steers away from adding such constraints to MGs, once again in an effort to encompass the essentials of Minimalist syntax in as simple a framework as possible (nonetheless the effects of well-known locality constraints on the formalism were a topic of interest from the very beginning; see example 3.1 in Sec. 3.1.1 as well as Gärtner and Michaelis 2007 and references cited there). But the wh-movement patterns above touch on a more fundamental issue that needs to sorted out, *viz.* cases where two or more LIs have the same licensee feature active at the same time in the derivation.

Example 1.8 Move and (non-)determinism

Consider the German remnant movement derivation in example 1.6 on page 18. This derivation is fully deterministic because every LI carries a different licensee

feature. However, if nom and acc were both replaced by the feature case, then subject and object would have the same licensee feature −case. What more, once the subject is merged, both licensee features would be active at the same time. How, then, does one determine which LI checks the +case feature on v, and which one the feature of T?

Rather than establish an elaborate locality algorithm that matches licensee features to licensor features based on structural configurations, Stabler (1997) flat-out blocks all cases of ambiguity via the *Shortest Move Constraint* (SMC).

SMC Two licensee features may both be active at the same time in the derivation only if they are distinct.

Example 1.9 The Shortest Move Constraint

Consider once more the German remnant movement derivation in example 1.6 on page 18, with the minor change that nom and acc have been replaced by case as described in the previous example.

When the verb selects the determiner *das*, the latter's licensee feature becomes active. The verb itself also carries a licensee feature, which becomes active once the verb is selected by v. So at this point in the derivation two licensee features are active. But since one is −case and the other −top, the SMC is obeyed and the derivation may continue. Once the subject *Hans* is selected by v, yet another licensee feature becomes active. This time, though, it is another instance of −case, which violates the SMC. The derivation is subsequently rejected as ungrammatical by the grammar.

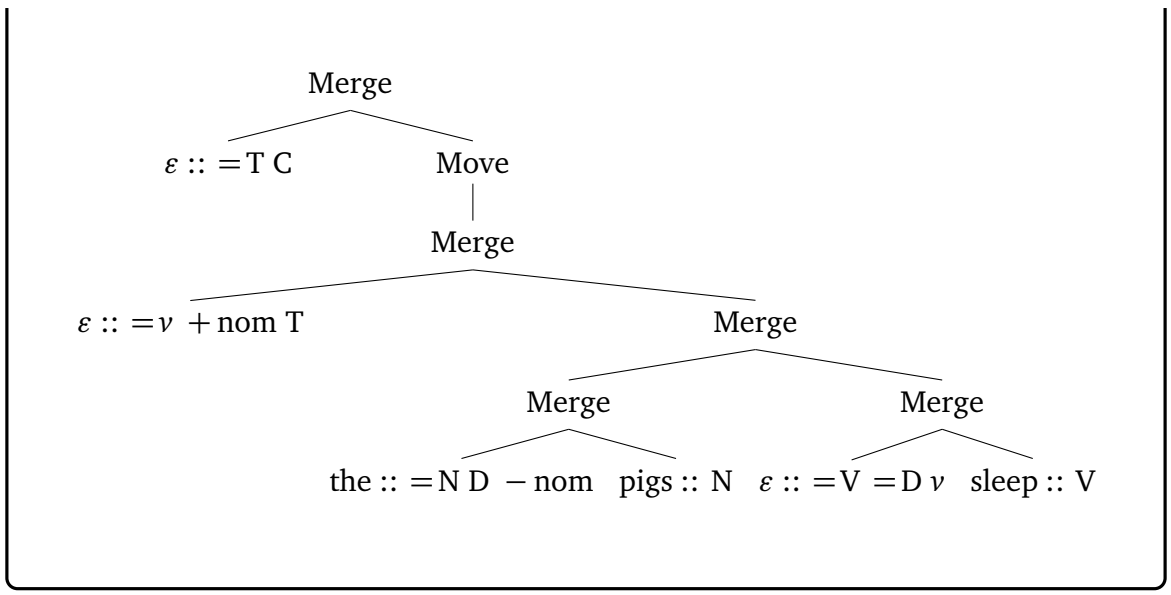
The SMC is a common source of confusion for syntacticians. For one thing, it has little in common with the eponymous constraint from the syntactic literature. The naming of the constraint is indeed unfortunate, but it makes slightly more sense once one realizes that the SMC is motivated by configurations such as (2d), which in turn are closely connected to discussions of locality in the literature.

A more important issue, however, is the abundance of cases in the literature where two or more LIs are taken to be eligible movers. It seems that by adopting the SMC, MGs actively exclude a significant portion of syntactic inquiry. As so often with mathematical work, though, looks are deceiving, and there are ways to weaken the SMC without jeopardizing the framework. Unfortunately, appreciating this point takes some formal background that is not in place yet. The issue will be picked up again in Sec. 2.3.3 and 2.3.4.

In sum, MGs do not impose any locality conditions but instead require that Move be deterministic. Notice that this makes the dashed lines of augmented derivation trees redundant, because one can always deduce from the feature calculus which LI moves where. Therefore I will no longer indicate movement dependencies in the derivation tree. I also drop the features from interior node labels to reduce clutter.

Example 1.10 From augmented derivations to standard derivation trees

Example 1.3 concluded with an augmented derivation tree for *The pigs sleep*. The derivation is repeated below in the simplified format.



Derivation trees play a central role in later chapters, so the reader is advised to familiarize himself with this simple format. It captures the actions of the MG feature calculus in a succinct manner while reflecting the shape of the phrase structure tree *modulo* displacement of phrases via Move. Since derivation trees fully specify both aspects in an elegant manner, they can easily serve as the primary data structure of MGs. That is to say, every MG can be equated with its set of well-formed derivation trees.

1.1.5 Slices

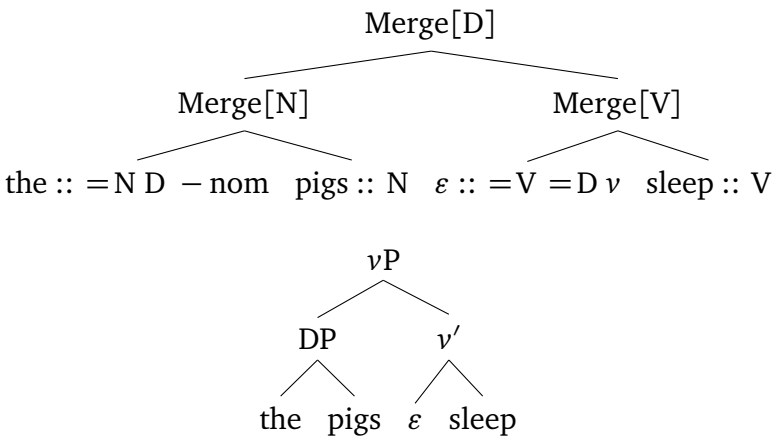
At the beginning of this section it was pointed out that MGs follow the Borer-Chomsky Conjecture in pushing all parametric variation into the lexicon while keeping the properties of Merge and Move as well as the feature calculus constant across grammars. So in order to define an MG, it suffices to list all its LIs. But we have also learned that every derivation specifies a unique multi-dominance tree thanks to the SMC, which makes Move a deterministic operation. Thus every MG is also uniquely specified by its set of derivation trees, which I also refer to as a Minimalist derivation tree language (MDTL). But in contrast to lexicons, MDTLs do

not constitute a very useful way of defining MGs because they might be infinite so that writing down a list of all well-formed derivations isn't feasible (even though it is very easy to compute the set of well-formed derivation trees for any given MG, see Sec. 2.1.1). Fortunately there is a way to decompose derivations into partial derivations such that every MG is uniquely specified by a finite set of such partial derivations.

Just like phrase structure trees can be decomposed into subtrees containing exactly one LI and all its projections, derivation trees can be decomposed into subderivations that consist of an LI and all the interior nodes that check one of the LI's positive polarity features (i.e. selector and licensor features). These subderivations are called *slices*.

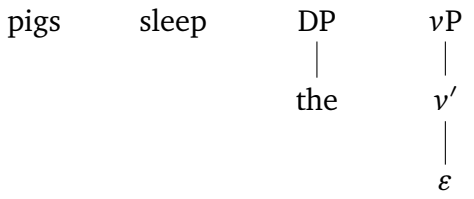
Example 1.11 Slices

Example 1.4 showed a simple MG derivation and the phrase structure tree it generates using a more standard labeling convention. The trees are repeated here for the reader's convenience, and their interior nodes are decorated in the style of augmented derivation trees. This is just an expository device, though, and thus not necessary for slices.

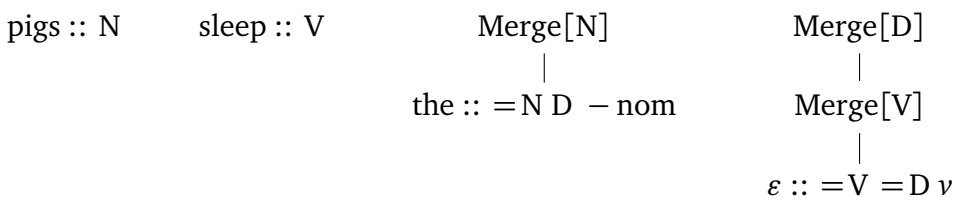


The phrase structure tree can be decomposed into four subtrees such that each

consists of an LI and its projections. Put slightly differently, each subtree corresponds to a maximal phrase with its arguments removed.



Notice that the phrase structure tree and the derivation tree differ only in their labels. Every node in the phrase structure tree corresponds to a specific node in the derivation tree. If the derivation tree is decomposed in a way that reflects the decomposition of the phrase structure tree, one obtains four partial derivation trees, that is to say, four slices.



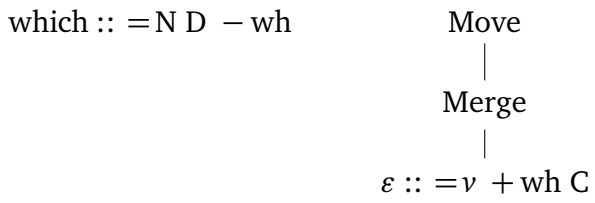
One can tell immediately that the nodes dominating an LI in a slice correspond exactly to its positive polarity features and also reflect their order. Left-to-right in the feature string corresponds to bottom-up in the slice.

Intuitively, slices are the derivational equivalent of phrasal projection. Just like every node in a phrase structure tree is either an LI or one of its projections, every node in a derivation belongs to the slice of some LI. The slice of an LI is readily determined: its leaf is the LI itself, and for every positive polarity feature an interior node of the appropriate type is added on top of the slice — Merge for selector features, Move for licensor features. Hence every Minimalist lexicon can be converted into a set of slices. These slices can be recombined to yield derivation trees,

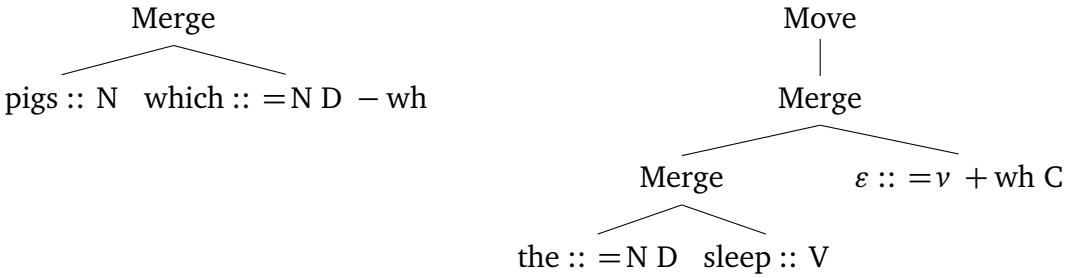
although not all combinations may obey the requirements of the feature calculus.

Example 1.12 Combining slices

Suppose the four slices from the previous example are supplemented by another two involving Move.



These slices can be recombined in a number of ways by attaching a slice as the second daughter of a Merge node. Some combinations respect the feature calculus, many do not.



But even the illicit combinations never include configurations where Move has more than one daughter — only Merge nodes may have a slice attached to them.

An MG's MDTL, then, is the result of I) freely combining all slices in all possible ways such that every Merge node has two daughters, and II) subsequently filtering out all derivations that fail the requirements of the feature calculus. Since slices are obtained from LIs, of which every MG has only a finite number, every MDTL is specified by a finite number of slices, wherefore every MG can be defined as a

finite set of slices. The decomposition of derivations into slices thus combines the succinctness of the lexical perspective with the clarity of the derivational perspective on MGs.

Slices are an essential tool in my investigation of constraints in Chap. 3. In particular, the fact that all derivations are assembled from slices entails that derivations are lexicalized in the sense that every node in the derivation is associated to a positive polarity feature of some LI. Figure 1.1 shows a Minimalist derivation and its implicit partition into slices.

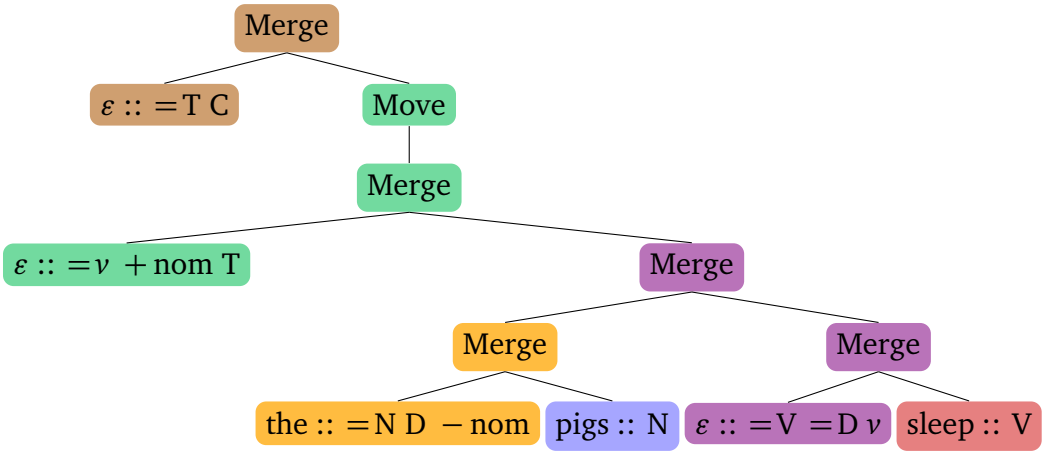


Figure 1.1: Minimalist derivation with slices indicated by color

1.2 Formal Definition

Several equivalent formalizations of MGs have been proposed in the literature, all of which serve slightly different purposes. The original definition in Stabler (1997) is the most intuitive one and casts MGs as a mechanism for combining LIs into (linearly ordered) phrase structure trees. The chain-based definition of Stabler and Keenan (2003), on the other hand, makes MGs a lot easier to work with mathematically at the expense of turning them into a string-based formalism. That is to say, sentences are generated directly from LIs without the intermediate step of phrase structure trees. Finally, the two-step approach to MGs has gained a lot of traction in recent

years (see Kobele et al. 2007, Graf 2012b,c for MGs, and Morawietz 2003, Mönnich 2006, 2012 for two-step approaches to other grammar formalisms).

The idea behind the two-step approach is that each MG is specified by two components: its set of well-formed derivations, and a mapping from derivations to the desired output structures. A mapping to phrase structure trees yields the formalism of Stabler (1997), a mapping to strings that of Stabler and Keenan (2003). Yet another mapping might turn derivations into logical formulas in order to add semantics to MGs. In a certain sense, the derivations become the central object constructed by syntax, while phrase structure trees, prosodic trees, logical formulas etc. are just the interpretation of these objects at the interfaces.

Even though the two-step approach takes a little bit more formal legwork to set up correctly, it is both accessible and more general than the previous approaches. By modularizing MGs into two components — derivations as a tree-geometric representation of the feature calculus and a given mapping as the linguistic interpretation of derivations — it becomes possible to look at each component in isolation. Moreover, both components can be modified independently to create new MG variants, a strategy explored in Graf (2012c). Given these advantages, and because derivations were already discussed extensively in the previous section, I forego alternative definitions and only present Graf's (2012c) two-step definition of MGs here.

I start with the definition of derivation trees as combinations of slices, followed by a number of tree-geometric constraints that rule out combinations violating the feature calculus. The mapping from derivations to multi-dominance trees concludes the definition. Readers who are content with the intuitive presentation so far may skip ahead to the next chapter.

1.2.1 Combining Slices Into Derivation Trees

As discussed in Sec. 1.1, MGs combine LIs into bigger structures via the feature calculus, the actions of which can be represented as (augmented) derivation trees. Graf (2012c) inverts this relation: LIs correspond to incomplete derivation trees, and the tree-geometric constraints on how these partial derivations may be combined implicitly encode the Minimalist feature calculus.

The derivational fragment introduced by an LI is called a *slice*. A slice contains a single LI and all the interior nodes that check one of said LI's positive polarity features. Since these are exactly the nodes that are mapped to a projection of the LI in the derived tree, slices could be called the derivational equivalent of phrasal projection. Slices are obtained from LIs, which are built over a given feature system and a fixed string alphabet (i.e. a finite, non-empty set of symbols).

Definition 1.1. Let BASE be a non-empty, finite set of *feature names*. Furthermore, OP := $\{merge, move\}$ and POLARITY := $\{+, -\}$ are the sets of *operations* and *polarities*, respectively. A *feature system* is a non-empty set $Feat \subseteq \text{BASE} \times \text{OP} \times \text{POLARITY}$.

Note that this is merely a different notation for the familiar MG feature system:

- category features $f := \langle f, merge, - \rangle$,
- selector features $=f := \langle f, merge, + \rangle$,
- licensee features $-f := \langle f, move, - \rangle$, and
- licensor features $+f := \langle f, move, + \rangle$.

In cases where only the name, operation, or polarity of f is of interest, $\nu(f)$, $\omega(f)$ and $\pi(f)$ will be used, respectively.

Example 1.13 Three functions to determine properties of features

A wh-licensee feature is encoded by the tuple $\langle \text{wh}, \textit{move}, - \rangle$. Thus we have:

$$\nu(-\text{wh}) = \nu(\langle \text{wh}, \textit{move}, - \rangle) = \text{wh}$$
$$\omega(-\text{wh}) = \omega(\langle \text{wh}, \textit{move}, - \rangle) = \textit{move}$$
$$\pi(-\text{wh}) = \pi(\langle \text{wh}, \textit{move}, - \rangle) = -$$

As before LIs consist of a phonetic exponent and a non-empty string of features. Here $\textit{Feat}^+$ denotes the set of all non-empty strings over *Feat*.

Definition 1.2. Given a string alphabet Σ and feature system *Feat*, a $(\Sigma, \textit{Feat})$-*lexicon* is a finite subset of $\Sigma \times \textit{Feat}^+$.

I continue to informally make use of :: as a separator of the two lexical components where convenient.

The next step translates each LI into its corresponding slice, i.e. the part of the derivation it has control over by virtue of its positive polarity features. This part is notationally dense, admittedly. A ranked alphabet is an alphabet in which each symbol is assigned a rank. If symbol σ is of rank n, written $\sigma^{(n)}$, the node it is assigned to must have exactly n daughters. Given a ranked alphabet Σ, T_Σ is the set of all possible trees with labels drawn from Σ.

Definition 1.3. Let *Lex* be a $(\Sigma, \textit{Feat})$-lexicon, $\textit{Lex}_\star := \{\sigma :: f_1 \cdots f_n \star \mid \sigma :: f_1 \cdots f_n \in \textit{Lex}\}$, and Ω the ranked alphabet $\{l^{(0)} \mid l \in \textit{Lex}\} \cup \{\text{Move}^{(1)}, \text{Merge}^{(2)}\}$. Then the *slice*

lexicon of Lex is $\text{slice}(Lex) := \{\zeta(l) \mid l \in Lex_\star\}$, where $\zeta : Lex_\star \to T_\Omega$ is given by

$$\zeta(\sigma :: f_1 \cdots f_i \star f_{i+1} \cdots f_n) := \begin{cases} \sigma :: f_1 \cdots f_n \\ \qquad \text{if } f_1 \cdots f_i = \varepsilon \\ \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n) \\ \qquad \text{if } \pi(f_i) = - \\ \text{Move}(\zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) \\ \qquad \text{if } \omega(f_i) = \textit{move} \text{ and } \pi(f_i) = + \\ \text{Merge}(\square_i, \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) \\ \qquad \text{if } \omega(f_i) = \textit{merge} \text{ and } \pi(f_i) = + \end{cases}$$

I follow (Graf 2012c) in stipulating that slices are right branching, but this is merely a matter of convenience — linear order is irrelevant in derivation trees for all linguistic purposes.

Example 1.14 Converting lexical items into slices

Suppose we want to determine the slice of the LI $\varepsilon :: {=}\text{V}\ {=}\text{D}\ {+}\text{acc}\ \nu\ {-}\text{top}$, which is an empty ν that selects a VP, a DP, and undergoes topicalization at some later point in the derivation. In order to obtain its slice, we need to compute $\zeta(\varepsilon :: {=}\text{V}\ {=}\text{D}\ {+}\text{acc}\ \nu\ {-}\text{top}\ \star)$.

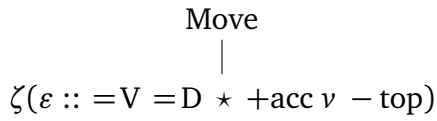
The feature immediately preceding $\star$ is $-$top. Since $\pi(-\text{top}) = -$, the second case in the definition of ζ applies, so

$$\zeta(\varepsilon :: {=}\text{V}\ {=}\text{D}\ {+}\text{acc}\ \nu\ {-}\text{top}\ \star) = \zeta(\varepsilon :: {=}\text{V}\ {=}\text{D}\ {+}\text{acc}\ \nu\ \star\ {-}\text{top}).$$

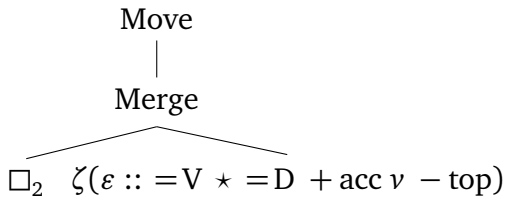
Now v immediately precedes $\star$, but once again $\pi(v) = -$, whence

$$\zeta(\varepsilon :: =\mathrm{V}\ =\mathrm{D}\ +\mathrm{acc}\ v\ \star\ -\mathrm{top}) = \zeta(\varepsilon :: =\mathrm{V}\ =\mathrm{D}\ +\mathrm{acc}\ \star\ v\ -\mathrm{top}).$$

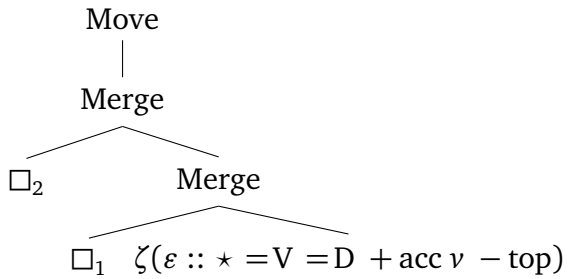
But now the next feature to be evaluated is $+acc$, and since both $\omega(+\mathrm{acc}) = \mathit{move}$ and $\pi(+\mathrm{acc}) = +$, the third case of ζ applies:



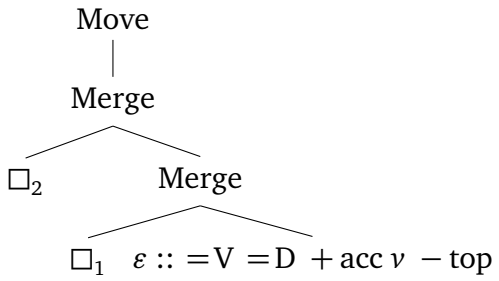
In the next step, we see that $\omega(=\mathrm{D}) = \mathit{merge}$ and $\pi(=\mathrm{D}) = +$, so the tree is expanded by a Merge node accordingly.



The last feature is $=\mathrm{V}$, for which the same procedure applies.



No features precede $\star$ anymore, which triggers the first case in the statement of ζ, the removal of $\star$ from the feature string and the termination of the recursive procedure. The slice of the LI is given below.



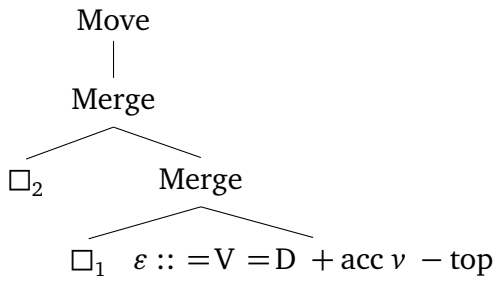
The white squares in a slice are called *ports* and can be replaced by other slices via tree concatenation. Combining the slices of a grammar in all possible ways yields its *free slice language*.

Definition 1.4. The closure of slice(*Lex*) under tree concatenation is the *free slice language* FSL(*Lex*).

Example 1.15 Free slice languages

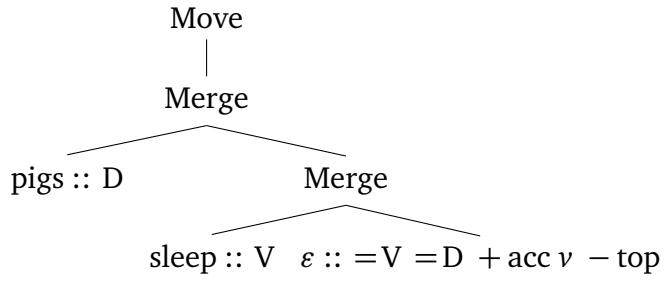
Suppose that slice(*Lex*) consists of three slices:

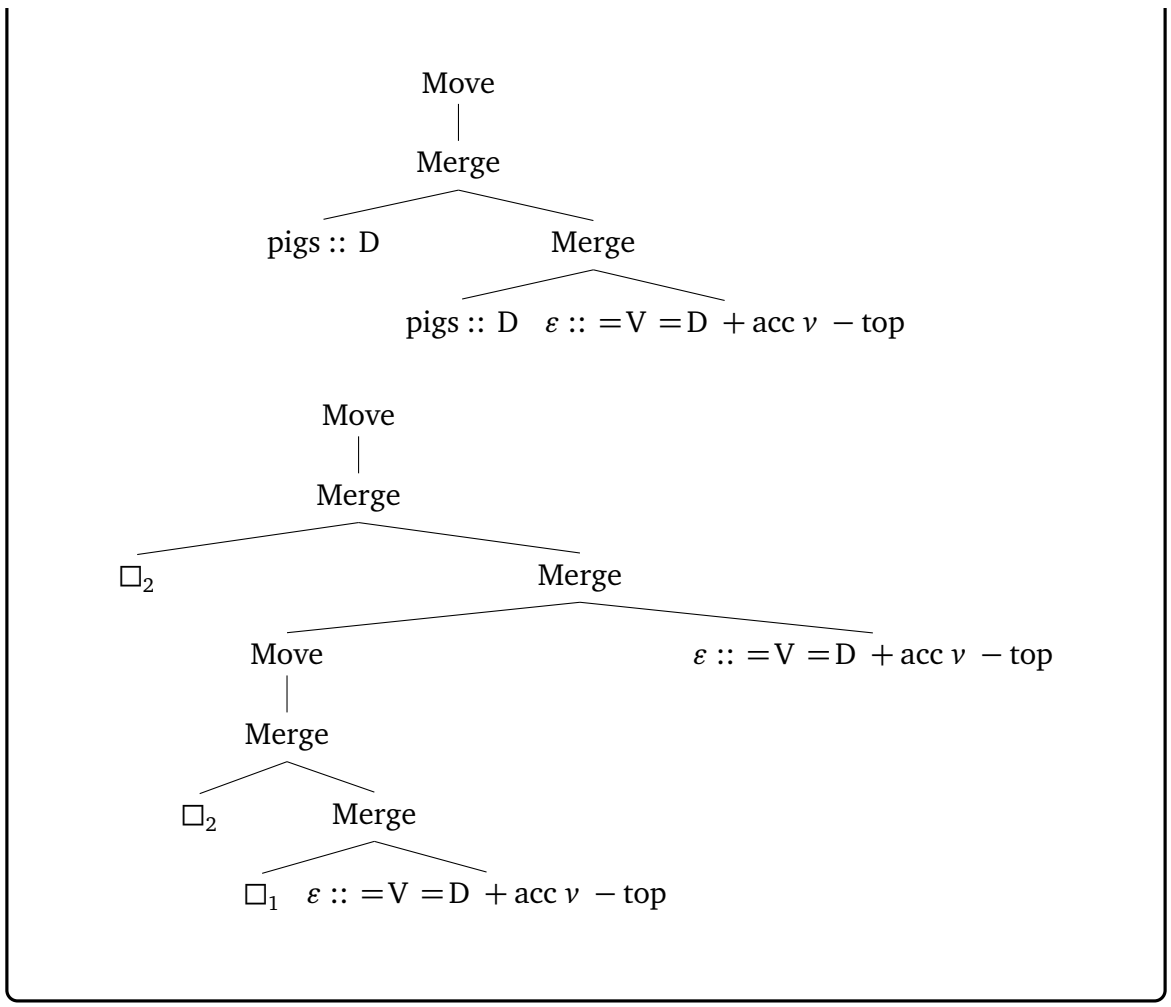
$\mathrm{pigs} :: \mathrm{D}$ $\quad$ $\mathrm{sleep} :: \mathrm{V}$



Then FSL(*Lex*) is an infinite set that contains, among others:

$\mathrm{pigs} :: \mathrm{D}$





Note that FSL(*Lex*) is almost never a well-formed derivation tree language because one possible way of combining a slice is not to combine it with anything. If it contains any ports, these are not filled in this case, and a slice with unfilled ports is not a well-formed derivation. Therefore FSL(*Lex*) is a well-formed derivation tree language iff $Lex \subseteq \Sigma \times \{\langle C, merge, - \rangle\}$, i.e. every LI carries a C-feature and nothing else.

1.2.2 The Feature Calculus as Tree-Geometric Constraints

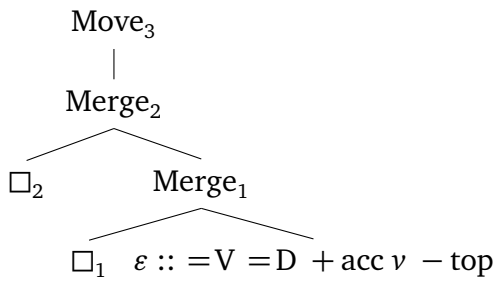
Although not all members of FSL(*Lex*) are well-formed derivations, it clearly contains all well-formed derivations by virtue of containing all possible combinations of slices

in slice(*Lex*). The only thing that needs to be done, then, is to enforce certain constraints on FSL(*Lex*) that are obeyed only by well-formed derivations. These constraints essentially express the MG feature calculus in terms of tree-geometric properties of derivations.

When formulating the relevant constraints, some additional notions come in handy. The *slice root* of LI $l := \sigma :: f_1 \cdots f_n$ is the unique node of $\zeta(l)$ reflexively dominating every node in $\zeta(l)$. An interior node of $\zeta(l)$ is *associated to feature* f_i *on* l iff it is the i-th node properly dominating l. Two features f and g *match* iff they have identical names and operations but opposite feature polarities. More formally, $v(f) = v(g)$, $\omega(f) = \omega(g)$, and $\pi(f) \neq \pi(g)$. An interior node m matches a feature g iff m is associated to a feature that matches g.

Example 1.16 Mapping interior nodes to the features that license them

Consider the slice from example 1.14, with interior nodes subscripted for easier reference.



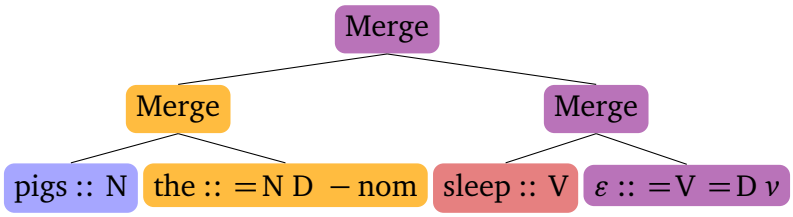
In this slice, Merge_1 is associated to the feature $=\text{V} = \langle V, merge, + \rangle$ and thus matches any occurrence of the feature $\text{V} = \langle V, merge, - \rangle$. Merge_2 and Move_3, respectively, are associated to $=$D and $+$acc and match D and $-$acc. Note that the features v and $-$top on the LI are not associated to any nodes as they have negative polarity.

Keeping aside Move for now, there are two conditions that a Minimalist derivation must satisfy. First, the derived tree must be a final category, by convention a CP.

In the derivation tree, this means that the highest slice must belong to an LI of category C. Second, Merge must be triggered by matching features. One of the two triggering features is the feature a Merge node is associated to, i.e. a selector feature of the LI whose slice the Merge node belongs to. Since slices are right branching by stipulation, this LI can be found by following the right branch down the Merge node. The other feature, then, must be a category feature that can be found along the left branch. More specifically, it must be the category feature of the LI whose slice root is immediately dominated by the Merge node.

Example 1.17 A tree-geometric characterization of the applicability of Merge

Consider the following fragment of the derivation in Fig. 1.1 on page 31, with the slices altered to be strictly right-branching per the new convention.



The arguments selected by v appear to the left of the respective Merge nodes, so that the matching category features must indeed occur along the respective left branch. It is also easy to see that the category feature must occur on the LI of the slice immediately dominated by the Merge node.

Little formal embellishment is required to precisely state the two conditions as constraints on free slice languages. For every $t \in \mathrm{FSL}(Lex)$, node m of t, and LI l:

Final If the slice root of l is the root of t, then the category feature of l is C.

Merge If m is associated to selector feature $=f$, then its left daughter is the slice root of an LI with category feature f.

Regulating Move is more difficult, but the solution is also more insightful for syntacticians. Just as with Merge, it has to be ensured that every Move node has a matching licensee feature. At the same item, the SMC must also be enforced. Since Move is inherently non-local in MGs, neither condition can be enforced by the simple kind of structural description used for Merge above. What is needed is an algorithm to find the Move nodes checking a given LI's licensee features.

Several obvious observations can be made. Movement is always upwards, so the Move nodes checking an LI's licensee features must all dominate it. Furthermore, no LI can check its own licensee features — only one feature can be active on a single LI, while checking involves two features and thus two distinct LIs. Consequently, the relevant Move nodes dominate not only the LI, but also its slice root.

Now suppose that l is an LI with $-f$ as its first licensee feature, and m is the lowest matching Move node properly dominating the slice root of l. Can we conclude, then, that m checks $-f$ on l? In a well-formed derivation, this is indeed the case. For if m does not check l's licensee feature, it must be checking the $-f$ licensee feature of some distinct LI l'. But this inevitably results in an SMC violation because l's licensee feature is already active (if l has any licensee features, they occur immediately after the category feature, so the first one among them becomes active once l is selected). It follows therefore that the lowest Move node properly dominating an LI's slice root and matching its first licensee feature is the only possible option for checking said licensee feature.

It should be clear that the logic of the argument above applies recursively to all other instances of Move, except that the Move node checking the LI's n-th licensee feature not only dominates the slice root of the LI, but also the Move node that checked the $n-1$-th licensee feature. This means that the Move nodes checking the licensee features of a given LI can be easily determined in a recursive fashion: Given an LI l, start at the slice root of l and move upwards through the derivation tree until you encounter a Move node associated to a feature that matches l's first

licensee feature. If l has another licensee feature, continue moving upwards until you encounter a Move node that can check this feature. Continue this way for all remaining licensee features of l. Note that picking the first matching Move node encountered this way is the only potentially licit choice given the SMC. This makes it clear that the SMC adds a certain notion of locality to MGs, weak as it might be.

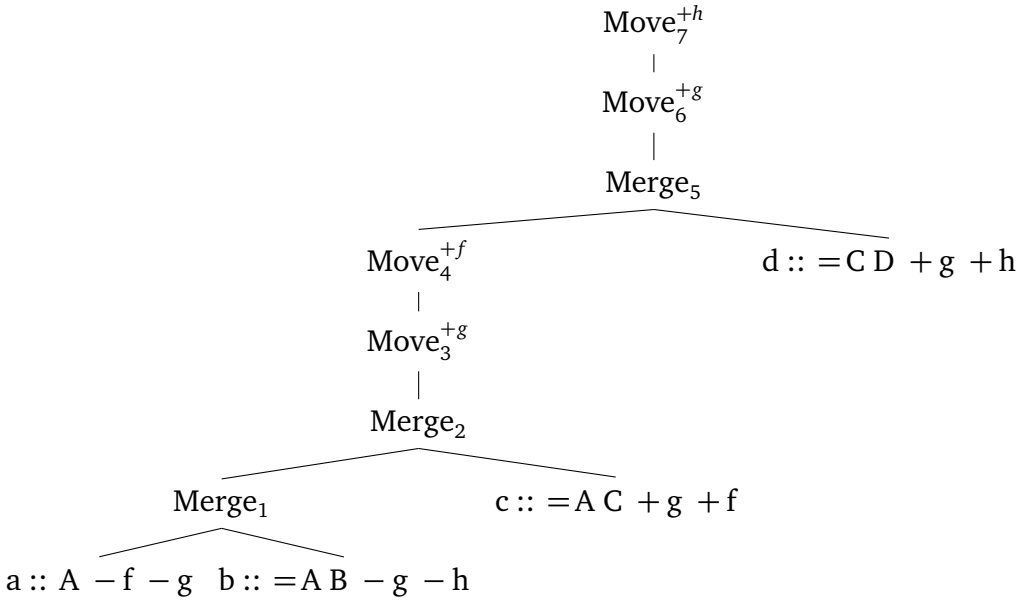
The search procedure above is easily recast in terms of a recursive definition. For every $t \in \text{FSL}(Lex)$ and LI l in t with string $-f_1 \cdots - f_n$ of licensee features, the *occurrences* of l in t are defined as below:

- $occ_0(l)$ is the slice root of l in t.
- $occ_i(l)$ is the unique node m of t labeled *move* such that m matches $-f_i$, properly dominates occ_{i-1}, and there is no node n in t that matches $-f_i$, properly dominates occ_{i-1}, and is properly dominated by m.

I also refer to $occ_0(l)$ as the *zero occurrence of* l, while all other occurrences of l are *positive occurrences*.

Example 1.18 Computing the occurrences of an LI

None of our examples so far included cases where an LI moves more than once. In the derivation below, two LIs undergo two movement steps each. Once again subscripts are used give each interior node a unique name, and each Move node has a superscript indicating the licensor feature it is associated to.



The zero occurrences of LIs a and b are easily determined: $occ_0(a) = a$ and $occ_0(b) =$ Merge$_1$. The first licensee feature of a is −f, wherefore its first occurrence must be the lowest Move node associated to +f that dominates the zero occurrence of a, i.e. a itself. A quick glance suffices to verify that Move$_4$ is the only node fitting these criteria, so $occ_1(a) =$ Move$_4$. The second occurrence must be the closest node that dominates Move$_4$ and matches −g, as Move$_6$ does. So $occ_2(a) =$ Move$_6$. At this point a has exactly as many occurrences as licensee features, so all occurrences of a have been found. Applying the same procedure for b one gets $occ_1(b) =$ Move$_3$ and $occ_2(b) =$ Move$_7$.

Now that the relevant Move nodes are easily picked out via the occurrences mechanism, two natural conditions are sufficient to regulate all aspects of Move: an LI must have as many occurrences as licensee features ("every licensee feature gets checked"), and every Move node is an occurrence for exactly one LI ("checking involves exactly two features"). The SMC does not need to be enforced separately, it

is a corollary of the second constraint given the definition of occurrence.

For every $t \in \text{FSL}(Lex)$, node m of t, and LI l with licensee features $-f_1 \cdots -f_n$, $n \geq 0$:

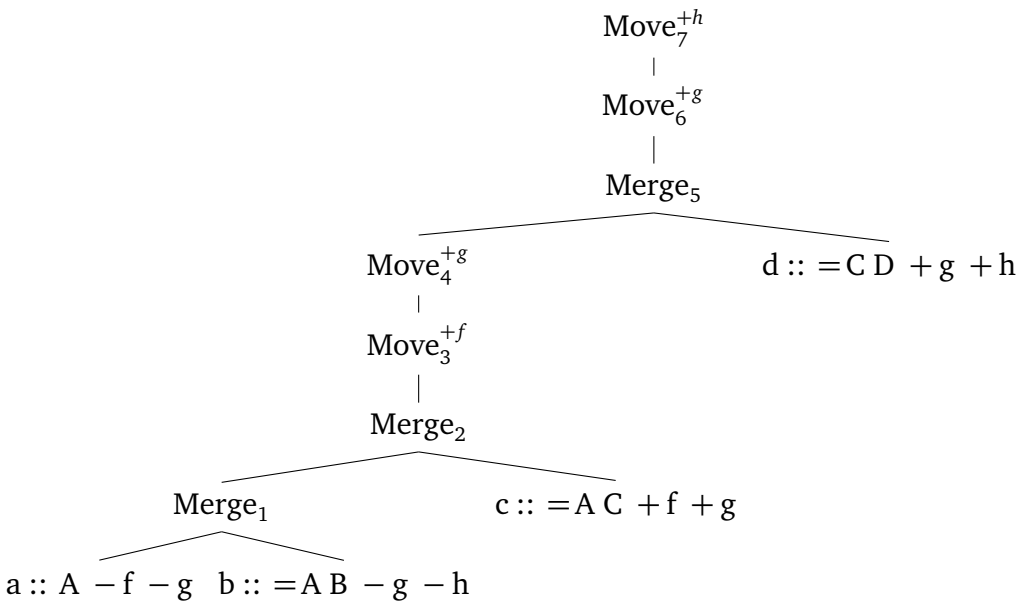
Move There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i^{th} positive occurrence of l, $1 \leq i \leq n$.

SMC If m is labeled *move*, there is exactly one LI for which m is a positive occurrence.

Example 1.19 A tree-geometric characterization of the applicability of Move

Both **Move** and **SMC** are satisfied in the derivation depicted in example 1.18. **Move** holds because both a and b have two licensee features and two occurrences each. Moreover, the occurrences of a and b are unique, so there aren't two Move nodes that are both a first or second occurrence for one of the two nodes. Nor is there a Move node that is an occurrence for both a and b, so **SMC** is also satisfied.

Now consider the minimally different derivation where the order of $+g$ and $+f$ on c is switched.



This derivation clearly isn't well-formed due to a SMC violation: once a gets to check its first licensee feature $-f$ thanks to Move_3, both a and b have $-g$ as their first active feature. Determining occurrences as before, we also see that $\text{Move}_4 = occ_2(a) = occ_1(b)$. That is to say, Move_4 is an occurrence for both a and b, a clear violation of **SMC**. This shows that **SMC** indeed enforces the SMC in a purely tree-geometric fashion.

A combination of slices is a well-formed derivation iff it satisfies all four constraints.

Definition 1.5. A set L of trees is a *Minimalist derivation tree language* (MDTL) iff there is some Minimalist lexicon *Lex* such that L is the greatest subset of FSL(*Lex*) whose members all satisfy **Final**, **Merge**, **Move**, and **SMC**.

Observe the maximality requirement in the definition, which ensures that an MDTL contains all derivations that are well-formed with respect to a given grammar.

1.2.3 From Derivations to Multi-Dominance Trees

As discussed in Sec. 1.1.3, derivation trees differ only minimally from the derived multi-dominance trees. In order to compute the derived tree represented by a derivation one has to

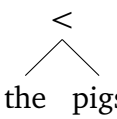
- insert branches for Move, and
- order siblings correctly, and
- relabel interior nodes with $<$ and $>$ to indicate projection, and
- remove each LI's feature component.

Making these ideas precise requires some advanced machinery that won't see much active use in the rest of this thesis, despite its elegance and utility in other areas. However, there are several references to concepts touched upon in this section — in particular during the discussion of formal properties of MGs in Sec. 2.1 and the relation between specific constraint classes in Sec. 3.2.4 — so readers are invited to proceed nonetheless.

Every linguist knows about the use logical formulas in formal semantics for precisely expressing the denotations of LIs and how they combine to yield specific truth conditions. What is less known is that logic can also be used in the area of syntax to specify tree structures. With just a few primitive relations such as immediate dominance $\triangleleft$, precedence $\prec$, equality $\approx$ and monadic predicates for node labels, it is an easy task to describe trees through logical formulas. An entire subfield of mathematical linguistics called model-theoretic syntax is dedicated to studying tree logics and their applications to linguistics and formal language theory (Kracht 1997; Cornell and Rogers 1998; Rogers 1998, 2003; Pullum 2007; Kepser 2008). The model-theoretic approach has a major role to play in Chap. 3 and is discussed in greater detail in Sec. 3.1.3.

Example 1.20 Specifying trees via logical formulas

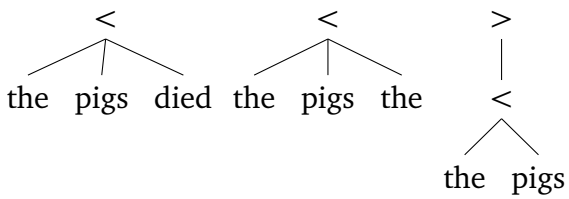
Consider the tree below.



Let the parent relation be expressed by $\triangleleft$, such that $x \triangleleft y$ is true iff x is the parent of y. Similarly, $x \prec y$ iff x precedes y. Then the tree above is described by the following formula of first-order logic:

$$\exists x \exists y \exists z [\text{the}(x) \wedge \text{pigs}(y) \wedge \prec(z) \wedge z \triangleleft x \wedge z \triangleleft y \wedge x \prec y]$$

This formula translates into "There are nodes x, y and z such that x is labeled *the*, and y is labeled *pigs*, and z is labeled $<$, and z immediately dominates x, and z immediately dominates y, and x precedes y." Note that this formula does not uniquely specify the tree above, because it also holds of any bigger tree, such as the following three.



In order to rule out these trees, we need to add another conjunct to the formula: $\wedge\forall z'[z' \approx x \vee z' \approx y \vee z' \approx z]$. Here $\approx$ denotes node identity, so the conjunct states that every node is either x, y, or z — in other words, x, y, and z are the only nodes in the tree.

It is also worth pointing out that the formula above isn't the only one that is true in the tree. For instance, every node is either a mother or a child but not both, so the formula $\forall x\left[\exists y[x \triangleleft y] \leftrightarrow \neg\exists y[y \triangleleft x]\right]$ holds.

Right now our concern isn't the description of trees but of mappings between trees. The specification of such mappings via logical formulas is considerably less common in the literature, even in mathematical linguistics. Suppose that the Minimalist derivation i should be mapped to the corresponding multi-dominance tree o. In Sec. 1.1.3, this was accomplished by rules describing the changes that need to be made to i in order to obtain o. These rules can also be expressed in terms of logics, yielding a logical specification of the translation procedure.

Example 1.21 Rewriting trees via logical formulas

A particularly easy example is sibling permutation. Assume that we want to map the tree i from the previous example to the tree o in which the order of *the* and *pigs* is reversed. To this end, we need two new predicates, which we might call "o-dominance" $\triangleleft_o$ and "o-precedence" $\prec_o$. Then we define $x \triangleleft_o y \iff x \triangleleft y$. This is equivalent to saying that the dominance relations in i are all preserved in o. For o-precedence, on the other hand, we have $x \prec_o y \iff y \prec x$. So x precedes y in o iff y precedes x in i. The corresponding rule could also be expressed as "Switch the order of siblings".

A logically expressed mapping thus consists of two components: a set of relations and predicates that describe the structure of the input trees, and a distinct set of relations and predicates for the output trees. Crucially, the latter must be expressed in terms of the former.

For MGs, several predicates are necessary when talking about derivation trees. As usual, the immediate dominance relation $\triangleleft$ is employed. Moreover, there are predicates for the labels Merge, Move, and every LI in the lexicon. This is all that is needed to fully specify derivation trees in logical terms, but a few ancillary predicates come in handy. First, $occ(x,l)$ iff x is an occurrence of l. Second, $sliceroot(x,l)$ iff x is the slice root of l. Third, $Lex(x)$ iff x is an LI. Finally, $x \sim y$ iff x and y belong to the same slice. See Sec. B.4 and Graf (2012b,c) for how these predicates can be expressed via dominance.

In order to specify multi-dominance trees, dominance and precedence are required. They will be denoted $\blacktriangleleft$ and $\prec$, respectively, and these are the predicates that need to be defined using only the predicates listed in the previous paragraph. The translation procedure from Sec. 1.1.3 offers a guideline for how this is to be accomplished.

The most important modification to the structure is the insertion of movement branches spanning from the slice root of an LI to its occurrences. At the same time, all branches already present in the derivation tree are carried over unaltered.

$$x \blacktriangleleft y \iff x \triangleleft y \vee \exists l \Big[occ(x,l) \wedge sliceroot(y,l) \Big]$$

This formula encodes that x dominates y in the multi-dominance tree iff one of the following two holds: x dominates y in the derivation tree, or there is some LI for which x is an occurrence and y the slice root.

The precedence relation is next. Phrases move to the left and never to the right, so the daughters of Move nodes in the derivation are always to the right, the root of the moving phrase to the left. As for Merge, if a node has a mother labeled Merge that belongs to the same slice, said node is a left daughter only if it is an LI.

$$\begin{aligned} x \prec y \iff \exists l \exists z [occ(z,l) \wedge sliceroot(x,l) \wedge z \triangleleft y] \vee \\ \exists z [\text{Merge}(z) \wedge z \triangleleft x \wedge z \triangleleft y \wedge (x \sim z \rightarrow Lex(x))] \end{aligned}$$

Relabeling the interior nodes is just as simple. Move nodes are always replaced by $>$, as are Merge nodes unless they dominate the LI whose slice they belong to.

$$> (x) \iff \text{Move}(x) \vee (\text{Merge}(x) \wedge \neg \exists y [x \triangleleft y \wedge Lex(y) \wedge x \sim y])$$

$$< (x) \iff \text{Merge}(x) \wedge \exists y [x \triangleleft y \wedge Lex(y) \wedge x \sim y]$$

Finally, LIs must lose all their features but keep their string exponents.

$$\bigwedge_{\sigma \in \Sigma} \Big(\sigma(x) \iff \bigvee_{l ::= \sigma :: f_1 \cdots f_n \in Lex} l(x) \Big)$$

The conjunction of all these formulae yields the intended mapping from derivation trees to multi-dominance trees, which I call Φ_{gr}. While the mapping is relatively simple, it cannot be expressed purely in first-order logic given the predicates we started out with. This is so because defining $occ(x, l)$ requires proper dominance, which cannot be stated in terms of the parent relation $\triangleleft$ in first-order logic. Of course one can easily start out with proper dominance $\triangleleft^+$ and then derive $\triangleleft$ from that via $x \triangleleft y \iff x \triangleleft^+ y \wedge \neg\exists z[x \triangleleft^+ z \wedge z \triangleleft^+ y]$. Alternatively, one might fall back to a slightly more expressive extension of first-order logic in which parenthood and proper dominance are interdefinable.

Monadic second-order logic (MSO) is an excellent candidate for such a step. MSO extends first-order logic with the option of quantifying not only over individual nodes, but also sets of nodes (see Sec. 3.1.3 for examples). With MSO proper dominance is obtained from immediate dominance in two simple steps. First, a set X is closed with respect to immediate dominance iff the children of a member of X are also in X. Then x properly dominates y iff every set that is closed with respect to immediate dominance contains y if it contains x.

$$closed(\triangleleft, X) \leftrightarrow \forall x, y[X(x) \wedge x \triangleleft y \rightarrow X(y)]$$

$$x \triangleleft^+ y \leftrightarrow \forall X[closed(\triangleleft, X) \wedge X(x) \rightarrow X(y)]$$

So depending on which route one takes, the mapping Φ_{gr} from derivation trees to multi-dominance trees can be viewed as a first-order transduction using dominance or an MSO transduction with the parent relation as the most basic predicate. The latter perspective, however, is more useful for mathematical work thanks to several appealing properties enjoyed by MSO transductions, as will be discussed in Sec. 2.1.4.

1.2.4 Formal Summary

There are many different ways MGs can be defined. I opted for a strategy that makes derivation trees the central structure of interest. Derivations provide a tree-geometric representation of the feature calculus and are easily converted into the kind of multi-dominance trees currently favored in Minimalist syntax. The basic building blocks of derivations are called slices, where a slice consists of an LI and all the Merge and Move nodes that are associated to one of the LI's positive polarity features. A derivation is well-formed iff it is a combination of slices respecting the following constraints. For every $t \in \text{FSL}(Lex)$, node m of t, and LI l with licensee features $-f_1 \cdots -f_n$, $n \geq 0$:

Final If the slice root of l is the root of t, then the category feature of l is C.

Merge If m is associated to selector feature $=f$, then its left daughter is the slice root of an LI with category feature f.

Move There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i-th positive occurrence of l, $1 \leq i \leq n$.

SMC If m is labeled *move*, there is exactly one LI for which m is a positive occurrence.

All structural aspects of a given MG are fully specified via its MDTL — i.e. its set of well-formed derivations. The only locus of parameterization is the lexicon, which determines the set of slices and thus the MDTL. In order to define a specific MG one merely has to specify a lexicon.

Definition 1.6. A *Minimalist Grammar* is a triple $G := \langle \Sigma, Feat, Lex \rangle$ such that Lex is a $(\Sigma, Feat)$-lexicon. The MDTL of G is the largest subset of FSL(Lex) that obeys **Final**, **Merge**, **Move**, and **SMC**. The set of multi-dominance trees generated by G is the image of its MDTL under the MSO transduction Φ_{gr}.

A rigorous definition of Φ_{gr} as well as the mapping Φ_{tr} from derivations to phrase structure trees can be found in Sec. B.4.5 and B.4.6.

1.3 The Chapter in Bullet Points

- Minimalist grammars are a bare-bones version of Minimalist syntax with a very explicit feature calculus.
- Both Merge and Move are triggered by symmetric feature checking of two features of opposite polarity. Merge involves selector and category features, Move licensor and licensee features.
- The timing of operations is recorded by derivation trees, and every MG is uniquely specified by its set of derivation trees.
- Each derivation tree is assembled from slices. The slice of an LI contains the LI and every interior node of the derivation that denotes an operation that was triggered by one of the LI's selector or licensor features. Intuitively, a slice is the derivation tree equivalent of the nodes projected by an LI in the phrase structure tree.
- An MG's derivation tree language contains all the possible ways of combining slices such that the constraints of the feature calculus are obeyed.

CHAPTER 2

Minimalist Grammars: Advanced Topics

Contents

This chapter discusses advanced aspects of MGs, first on a technical and then on a conceptual level. None of them are indispensable to follow the gist of the argument in later chapters, but they are conducive to a more profound understanding of the issues at play there. In particular readers who intend to work through any of the proofs are encouraged to read at least Sec. 2.1, which covers the most important formal properties of MGs. The focus is firmly on the complexity of the different types of languages generated by MGs: derivation tree languages (2.1.1), string languages (2.1.2), and phrase structure tree languages (2.1.4). The reader will see that MGs are underlyingly context-free but nonetheless generate mildly context-sensitive string languages. This increase in complexity is brought about by Move, implemented by the mapping from derivations to derived trees. More precisely, it is remnant movement that propels standard MGs out of the realm of context-free grammar formalisms, as is discussed in Sec. 2.1.3. This refactoring of structurally complex patterns into two relatively simple components — context-free derivations and the mapping realized by Move — is what grants MGs their power in spite of their simplicity.

Sec. 2.3 is concerned with the linguistic faithfulness of MGs. It is decidedly less technical in nature, although some arguments necessarily involve advanced concepts. The general upshot is that even though MGs differ from standard Minimalism in various ways, these differences are immaterial for the issues studied in the MG literature — including this thesis. I demonstrate that MGs can easily be modified to make them resemble Minimalist syntax more closely, but I also point out why the bare-bones MGs defined in the previous chapter are a better choice if one desires a genuine understanding of Minimalist syntax.

The two sections can be read independently of each other, but just like the remaining chapters of this thesis, Sec. 2.3 contains some ideas that cannot be fully appreciated without the technical background from Sec. 2.1.

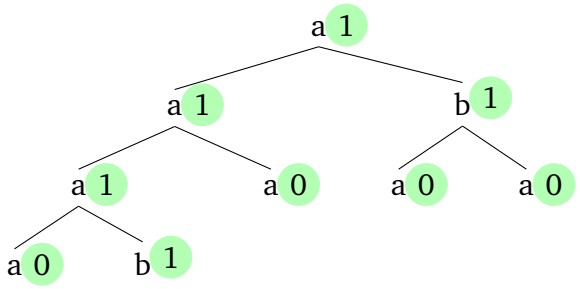
2.1 Selected Formal Results

2.1.1 Derivational Complexity

The MG formalism enjoys several properties that make it very attractive from a computational perspective. Arguably the most important one is that every MDTL forms a *regular tree language*. That is to say, for every MG its set of well-formed derivations can be recognized by a *bottom-up tree automaton*, a device that processes the tree from the leaves towards the root and assigns each node one of finitely many states based on the label of said node and the states that have been assigned to its daughters. If the state assigned to the root is a designated final state, the tree is accepted by the automaton and thus considered well-formed. If a non-final state is assigned, or the automaton cannot assign a state to some node in the tree, the automaton aborts and rejects the tree (see Sec. B.3 for technical details).

Example 2.1 A simple bottom-up tree automaton

Suppose we are given a strictly binary branching tree with nodes labeled either a or b, and we want to determine if the tree contains at least one b. This is a simple task for a tree automaton. The relevant automaton has two states called 0 and 1 that keep track of whether a b has already been encountered or not. The logic driving the state assignment is the obvious one: a node labeled b receives state 1, and if a node has a daughter with state 1, it is also assigned state 1. That way the state 1 is percolated towards the root. Thus, if 1 is a final state but 0 is not, the automaton accepts a tree iff it contains at least one b. The run of this automaton over a particular tree is shown below.



The behavior of an automaton is usually given as a set of *transition rules*. A rule of the form $\sigma(a, b) \to c$ means that a node labeled σ is assigned state c if its daughters have the states a and b, respectively. For leaves this simply shortens to $\sigma \to c$.

Example 2.2 Transition rules of the automaton

The automaton from the previous example is described by ten transition rules:

$$\begin{array}{llll} a \to 0 & a(0,0) \to 0 & b(0,0) \to 1 \\ b \to 1 & a(0,1) \to 1 & b(0,1) \to 1 \\ & a(1,0) \to 1 & b(1,0) \to 1 \\ & a(1,1) \to 1 & b(1,1) \to 1 \end{array}$$

These rules can be represented more succinctly using some basic mathematical notation. Let $Q := \{0, 1\}$ be the set of states of the automaton, and $q, q' \in Q$ two

arbitrary states. Then the above rules can be conflated into four basic patterns.

$$
\begin{aligned}
a &\to 0 \\
b &\to 1 \\
a(q, q') &\to \begin{cases} 0 & \text{if } q = q' = 0 \\ 1 & \text{otherwise} \end{cases} \\
b(q, q') &\to 1
\end{aligned}
$$

Example 2.3 Recognizing trees with an odd number of nodes

Things become more interesting if one further requires that the number of b is a tree is not only strictly greater than 0 but also an odd number. Hence a tree must contain 1 or 3 or 5, ... occurrences of b. This can be accommodated by splitting the state 1 in the previous example into two states o and e for "odd" and "even", respectively. Not only does the automaton now keep track of whether some b has been already encountered, it also performs some basic arithmetic to determine if the number of bs seen so far is odd or even. Only if the state assigned to the root is o does it accept the tree.

As before leaves labeled a have state 0. Leaves labeled b, on the other hand, are assigned o — only one b has been seen so far, namely the leaf itself, and 1 is an odd number. When percolating the information upwards towards the root via state assignments, the automaton has to change between e and o depending on the states of the daughters. For example, a node labeled b with daughter states o and e

is given state e. This is so because the left subtree contains an odd number of bs, the right one an even one, an odd number plus an even number is an odd number, and an odd number of bs plus the one occurrence of b at the current node yields an even number of bs.

The additional complexity brings about an according increase in the number of rules. But the use of meta-rules once again allows for a fairly succinct representation.

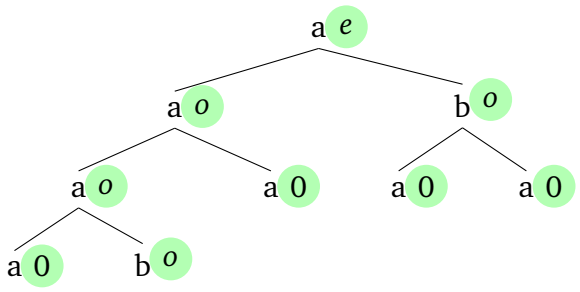
$$a \to 0$$

$$b \to o$$

$$a(q, q') \to \begin{cases} 0 & \text{if } q = q' = 0 \\ o & \text{if either } o = q \neq q' \text{ or } q \neq q' = o \\ e & \text{otherwise} \end{cases}$$

$$b(q, q') \to \begin{cases} o & \text{if } q = q' \\ e & \text{otherwise} \end{cases}$$

This automaton correctly rejects the tree from the previous example, as can be told immediately from the depiction of the run.



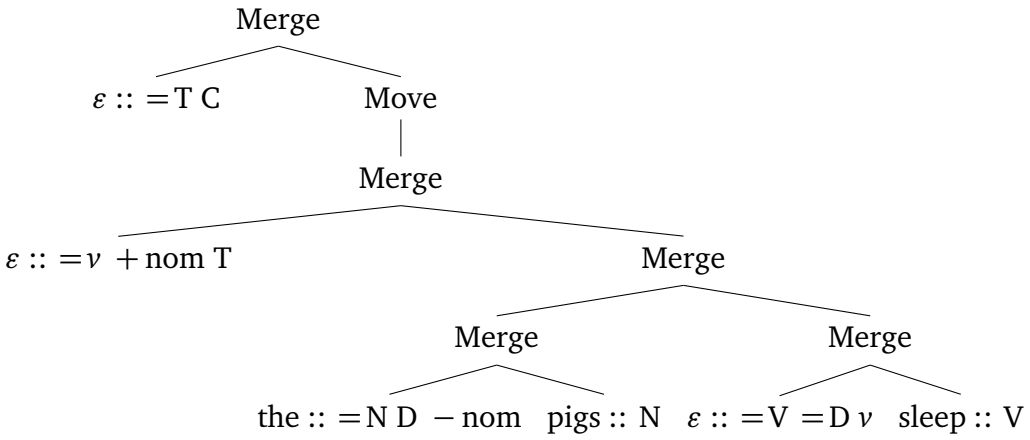
What makes bottom-up tree automata so interesting from a linguistic perspective is that their states are abstract stand-ins for working memory configurations. Conse-

quently, the fact that MDTLs are regular entails that they can be computed using only a finite amount of working memory (by virtue of bottom-up tree automata using only a finite number of states).

There are several viable strategies to show the regularity of MDTLs. Regularity follows immediately from the fact that all constraints and properties invoked in the definition of MDTLs in Sec. 1.2 are easily stated in terms of MSO. This is so because MSO-definability is equivalent to being regular (Büchi 1960; Rabin 1969). Insights from Michaelis (1998, 2001), which were later made fully explicit in Kobele et al. (2007), furnish a more instructive proof, though. In order for an automaton to determine if a derivation tree is grammatical, it must keep track of the computations of the feature calculus. It is a natural idea, then, that the states of the automaton should simply be tuples where each component is a string of features that still need to be checked.

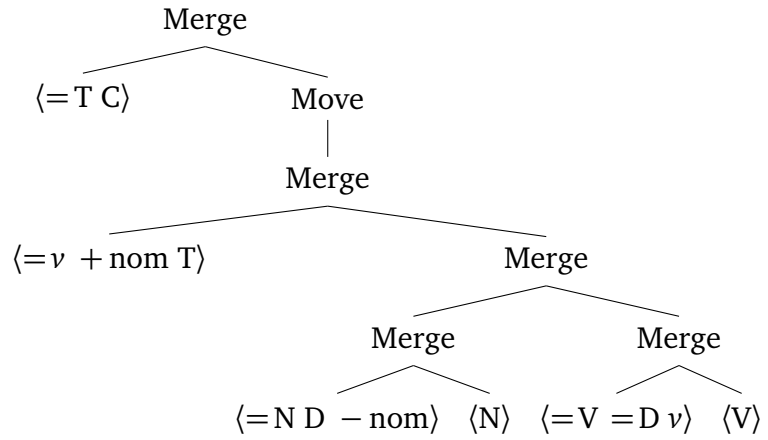
Example 2.4 A tree automaton for Minimalist derivations

Let us return to the familiar derivation of *the pigs sleep* from Sec. 1.1, depicted here as a standard derivation tree rather than an augmented one.

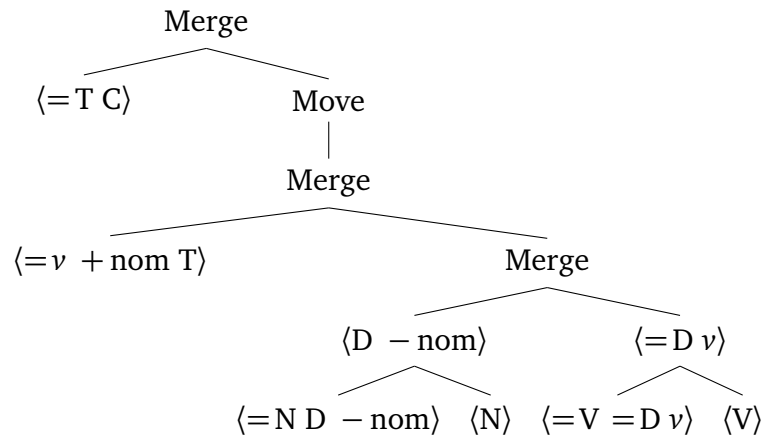


How would a tree automaton process this tree using tuples of unchecked feature strings as its states? Recall that the automaton proceeds bottom-up, meaning that

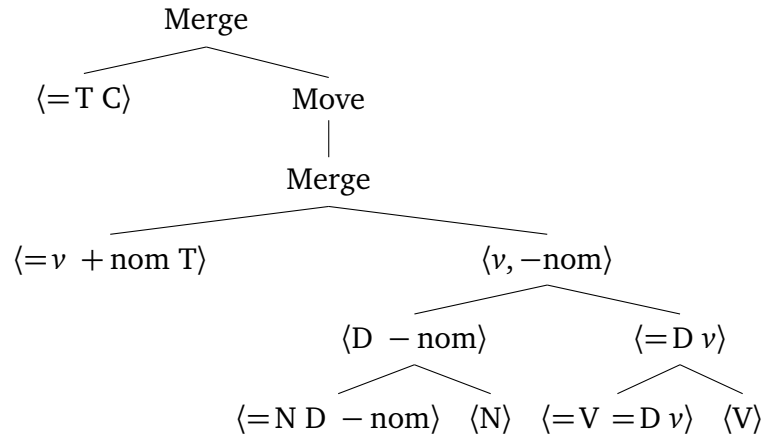
it starts at the leaves. Suppose, then, that *the* is the first leaf to be read. It has the feature string $=$N D $-$nom, and each feature must still be checked at this point in the derivation because no operations have applied yet. So this leaf is assigned the state $\langle =$N D $-$nom$\rangle$. More generally, every leaf is assigned the unique state that is identical to the feature component of the LI the leaf is labeled with. To the automaton, then, the derivation looks as follows after all LIs have been mapped to states.



The automaton now moves up by one level and calculates the new states based on the label of the node and the states of the daughters. For example, the Merge node immediately dominating $\langle =$N D $-$nom$\rangle$ and $\langle$N$\rangle$ receives the state $\langle$D $-$nom$\rangle$. This assignment is correct because the two states start with matching Merge features that can be checked and the node in question is labeled Merge; checking completely exhausts the string of unchecked features in the state $\langle$N$\rangle$ and reduces $\langle =$N D $-$nom$\rangle$ to $\langle$D $-$nom$\rangle$. The Merge node immediately dominating $\langle =$V $=$D $v\rangle$ and $\langle$V$\rangle$ receives the state $\langle =$D $v\rangle$ for the same reason.



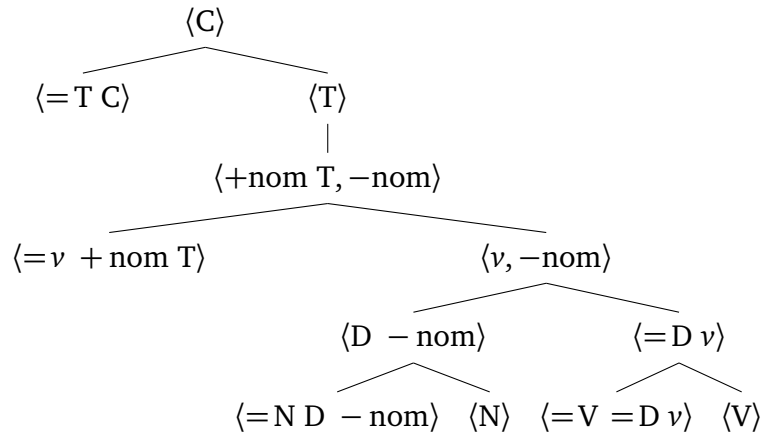
Once again we have two states that start with matching Merge features: $\langle \mathrm{D} - \mathrm{nom} \rangle$ and $\langle = \mathrm{D}\ \nu \rangle$. However, this time neither state is purged of all its unchecked features by Merge. Instead, both $-$nom and ν become active features in the derivation. The automaton has to take note of this by switching from a singleton tuple to a pair $\langle \nu, -\mathrm{nom} \rangle$. The feature ν occupies the first component to indicate that it is carried by the LI of the current slice — i.e. the head of the projected phrase in the corresponding derived tree.



The automaton now has to determine if $\langle -\mathrm{nom}, \nu \rangle$ can be combined with $\langle = \nu\ + \mathrm{nom\ T} \rangle$ via Merge. While the feature $-$nom has nothing to contribute at this

point, v is also active and matches $=v$. So feature checking can take place, yielding the new state $\langle +\text{nom T}, -\text{nom}\rangle$.

The next node is a Move node and we see that two matching Move features are active, resulting in the state $\langle \text{T}\rangle$ following the familiar logic. This state is of course compatible with $\langle =\text{T C}\rangle$, so that the root of the derivation is assigned $\langle \text{C}\rangle$. Recall that MGs deem a derivation well-formed iff all features have been checked except the category feature of the highest projecting head, which must be C. The automaton has to enforce the same condition in order to determine whether a derivation is well-formed. This is accomplished by having $\langle \text{C}\rangle$, and nothing else, be a final state of the automaton. In our case, $\langle \text{C}\rangle$ is the state assigned to the root of the tree, wherefore the automaton deems the derivation well-formed, as intended.



As shown in the example above, it is a simple procedure to annotate derivations with tuples such that each component keeps track of the features of an LI that still need to be checked. But a bottom-up tree automaton is limited to a finite number of distinct tuples, and nothing said so far guarantees that for any given MDTL only a finite number of distinct feature configurations need to be considered — fortunately a quick proof sketch suffices to establish that this is indeed the case.

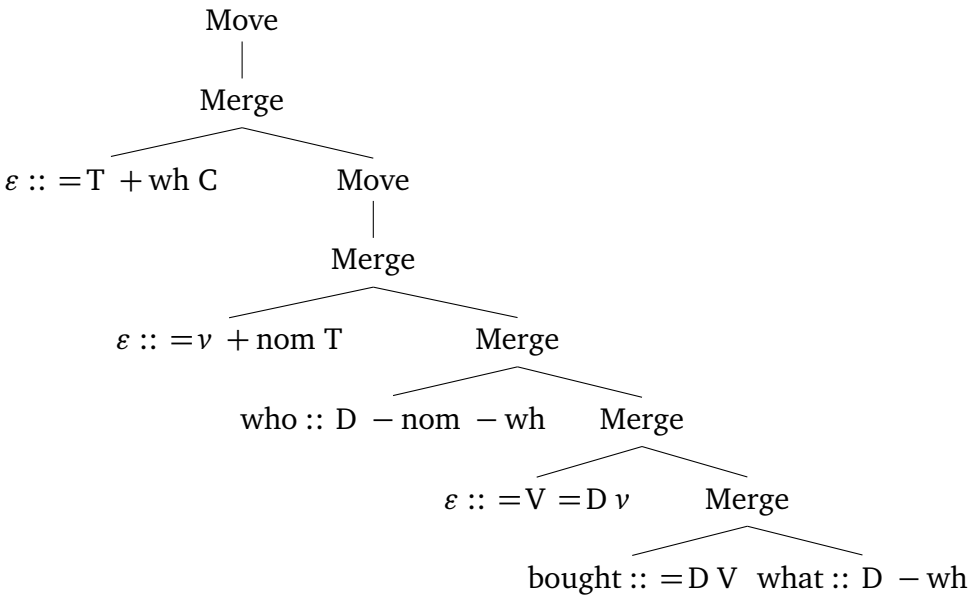
Proof. Suppose towards a contradiction that the number of distinct tuples is infinite. If there is an upper limit on the number of components per tuple, then at least one component can take an infinite number of values. But since each component contains the feature component of some LI or a proper suffix thereof, the number of values is finite by virtue of the finiteness of MG lexicons. Therefore the maximum number of components per tuple must be unbounded given our initial assumption.

It is easy to see, though, that for every MG there is some upper bound k such that no tuple needs to have more than k components. Observe first that if an n-tuple is assigned to a node in a well-formed derivation, all components except the first one start with a licensee feature. If some component after the first one starts with a non-licensee feature, this feature is not carried by the LI of the current slice and hence inaccessible to further operations throughout the derivation. If, on the other hand, the first component starts with a licensee feature, no further Merge steps are licensed yet the state is distinct from the unique final state $\langle C \rangle$. In either case the derivation is ill-formed. Therefore an n-tuple contains exactly $n - 1$ strings that start with a licensee feature, $n \geq 2$. But by the SMC two identical licensee features may not be active at the same time, wherefore every component must begin with a distinct licensee feature. Since the number of licensee feature is finitely bounded, so is the number of components. It follows that every MDTL can be recognized using only a finite number of such tuples. ■

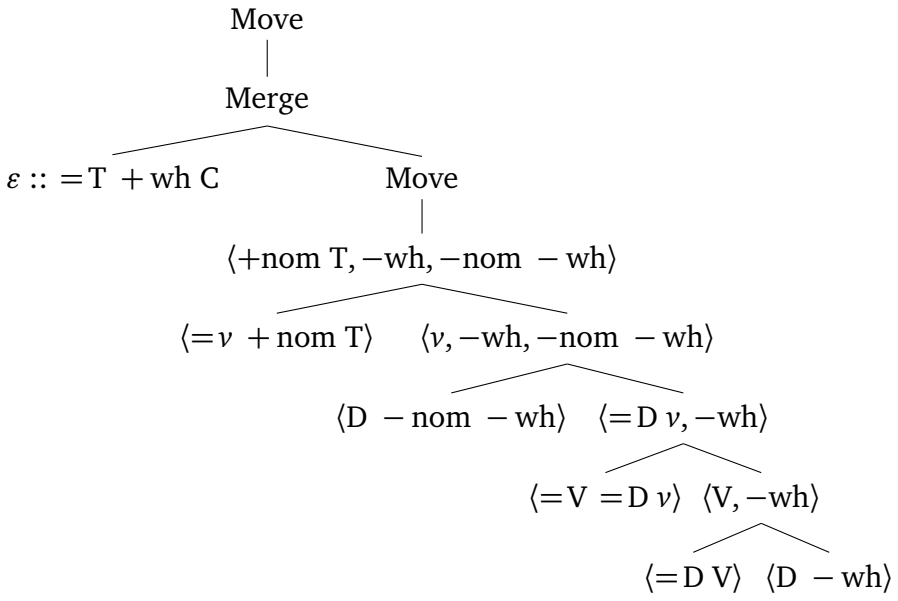
Intuitively, the automaton needs to keep track of only two things: the features on the head projecting the current node, and any LIs in the current subtree that still need to undergo movement. Since the SMC limits the maximum number of LIs that are allowed to move at any given point in the derivation, only a finite amount of information must be recorded in each state. Thus MDTLs are indeed computable with a finitely bounded amount of working-memory, but this appealing property crucially hinges on the SMC.

Example 2.5 Regularity and the SMC

Consider the ill-formed derivation below where both the subject and the object need to undergo wh-movement.



The automaton assigns the states in the familiar way until the first Move node is encountered.



At this point the automaton gets stuck. While checking of the nominative features is possible, this would cause the −wh feature of the subject to become active. Since there is already an active −wh feature on the object, the result is an SMC violation. This is captured by the fact that the automaton would have to assign the state ⟨T, −wh, −wh⟩, which never occurs in a well-formed derivation.

A complete description for how one may construct an automaton that verifies the well-formedness of derivation trees is given in Sec. B.3.

2.1.2 Weak Generative Capacity

The expressive power of a grammar formalism can be evaluated with respect to the generated string languages (weak generative capacity) and the generated tree languages (strong generative capacity). The latter is obviously of greater interest to linguists, but solid results have emerged only recently. Weak generative capacity results, on the other hand, abound in the mathematical linguistics literature, and while their linguistic relevance has been contested (a point I pick up again in Sec. 2.3.6), they still are useful in establishing a lower bound — if a formalism cannot even generate the right set of strings, it cannot generate the right set of phrase structure trees either. Fortunately, both the weak and the strong generative capacity of MGs is relatively well understood.

The weak generative capacity of MGs was determined early on by Harkema (2001a,b) and Michaelis (1998, 2001), who proved their equivalence to multiple context-free grammars (MCFGs; Seki et al. 1991). This puts them in the class of mildly context-sensitive formalisms, which are more expressive than context-free grammars but weaker than context-sensitive grammars.

Theorem 2.1 (MG ≡ MCFG). For every MG there exists a weakly equivalent MCFG,

and the other way round. □

In contrast to context-free grammars, mildly context-sensitive formalisms can handle a limited amount of crossing dependencies but none of the highly complicated patterns a context-sensitive grammar can produce. Take the language $a^m b^n c^m d^n$ for instance. It consists of all strings such that no as are preceded by occurrences of b, c, or d, no occurrences of a, b, or c follow any d, all bs precede all cs, and the number of as and bs matches the number of cs and ds, respectively. Hence this language contains $abbcdd$, $aaabcccd$, and $aabbccdd$, among others, but not $bbacdd$, $aaabcccddb$, or $abcdd$. This language is mildly context-sensitive but not context-free, even though $a^m b^n c^m d^k$ is a context-free language. This is so because $a^m b^n c^m d^n$ contains two crossing dependencies — one between a and c, the other between b and d — whereas $a^m b^n c^m d^k$ only has one dependency, so crossing of dependencies is trivially precluded. In a sense, $a^m b^n c^m d^n$ is the result of interleaving the strings of two distinct context-free languages, namely $a^m c^m$ and $b^n d^n$. The language of strings of as whose length is prime (aa, aaa, $aaaaa$, ...), on the other hand, is context-sensitive but not mildly context-sensitive. The consensus among mathematical linguists is that natural languages are mildly context-sensitive, so MGs' membership in this class is a welcome state of affairs.

The term mild context-sensitivity actually subsumes two classes of languages: Tree Adjoining languages (TALs) and multiple context-free languages (MCFLs), with the latter properly subsuming the former. Both are named after the prototypical formalism generating them, *viz.* Tree Adjoining Grammar (TAG) and MCFGs. The separating cases between the two classes look rather arbitrary from a naive perspective. For example, both $a^n b^n c^n d^n$ and $a^n b^n c^n d^n e^n$ are MCFLs, but only the first one is also a TAL. It is unclear which class is a better approximation of natural language. Both contain languages that differ significantly from all attested natural languages, but there is no consensus whether there are natural languages that do not belong to anyone of the two. So while both classes overgenerate, they might subsume natural

language and thus provide a reasonable first approximation.

Several arguments have been put forward in the literature that some natural language phenomena necessitate an extension of MCFLs, called parallel MCFLs (PMCFLs; Culy 1985; Radzinski 1991; Michaelis and Kracht 1997; Kobele 2006). PMCFLs allow for a limited amount of copying, as for example in the language a^{2^n}, which is the smallest set containing the string a and every string whose length is double that of some other string (a, aa, $aaaa$, $aaaaaaaa$, ...). As the jump from MCFLS to PMCFLs involves copying, it is hardly surprising that PMCFLs constitute exactly the class of languages generated by MGs where movement may leave behind pronounced copies. So with respect to string language, standard MGs are strictly more powerful than TAG (because TAL $\subset$ MCFL), but they are also strictly weaker than MGs with overt copying (because MCFL $\subset$ PMCFL).

2.1.3 The Importance of Remnant Movement 

It is worth noting that the equivalence between MGs and MCFGs falters if movement must obey the Proper Binding Condition (PBC). The PBC requires that a mover c-commands all its traces, which rules out remnant movement. Kobele (2010b) shows that MGs without remnant movement are equivalent to context-free grammars (CFGs). So rather than MCFLs, these MGs generate only context-free string languages. Since not all natural languages are context-free on the level of strings, these PBC-obeying MGs are empirically inadequate. Moreover, MGs that shun Move and only use Merge are already capable of generating all context-free string languages. Consequently, PBC-obeying movement adds nothing to the formalism as far as weak generative capacity is concerned.

Lemma 2.2. For every CFG there is a weakly equivalent MG whose LIs only carry Merge features. □

Proof. It is a well-known fact that every CFG can be brought into Chomsky normal

form such that the right-hand side of a rule consists of exactly two non-terminals or exactly one terminal. Assume w.l.o.g. that the start category of a given CFG is C rather than the standard S. Each rule R of a CFG in Chomsky normal form can be translated into a Minimalist LI as follows. If R is of the form $A \rightarrow BC$, then its corresponding LI is ε :: =C =B A. Otherwise R is of the form $A \rightarrow a$ and corresponds to LI a :: A. Proving the correctness of this translation is left as an exercise to the reader. ■

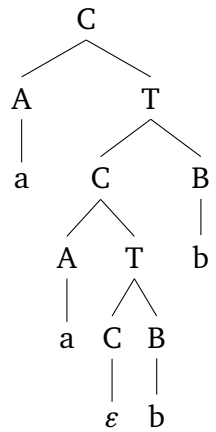
Example 2.6 Converting CFGs into MGs

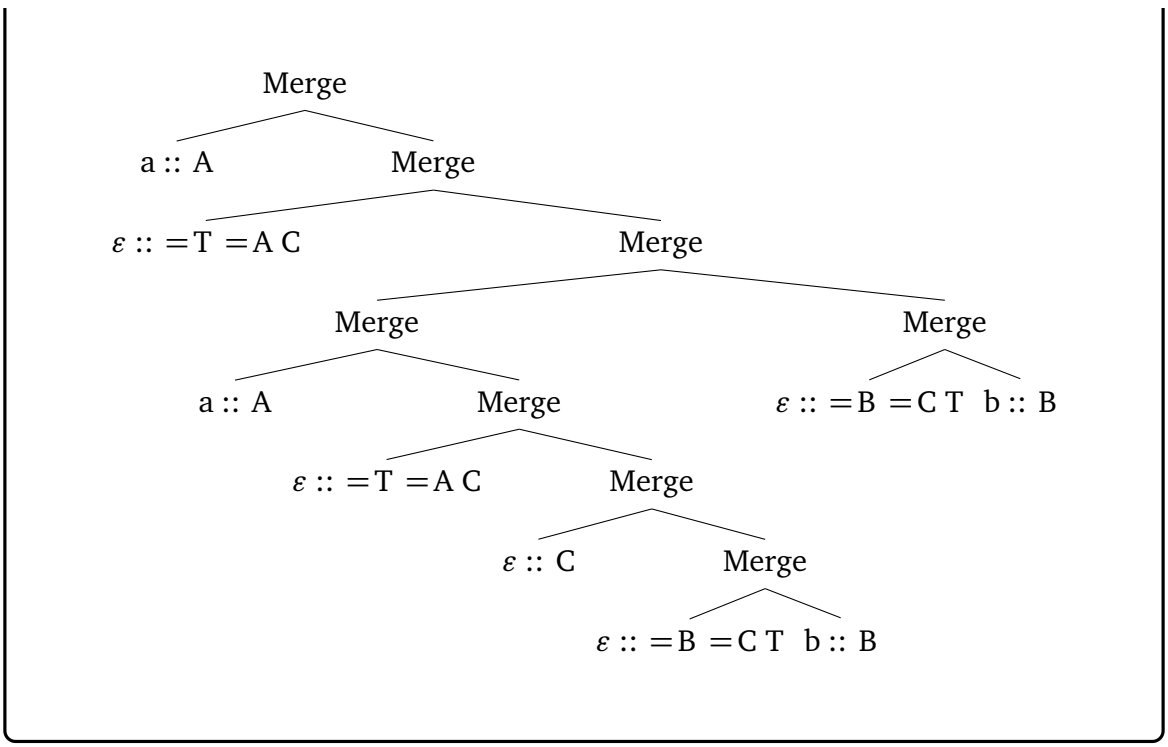
Consider the following CFG for the string language $a^n b^n$, $n \geq 0$, and the lexicon of its corresponding MG.

$$\begin{array}{ll} \text{C} \rightarrow \text{A T} & \text{C} \rightarrow \varepsilon \\ \text{T} \rightarrow \text{C B} & \text{A} \rightarrow \text{a} \\ & \text{B} \rightarrow \text{b} \end{array}$$

$$\begin{array}{ll} \varepsilon :: =\text{T} =\text{A C} & \varepsilon :: \text{C} \\ \varepsilon :: =\text{B} =\text{C T} & \text{a} :: \text{A} \\ & \text{b} :: \text{B} \end{array}$$

The CFG derivation for $aabb$ is shown below together with its MG pendant.





Lemma 2.3. For every MG whose LIs only carry Merge features there is a weakly equivalent CFG. □

Proof. This already follows from Kobele's (2010b) result for PBC-obeying MGs, but can also be shown independently. Given an MG M, a weakly equivalent CFG G is constructed as follows. Let L be the MDTL of M, and L_{loc} the result of relabeling each node in every derivation of L with the state it is assigned by the automaton described in example 2.4.

Suppose that slices are right branching as defined in Sec. 1.2. For every node m with daughters d and d', add the rule $m \to d'\ d$ to G if d' is a leaf and the first component of m's label is a proper suffix of the label of d'. Otherwise add the rule $m \to d\ d'$. For every LI $\sigma :: \mathrm{f_1\ f_2 \cdots f_n}$ of M add the rule $\langle f_1\ f_2 \cdots f_n \rangle \to \sigma$. Since both the number of states and the size of M's lexicon are finite, G has a finite number of rules, wherefore it is a CFG.

The weak equivalence of M and G is a corollary of Thatcher's theorem (Thatcher

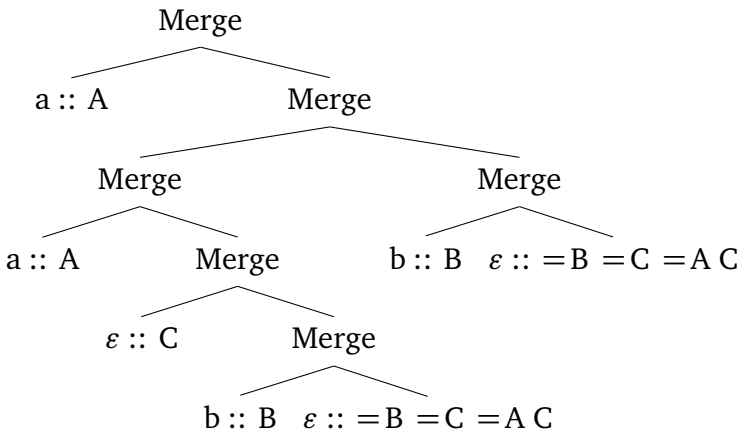
1967; see also Sec. 3.2.2), which, loosely paraphrased, establishes that if one regards the states an automaton assigns to the trees of a regular language as part of the node labels, then one obtains the derivation tree language of some CFG. ■

Example 2.7 Converting MGs without Move into CFGs

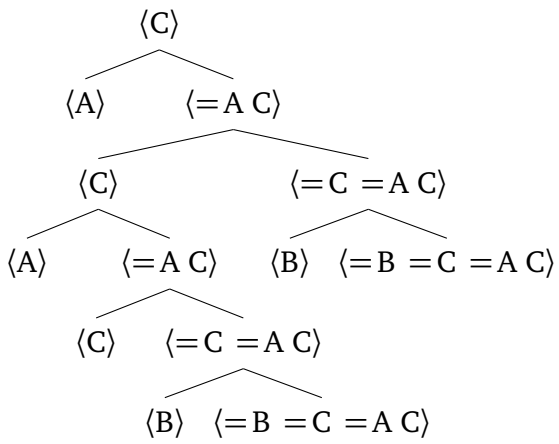
We already saw an MG for $a^n b^n$, but this grammar can be made more succinct.

$$\varepsilon :: \text{C} \qquad \text{a} :: \text{A}$$
$$\varepsilon :: {=}\text{B}\ {=}\text{C}\ {=}\text{A}\ \text{C} \qquad \text{b} :: \text{B}$$

The derivation tree for $aabb$ is also less verbose (depicted here with right-branching slices).



The automaton then assigns states in the usual fashion.



Proceeding through the tree from top to bottom, we add the following three rules:

$$\langle \mathrm{C} \rangle \to \langle \mathrm{A} \rangle \, \langle {=}\mathrm{A}\ \mathrm{C} \rangle$$

$$\langle {=}\mathrm{A}\ \mathrm{C} \rangle \to \langle \mathrm{C} \rangle \, \langle {=}\mathrm{C}\ {=}\mathrm{A}\ \mathrm{C} \rangle$$

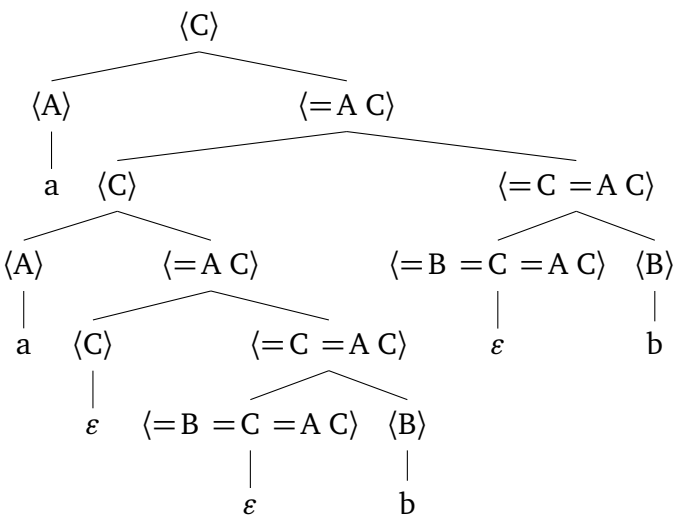
$$\langle {=}\mathrm{C}\ {=}\mathrm{A}\ \mathrm{C} \rangle \to \langle {=}\mathrm{B}\ {=}\mathrm{C}\ {=}\mathrm{A}\ \mathrm{C} \rangle \, \langle \mathrm{B} \rangle$$

Each LI also needs to be converted into a rule.

$$\langle \mathrm{C} \rangle \to \varepsilon \qquad \langle \mathrm{A} \rangle \to a$$

$$\langle {=}\mathrm{B}\ {=}\mathrm{C}\ {=}\mathrm{A}\ \mathrm{C} \rangle \to \varepsilon \qquad \langle \mathrm{B} \rangle \to b$$

It is easy to see that processing additional derivation trees will not add further rules to G, so we can terminate the procedure here. The derivation tree of G for $aabb$ is almost indistinguishable from the one above. The major difference is that leafs have no siblings, and the order of heads and their first argument is switched to reflect their order in the string.



The two lemmata above and the findings of Kobele (2010b) jointly imply the weak equivalence of CFGs, MGs without movement, and MGs without remnant movement.

Theorem 2.4. Given a string language L, the following three claims are equivalent:

- L is context-free,
- L is generated by an MG whose LIs carry no Move features,
- L is generated by an MG where every instance of Move obeys the PBC. □

It is tempting to interpret this theorem as a slight against the PBC and in favor of remnant movement. But things are not that simple. While it is true that context-free formalisms are empirically inadequate because of the mild context-sensitivity of natural language, the inadequacy of a grammar formalism does not necessarily entail the inadequacy of all its subparts. In the case at hand, it is certainly true that MGs are too weak if they only have PBC-obeying movement; yet adding other movement types such as head movement renders them once again weakly equivalent to MCFGs

and MGs with remnant movement (Stabler 1999, 2003). Regarding generative capacity, then, Thm. 2.4 only states that PBC-obeying phrasal movement by itself is insufficient and must be complemented by another movement type, be it remnant movement, head movement or something completely different.

The reader might also be wondering at this point why the conversion from MGs to CFGs does not work with Move. In other words, why are MGs weakly equivalent to MCFGs rather than CFGs? For instance, the derivation in example 2.4 contains a Move node that is assigned the state $\langle T \rangle$ by the automaton, while its daughter has the state $\langle +\text{nom } T, -\text{nom} \rangle$. Why, then, does it not suffice to simply add the rule $\langle T \rangle \rightarrow \langle +\text{nom } T, -\text{nom} \rangle$ to the CFG? The answer is that even though this rule correctly represents the inference of the feature calculus, it is string-vacuous — the rule does not alter the derived string in any way. But changing the derived string is the main task of Move, it is designed to account for the fact that constituents do not always appear in their original position. In order to handle Move, the rule above must be combined with an instruction that selects a specific substring of the string generated so far and puts it in front of it. Adding such displacement instructions to CFGs in order to handle Move yields MCFGs, which we already saw are weakly equivalent to standard MGs.

The last two sections have touched on several points that open up a very different perspective on MGs. The importance of derivation trees to MGs was already established in the previous chapter, but now we also know that derivation trees can be computed in an efficient manner using only a finitely bounded amount of working memory. This fact also makes it possible to describe MDTLs in terms of CFGs. In other words, MGs are underlyingly context-free, but their context-freeness is obscured by the fact that the linear order in the string is altered by Move. This is reminiscent of the Aspects model, where context-free D-structures were manipulated by transformational rules to yield the intended surface structure. The crucial difference, though, is that the mapping from Minimalist derivations to phrase structure

trees is very limited. If only Merge and ECP-obeying Move are present, the mapping is so weak it even fails to increase the class of string languages that can be generated. More powerful movement types such as remnant movement, head movement, or sideward movement allow MGs to take the step beyond context-freeness. The details of this transformational step are discussed next.

2.1.4 Strong Generative Capacity

The modular perspective of MGs in terms of I) a context-free feature calculus represented by regular derivation tree languages, and II) a mapping from derivations to phrase structure trees has proved invaluable in establishing their strong generative capacity. While some results go as far back as Kolb (1999), arguably the first intuitive application of the idea is given in Kobele et al. (2007). Rather than discuss each finding and the technical machinery underlying it, I restrict myself to a few intuitive observations here that serve to situate MG tree languages with respect to other formalisms.

In Sec. 1.2.3, I showed that a very simple mapping suffices to turn Minimalist derivation trees into their respective multi-dominance trees. This mapping consisted of three steps. First the linear order of siblings had to be modified such that heads appear to the left of their complements rather than to the right. Then nodes had to be relabeled such that LIs lost their feature component and interior nodes were replaced by $<$ and $>$ to indicate projection. Finally, new branches were introduced between an LI l's slice root and all the occurrences of l, i.e. the Move nodes that check one l's licensee features. This mapping was specified in terms of an MSO transduction. That is, the predicates encoding the relations in the multi-dominance tree (labels, dominance, precedence) were explicitly defined in terms of the predicates for the derivation tree. Dominance in the derived tree, for instance, is defined by a very

simple MSO formula.

$$x \blacktriangleleft y \leftrightarrow x \triangleleft y \vee \exists l \left[occ(x,l) \wedge sliceroot(y,l) \right]$$

This formula can be translated as "node x is a mother of node y in the derived tree iff x is a mother of y in the derivation tree or there is some LI l such that x is an occurrence of l and y the slice root of l." The other formulas are of comparable complexity, showing that the mapping from derivation trees to multi-dominance trees is a rather simple MSO transduction.

Multi-dominance trees are a peculiarity of recent Minimalism and seldom considered in other frameworks such as TAG. So in order to simplify comparisons between MGs and these formalisms, one should modify the transduction such that it yields standard trees instead. This is a simple task if one isn't concerned about the insertion of traces. Intuitively, if a node has multiple mothers in the multi-dominance tree, it suffices to remove all branches except the one attached to the highest mother.

$$x \blacktriangleleft_{standard} y \leftrightarrow x \blacktriangleleft y \wedge \neg \exists z [z \blacktriangleleft y \wedge z \blacktriangleleft^{+} x]$$

The formula reads out as "node x is a mother of node y in the phrase structure tree iff x is a mother y in the multi-dominance tree and there is node z in the multi-dominance tree that is a mother of y and (properly) dominates x."

Note that this transduction yields phrase structure trees without traces. Adding traces is rather complicated since a derived tree with traces contains more nodes than its corresponding derivation tree. Increasing the size of a tree, albeit possible, is slightly cumbersome to achieve with an MSO transduction. Also note that the formula above only yields the correct result for phrasal movement and needs to be expressed more generally for head movement, sideward movement, or other movement types. These caveats aside, it is clear that applying the transduction encoded by this MSO formula after the one mapping derivations to multi-dominance

trees yields standard phrase structure trees. Since MSO transductions are closed under composition, the result of applying these two transductions in sequence is itself an MSO transduction. Hence mapping derivation trees to standard phrase structure trees can be achieved via an MSO transduction, too.

The use of MSO transductions establishes an upper bound on the strong generative capacity of MGs. This is due to the following characterization: a tree language L is the result of applying an MSO transduction to a regular tree language iff L is the tree yield of a hyperedge replacement grammar (cf Engelfriet 1997). Hyperedge replacement grammars are a formalism for generating graphs rather than trees, but just like every tree has a string as its yield, graphs have trees as their yield. Interestingly, hyperedge replacement grammars are weakly equivalent to MCFGs, just like MGs (a corollary of Engelfriet and Heyker 1991 and Weir 1992). That is to say, the string yield of the tree yield of a hyperedge replacement grammar is an MCFL, and for every MCFL there is a hyperedge replacement grammar that weakly generates it. Since MGs and hyperedge replacement grammars generate the same class of string languages, do the two also generate the same class of tree languages?

The answer is no, hyperedge replacement grammars are strictly more powerful than MGs regarding tree languages. This is so because the tree languages of MGs are the image of regular tree languages under *direction-preserving* MSO transductions, which form a proper subclass of the class of all MSO transductions (Mönnich 2006, 2007). An MSO transduction is direction preserving iff it holds for all nodes x and y that if x is the mother of y in the output tree, then x properly dominates y in the input tree. This class, and thus the class of phrase structure tree languages generated by MGs, coincides with the class of multiple regular tree languages (MRTL; see Raoult 1997 and Engelfriet 1997). These are the tree analogue of MCFGs, or equivalently, regular tree grammars over tuples of trees rather than single trees.

Obviously one would like to know what kind of tree structures can be generated by hyperedge replacement grammars but not by MGs. To my knowledge, no good

characterization of this class is known at this point. But comparisons with other frameworks furnish some preliminary answers. In particular, some TAGs generate tree languages that are beyond the reach of MGs. Note that his holds even though TAGs are strictly weaker than MGs on a string level (so the relation between strong and weak generative capacity is indeed rather indirect). TAG tree language coincides with two classes. The first one is the class of tree languages that are the image of regular tree languages under *inverse direction preserving* MSO transductions, where an MSO transduction is inverse direction preserving iff it holds for all nodes x and y that if x is the mother of y in the output tree, then y properly dominates x in the input tree. The second class is given by the tree languages generated by monadic, simple context-free tree grammars (Mönnich 1997; Fujiyoshi and Kasai 2000; Kasprzik 2007; Kepser and Rogers 2011). A context-free tree grammar operates similar to a context-free string grammar. Where the latter "slices" a generated string in half to insert a new string between the two, a context-free tree grammar slices a tree in half and inserts a new tree in between. Notice the difference to regular rewriting mechanisms, which only insert new material at the edge of the already assembled structure. So while MGs rely on a generalized notion of regularity (the connection to MRTGs), TAGs depend on a limited version of context-free tree manipulation.

Graf (2012e) shows that TAG tree languages are generated by MGs with a particular kind of lowering movement. So subtrees move to a position they c-command, not one they are c-commanded by. This result translates the rather abstract issues of strong generative capacity into a concrete linguistic one: are there any natural language phenomena that can only be described by lowering? If not, then no natural language involves tree structures that are generated by TAGs but not by MGs.

While TAGs and MGs disagree on the kind of mapping from derivation trees to derived trees, they both have regular derivation tree languages and rely on MSO-

definable mappings. The same kind of split can be found with other formalisms. MGs with overt copying, for example, have the same derivation trees as MGs without copying; they only differ from the latter in that they use an MSO-mapping to multi-dominance trees which are then converted into standard trees with over copies instead of traces. There seems to be a general consensus, then, that the complexity of natural language stems from the interaction of two simple components. The only disagreement is about the specifics of the mapping from derivations to derived trees. Narrowing down these parameters will prove a lot more difficult than weak generative capacity results, and it will necessarily require a lot more linguistic expertise.

2.2 Adding Head Movement and Affix Hopping 

The derivational perspective makes it very easy to extend and modify MGs while maintaining their defining properties: the regularity of MDTLs and a mapping from derivation trees to derived trees that can be expressed in terms of MSO. Graf (2012c), for instance, uses MSO to define a parameterized system that allows for a wide range of new movement types to be added to MGs, including lowering, sideward movement, head movement, and affix hopping. The generality of the system comes at the cost of increased abstractness, but the underlying ideas are relatively easy to glimpse from a few examples. As a quick demonstration I show how the MSO-based definition of MGs in Sec. 1.2 can be altered to accommodate head movement and affix hopping. Note that this is merely meant as a technical demonstration, so the linguistic faithfulness of the implementation is of little concern here.

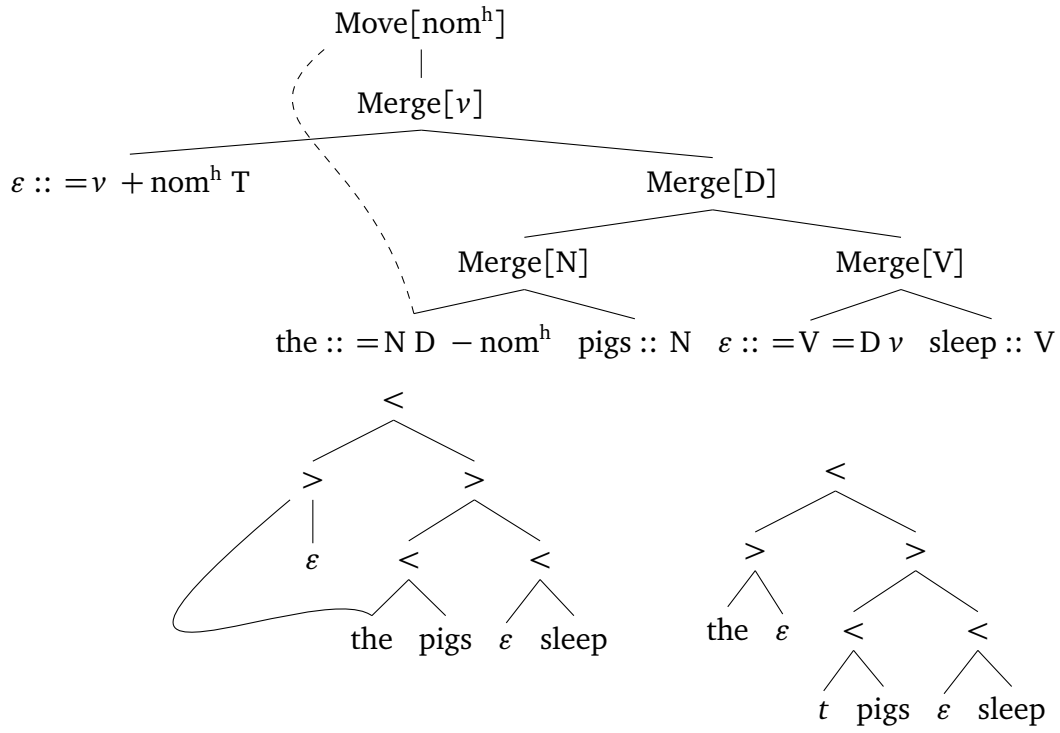
2.2.1 Head Movement

There are many different incarnations of head movement in the literature (see Travis 1984, Matushansky 2006, and Roberts 2010, among others). I will assume here that

head movement differs from phrasal movement in that I) the LI carrying a licensee feature moves alone without carrying along the remainder of its projected phrase, and II) the LI does not move to a specifier but attaches to the head carrying the matching licensor feature.

Example 2.8 Head movement

In example 1.5 on page 17 we saw the subject DP *the pigs* undergo phrasal movement to Spec,TP to check its −nom feature. Suppose now that head movement takes place instead of phrasal movement, so *the* moves by itself and attaches to the empty T-head, as depicted below.



Comparing the trees above to those in example 1.5, one notices immediately that the derivations are identical *modulo* the name of the feature triggering movement. This means that head movement behaves exactly like phrasal movement with respect to the feature calculus, it is only in the mapping from derivations to derived trees that

the two types of movement diverge. Phrasal movement is the result of shifting the tail of the movement branch in the augmented derivation tree from the moving LI upwards to the slice root of said LI. Head movement takes place when the targeted Move node is lowered from its position in the derivation tree to just above the LI hosting the relevant licensor feature.

Since the differences between head movement and phrasal movement affect only the mapping from derivations to phrase structure trees, the feature calculus can remain mostly unchanged. However, we need to extend our tuple-definition of features with another component that serves to distinguish phrasal movement features from head movement features.

Definition 2.5. Let BASE be a non-empty, finite set of *feature names*. Furthermore, OP $:= \{merge, move\}$, POLARITY $:= \{+, -\}$, SIZE $:= \{head, phrase\}$ are the sets of *operations*, *polarities*, and *movement sizes*, respectively. A *feature system* is a non-empty set *Feat* $\subseteq$ BASE $\times$ OP $\times$ POLARITY $\times$ SIZE.

The notion of match also needs to be amended such that two features must also agree on their movement size value. Otherwise a phrasal licensee feature could be matched by a head movement licensor and the other way round. So far, then, not much has been accomplished beyond bifurcating movement features into two subtypes that behave exactly as before with respect to feature checking.

The actual locus of variation is in the MSO mapping. Recall that when formalizing phrasal movement in MSO terms, we had to express immediate dominance in the derived multi-dominance tree ($\blacktriangleleft$) in terms of dominance in the derivation tree ($\lhd$). The crucial insight was that we only had to keep all existing branches and add new

ones for movement, which is accomplished via a disjunction in the MSO formula.

$$x \blacktriangleleft y \leftrightarrow x \triangleleft y \vee \exists l \left[occ(x,l) \wedge sliceroot(y,l) \right]$$

So we keep all dominance branches from the derivation ($x \triangleleft y$) and add new ones between Move nodes and the slice root of the LI they check a feature of ($\exists l[occ(x,l) \wedge sliceroot(y,l)]$).

Implementing head movement, on the other hand, requires more than just adding branches because the geometry of the derived tree does not completely match that of the derivation tree. Consider the previous example. The Move node in the derivation is no longer reflected in the output tree, whereas a new node has been inserted immediately above the head that acts as landing site for head movement. The label of this node is $>$, since the moving LI attaches to the left of the head. Notice that in the case of phrasal movement, the Move node would have been realized as $>$, too, but in a higher position. In a sense, then, one may view the new node in the derived tree as the result of lowering the Move node in the derivation tree.

In order to lower a node, we have to first remove the dominance branches connecting it to the rest of the tree, and then add new branches to its new mother m and daughters d_1 and d_2. We also have to ensure that m does not immediately dominate d_1 and d_2. In the case of head movement, only one daughter has to be disconnected, namely the LI targeted by head movement. Our next step, then, is to replace $x \triangleleft y$ in our definition of the output dominance relation $\blacktriangleleft$ by a more sophisticated MSO formula that achieves the desired behavior.

As many times before, we first define some auxiliary predicates that allow us to write our formulas more succinctly. A Move node m is a *head occurrence* of LI l, $h\text{-}occ(m,l)$, iff m is an occurrence of l and associated to a head movement licensor feature. Whenever l is irrelevant, we only write $h\text{-}occ(m)$. That is to say, $h\text{-}occ(m) \leftrightarrow \exists l[h\text{-}occ(m,l)]$. Furthermore, m and n are *head mates* iff both belong

to the same slice and are either head occurrences or LIs:

$$x \sim_h y \leftrightarrow x \sim y \wedge (h\text{-}occ(x) \vee lex(x)) \wedge (h\text{-}occ(y) \vee lex(y))$$

Next we define a peculiar variant of dominance, *immediate dominance with respect to formula* ξ, abbreviated $\lhd_\xi$. Node x immediately dominates y with respect to ξ iff x properly dominates y, $\xi(x,y)$ holds, and no node between x and y satisfies ξ. In MSO terms:

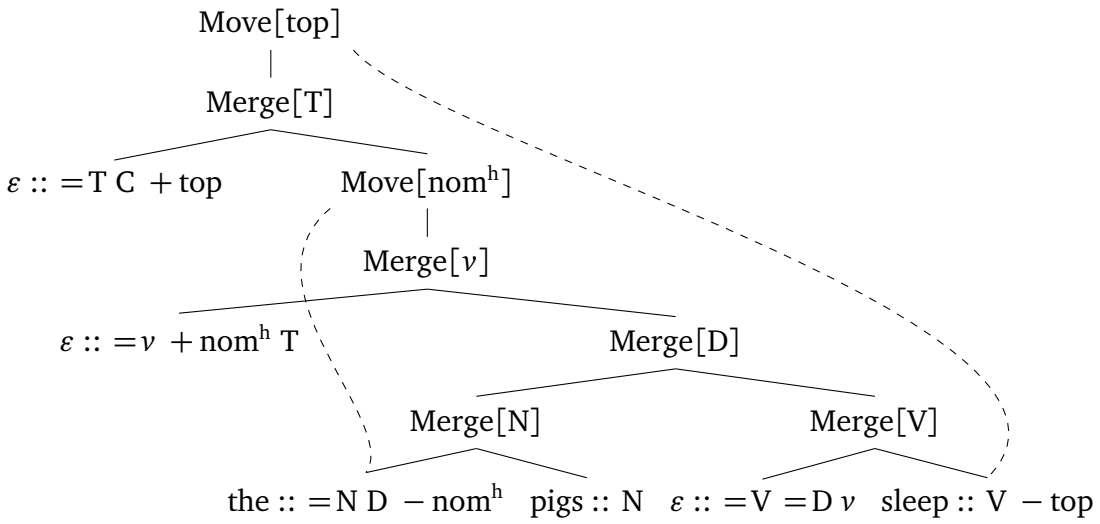
$$x \lhd_\xi y \leftrightarrow x \lhd^+ y \wedge \xi(x,y) \wedge \forall z \left[x \lhd^+ z \wedge z \lhd^+ y \rightarrow \neg\xi(x,z) \vee \neg\xi(z,y) \right]$$

Replacing ξ by $\sim_h$ allow us to pick out the dominance relations between head occurrences (and LIs). To avoid clutter I shorten $\lhd_{\sim_h}$ to $\lhd_h$. In order to obtain the dominance relations nodes that are neither LIs nor head occurrences, we interpret $\xi(x,y)$ as $\neg h\text{-}occ(x) \wedge \neg h\text{-}occ(y) \wedge \neg lex(x) \wedge \neg lex(y)$. The corresponding relation is denoted $\lhd_{\overline{h}}$.

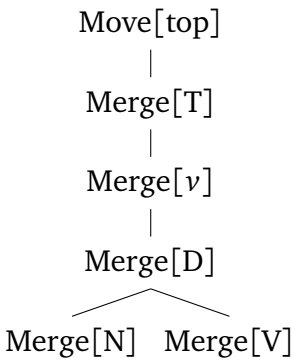
Notice that every LI is immediately dominated by a Merge node in the derivation, so every slice with at least two nodes contains at least one node that is neither an LI nor a head occurrence. Hence equating output dominance with $\lhd_{\overline{h}}$ yields a single tree (rather than a number of disconnected trees) that contains all nodes of the derivation that are phrasal, in a loose sense. On the other hand, $\lhd_h$ yields, for each slice s, a tree that consists of the LI l of s and any head occurrences in s that dominate l.

Example 2.9 Factoring head movement out of the derivation

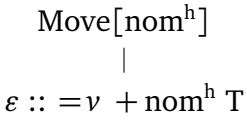
Consider a minimally extended variant of the augmented derivation from example 2.8 that also contains a C-head which selects the TP and then licenses topicalization of the VP.



The tree induced by $\lhd_{\overline{h}}$ is given below. Notice that even though the VP consists of a single LI the topicalization step still constitutes phrasal movement, wherefore Move[top] is included in the $\lhd_{\overline{h}}$-tree.



The relation $\lhd_h$, on the other hand, yields six distinct trees. Five of them are simply the LIs ε :: =T C, the :: =N D −nom$^{\text{h}}$, pigs :: N, ε :: =V =D v, and sleep :: V −top. The last one contains the T-head and the head movement node in the same slice.



Now it only remains for us to attach each tree created by $\lhd_h$ to the appropriate Merge node in the tree created by $\lhd_{\overline{h}}$. This will always be the Merge node m such that the LI of the relevant slice is immediately dominated by m in the derivation tree. The predicate $joint(m, r)$ holds between Merge node m and the root r of the tree induced by $\lhd_h$ that attaches to m. The root r is also called a *head root*.

$$headroot(x, y) \longleftrightarrow lex(y) \wedge x \sim_h y \wedge \forall z[z \sim_h y \rightarrow x \approx y \vee x \lhd^+ z]$$

$$joint(x, y) \longleftrightarrow \exists l \left[x \lhd l \wedge headroot(y, l)\right]$$

Modulo new branches induced by movement, the derived tree with correctly lowered head occurrences is given by the formula below.

$$x \blacktriangleleft y \iff x \lhd_{\overline{h}} y \vee x \lhd_h y \vee joint(x, y)$$

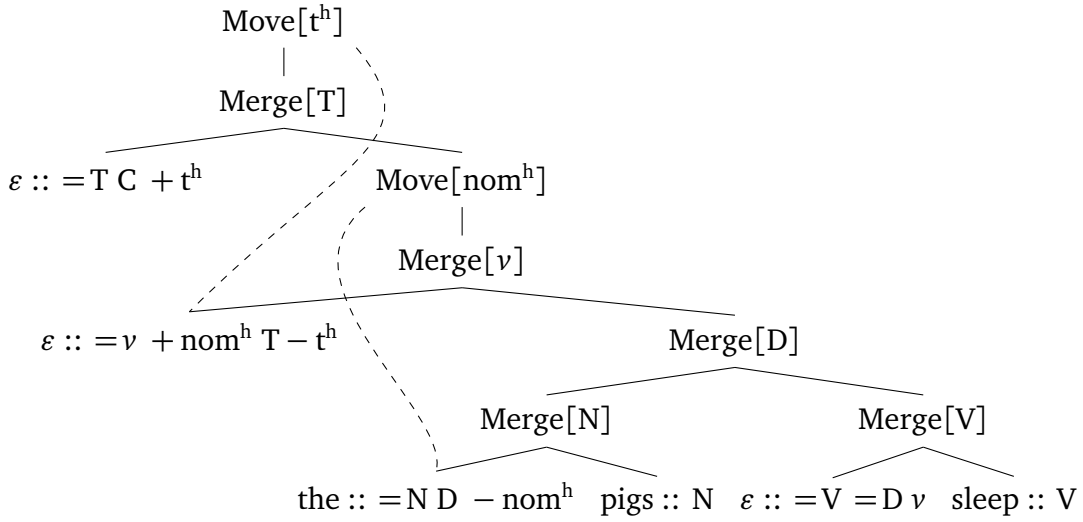
In order to bring in movement, we have to add another disjunct. For MGs without head movement, it was enough to add $\exists l[occ(x, l) \wedge sliceroot(y, l)]$. This formula must be amended to capture the split between head movement and phrasal movement. Thankfully it suffices to check if x is a head occurrence. If it is not, the tail of the movement branch starts at the highest node of the slice, yielding standard phrasal movement. If x is a head occurrence, the head root is used instead. This ensures that whenever LI l moves to LI l', l is pied-piped along should l' undergo head movement, too. The dominance relation in the derived tree thus is given by

$$\begin{aligned} x \blacktriangleleft y \iff\ & x \lhd_{\overline{h}} y \vee x \lhd_h y \vee joint(x, y) \vee \\ & \exists l \left[occ(x, l) \wedge \big(\neg h\text{-}occ(x) \rightarrow sliceroot(y, l)\big) \wedge \big(h\text{-}occ(x) \rightarrow y \approx l\big)\right] \end{aligned}$$

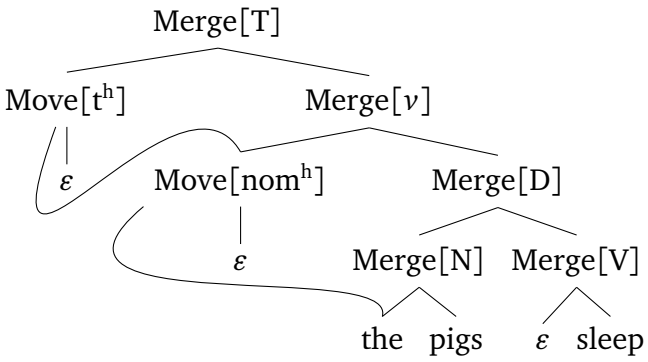
The transduction formulas for computing linear precedence and relabeling nodes carry over unchanged from standard MGs.

Example 2.10 Transducing head movement

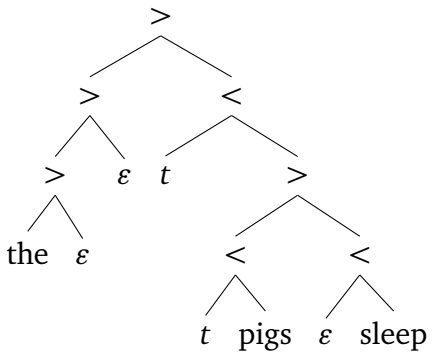
Suppose for the sake of argument that rather than topicalization, the derivation in example 2.9 involves T-to-C movement.



The derivation is mapped to the following multi-dominance tree by our MSO transduction (interior node labels are unaltered for the sake of clarity).



Crucially, the second movement branch originates in Move[nom^h] because this node is the root of the tree induced by $\triangleleft_h$ for the slice corresponding to the TP. Unfolding the multi-dominance branches in the standard way produces the intended phrase structure tree.



There are of course further aspects of head movement that could be incorporated. For example, MG implementations of head movement often add a parameter to specify whether a head-moved LI should be linearized to the left or to the right of the targeted head. While this is a trivial modification from an MSO perspective, it is also unnecessary — since the system developed here treats head movement and phrasal movement exactly the same with respect to the feature calculus, it is possible for LIs to undergo remnant head movement. That is to say, a head can move out of a head it has previously moved into. So if y should appear to the right of x and standard movement would put it to the left, one can introduce an empty head z such that y head moves to z, followed by head movement of x to z. All phrases $p_1, \ldots, p_n$ residing in a specifier of x must also move into a specifier of z. The resulting structure linearizes as $p_1 \cdots p_n xyz = p_1 \cdots p_n xy$, as desired. So just as with phrasal movement, linearization parameters for head movement can be emulated by remnant head movement if one is only concerned with linear order rather than constituency.

From a linguistic perspective, the MSO implementation is too permissive as it neither enforces cyclicity nor puts strong limitations on how often a head may move. This is at odds with the commonly posited Head Movement Constraint (Travis 1984),

which blocks every head from moving more than once or skipping any heads when moving. Both conditions are easily expressed in MSO. The former simply prevents all LIs from carrying more than one head licensee feature. The latter ensures that head movement does not cross any slices that contain neither the source nor the target of movement. In other words, the path from an LI to its head occurrence contains exactly one slice root (not counting the head occurrence itself in those cases where it also happens to be a slice root). Thus the Head Movement Constraint can also be incorporated into our system without leaving the realm of MSO-definability.

Right now my implementation also allows for a phrase to move independently after its head has moved to a higher head, which is highly unnatural from a linguistic perspective. This kind of movement happens whenever an LI has a head licensee feature followed by a licensee feature for phrasal movement. So in order to block such scenarios one only has to enforce that every phrasal licensee feature precedes all head licensee features.

2.2.2 Affix Hopping

Head movement mirrors phrasal movement in that it always targets a higher position in the tree. However, there are instances where a head is apparently spelled out in a lower position. For example, it is commonly assumed that the parts of inflectional morphology related to subject-verb agreement are hosted by T. This creates a problem in English, where it seems that the verb remains in its low VP position yet carries the inflectional affixes that should be residing in T.

(3) a. John often chastises Mary.

b. * John s often chastise Mary.

One solution to this puzzle is head movement to a lower position, also known as affix hopping. The T head lowers to the V head, thereby crossing the VP-adjunct *often* to produce the surface string *chastises*.

Intuitively, affix hopping proceeds exactly like standard head movement: a head carries a feature, some Move node is a matching head occurrence, and a dominance branch is established between said Move node and the head root of the relevant $\triangleleft_h$-tree. What makes affix hopping interesting is that the Move node can no longer dominate the LI because Move nodes designate the target positions of movement, which in the case of affix hopping must not dominate the LI by definition. Consequently our standard procedure for determining the occurrences of an LI is no longer valid — affix hopping requires a new kind of derivational search algorithm. So while the transduction from derivations to derived trees is exactly the same as for head movement, the conditions on derivation trees are not. In other words, affix hopping is incorporated via a change in the MG feature calculus.

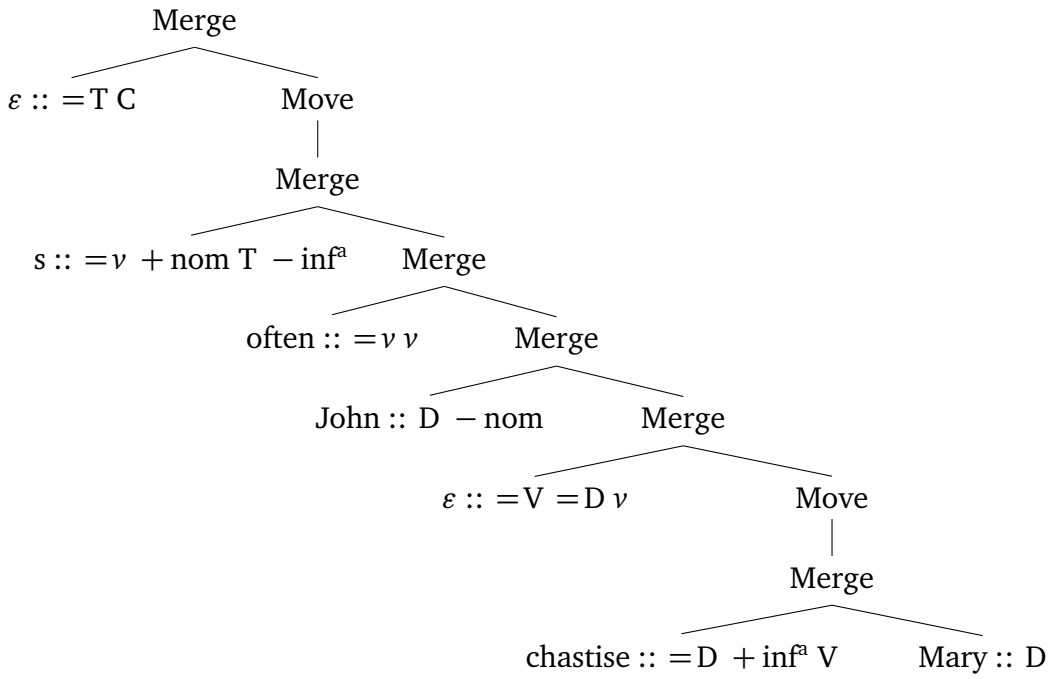
But what exactly should be changed, and where? In Sec. 1.2.2 MG derivations were limited by four constraints: **Final**, **Merge**, **Move**, and **SMC**. **Final** and **Merge** do not regulate the distribution of Move nodes, so they have no bearing on affix hopping. **Move** and **SMC**, on the other hand, jointly enforce the one-to-one correspondence between licensor and licensee features, and they also render Move deterministic. Clearly affix hopping does not differ from other movement types with respect to these properties, so these constraints should be tampered with either. Keep in mind, though, that **Move** and **SMC** do not talk about Move nodes as such but rather occurrences of LIs. In a sense, they do not care at all about what counts as an occurrence. Once a definition is supplied, they simply ensure that nodes are distributed in a particular way. One might say that **Move** and **SMC** are meta-constraints on movement. So the locus of variation for different movement types isn't **Move** or **SMC**, but the notion of occurrence that they build on.

Recall that the occurrences of an LI are defined in a recursive fashion. The zero occurrence of LI l, $occ_0(l)$, is the slice root of l. The i-th occurrence of l, $occ_i(l)$, is the unique Move node m such that I) m matches the i-th licensee feature of l, II) m dominates $occ_{i-1}(l)$, and III) no node dominated by m satisfies both I and II. That

movement always targets a higher position is captured by the use of dominance in II and III. But from an MSO perspective nothing hinges on the use of dominance, any MSO-definable relation could have been used in its place. In particular, we could have used the inverse of dominance, so that m must be dominated by $occ_{i-1}(l)$ and no node dominating m may satisfy the other conditions. This results in exactly the kind of downward movement needed for affix hopping.[1]

Example 2.11 Derivations for affix hopping

A reasonable (non-augmented) derivation for *John often chastises Mary* is given below, where adjunction of *often* is treated as category preserving selection for the sake of simplicity.



[1] Graf (2012c) employs a more complex definition that can also deal with certain situations where two Move nodes are not related by dominance, satisfy conditions I–III, and one of the two is an occurrence of another LI l. According to our definition both would be considered occurrences, thus inducing a violation of **Move**. In Graf (2012c), the Move node that is an occurrence of l would not be counted as an occurrence for any other LIs so that **Move** is satisfied after all.

The lower Move node is supposedly involved in the affix hopping of T to V, but let us see if our new definition of occurrence captures this fact correctly. The zero occurrence of T is the root of its slice, which is the higher one of the two Move nodes. This Move node is also an occurrence for *John*, but this is unproblematic because even though **SMC** does not allow for a Move node to be an occurrence for more than one LI, zero occurrences do no count for this purpose. Now that the zero occurrence has been identified, we have to look for the first occurrence. According to our definition, the first occurrence must be among the set of Move nodes that are dominated by the zero occurrence. In the case at hand there is only one such Move node, which belongs to the slice of *chastise*. From this we can easily infer that the Move node is associated to the feature $+\text{inf}^{a}$ and hence matches T's $-\text{inf}^{a}$. There clearly is no other Move node that satisfies these criteria and occurs along the path from this Move node to the zero occurrence, so the lower Move node is indeed an occurrence of the T head, as intended.

While this approach to affix hopping yields the desired structure, some may find it odd that a Move node may be present in the derivation before the element undergoing the movement step has even entered the workspace. Crucially, though, this is only puzzling if one interprets derivations as a temporal record of the sequence in which certain operations take place. The primary role of MG derivations, however, is that of a tree-geometric representation of the feature calculus. The difference between the two is mostly immaterial for standard upward movement. When an item with a licensee feature is introduced, a debt is incurred and the licensee feature must be checked by a matching licensor feature. The timing of checking happens to coincide with the location of the Move node. For downward movement such as affix hopping, it is the introduction of the licensor feature that incurs a debt. The corresponding Move node is, so to say, a leap of faith by the feature calculus,

indicating that this unmatched licensor feature is checked at this point on the assumption that a matching licensee feature will occur at some later point in the derivation. Under this interpretation, downward movement is closely related to the kind of hypothetical reasoning added to the feature calculus in Kobele (2010a, 2012). Viewing downward movement as hypothetical discharging is, to the best of my knowledge, the only view that can make sense of the fact that a Move node licensing downward movement can immediately dominate the LI of its slice. That is to say, the licensor feature is considered checked before the LI has even entered the workspace.

Example 2.12 Affix hopping and hypothetical feature checking

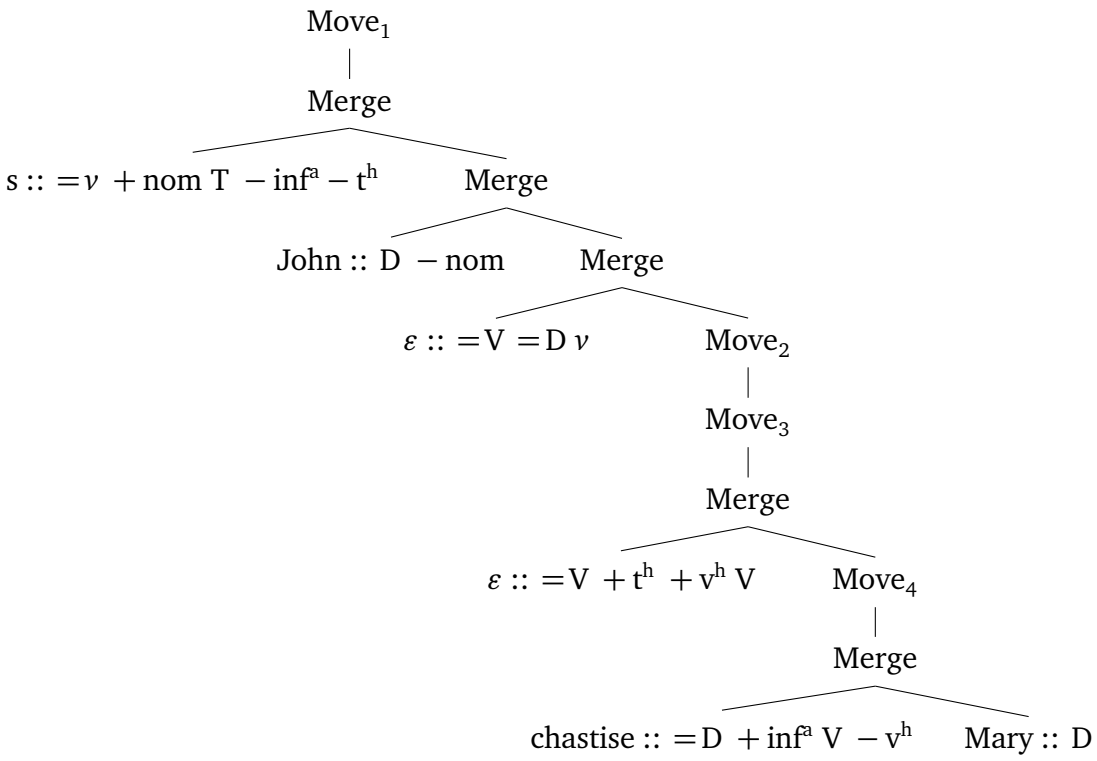
Suppose that the feature string of *chastise* in the previous example was $+\text{inf}^{a} = \text{DV}$ rather than $= \text{D} + \text{inf}^{a}\ \text{V}$. This only causes the Merge node that is the daughter of the Move in the derivation to now be its mother. Since the computation of occurrences ignores all Merge nodes, this minor structural change has no repercussions on well-formedness, even though it now seems as if movement takes place before *chastise* enters the derivation.

Parameterizing the notion of occurrence with respect to the type of movement has the appealing property that different movement types can easily be interleaved. In order to find the i-th occurrence of an LI, one only needs to know which Move node is the $(i-1)$-th occurrence. How the latter was found is irrelevant. Given the $(i-1)$-th occurrence, one merely checks whether the i-th licensee feature of the LI in question involves phrasal movement, head movement, or affix hopping, and then proceeds upwards or downwards through the tree depending on the type of movement.

Example 2.13 Interleaving head movement and affix hopping

The derivation in example 2.11 does not yield the correct string under the MSO transduction for head movement. Rather than to the right of *chastise*, *s* ends up to its left. We could of course modify the transduction such that the linear order of heads is switched in the case of affix hopping, but as an alternative let us explore the remnant head movement proposal discussed at the end of the last section. The idea is to switch the order of the heads by having them undergo remnant movement to an empty head.

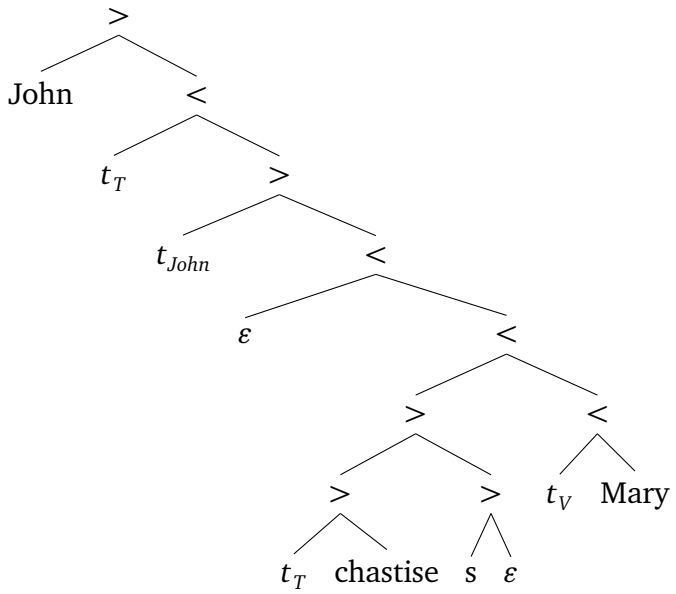
More precisely, the LI $\varepsilon :: {=}\mathrm{V}\ {+}\mathrm{t}^{\mathrm{h}}\ {+}\mathrm{v}^{\mathrm{h}}\ \mathrm{V}$ is added to the derivation just above the VP-level, and both *s* and *chastise* are given matching head movement licensee features $-\mathrm{t}^{\mathrm{h}}$ and $-\mathrm{v}^{\mathrm{h}}$, respectively. The new derivation is given below. Move nodes are subscripted to allow for easier referencing, while *often* and the C-head are omitted to reduce clutter.



The intended sequence of events is as follows: First *s* affix hops onto *chastise*. Then the former undergoes head movement to the empty head selecting *chastise*, followed by head movement of *chastise* to the same head. Since head movement is linearized to the left, the fact that *chastise* target *s* the same head but the latter moves before the former will put *chastise* to the left *s* in the generated string.

Let us verify that is indeed how things play out. We already know that Move_1 is an occurrence of *John* and thus of no interest here beyond also serving as the zero occurrence for *s*. Similarly, Move_4 is still the first occurrence of *s*, as in the previous example. It is also easy to see that Move_2 is the first occurrence of *chastise*. This leaves us with the question of whether Move_3 is the second occurrence of *s*. Since the second licensee feature of *s* is $-\text{t}^{\text{h}}$ and hence involves head movement, Move_3 must be the closest node that matches $-\text{t}^{\text{h}}$ and dominates the first occurrence of *s* (rather than being dominated by it). This is clearly the case.

A phrase structure tree with indexed traces is given below to further illustrate how this sequence of movement steps derives the correct word order.



The astute reader may point out that remnant movement is not needed in this case since *s* could affix hop directly onto the empty head, followed by head movement of *chastise*. This observation raises the interesting question whether interleaving of different movement types is ever needed.

The idea of interleaving upward and downward movement is rather puzzling from a linguistic perspective. After all, why would one lower X to Y and then raise it to Z rather than move it directly to Z? In the case of head movement and affix hopping, there really seems to be no good reason for having an LI undergo both. When it comes to phrasal movement, however, interleaving different movement types increases the strong generative capacity of MGs. As mentioned in Sec. 2.1.4, standard MGs and TAG are incomparable with respect to strong generative capacity (cf. Kobele et al. 2007), but MGs with a particular kind of lowering movement can generate all TAG tree languages (Graf 2012e). Said type of lowering movement, in turn, can be emulated by a sequence of interleaved upward and downward movement steps. This proves that not all instances of interleaved phrasal movement can be replaced by pure upward or pure downward movement.

The power of interleaving movement types has not been explored much so far. With respect to the movement generalized MGs developed in Graf (2012c), however, the following conjectures are very likely to hold:

- Every movement step can be replaced by an upward step (possibly) followed by a downward step (because every node in a tree can be reached by moving sufficiently far upwards and then downwards).
- Remnant upward movement cannot be replaced by downward movement (because that would incorrectly entail that the derived tree languages of MGs are regular).

- All instances of PBC-obeying upward movement (see Sec. 2.1.3) can be replaced by downward movement (because PBC-obeying upward movement preserves regularity).
- All instances of downward movement that are not preceded by upward movement can be replaced by PBC-obeying upward movement (once again because regularity is preserved).

Provided all these claims turn out to be correct, it seems that downward movement by itself does not add anything on top of upward movement unless the two are allowed to interleave. It will be interesting to see just how much power interleaving adds to MGs. Thanks to the MSO perspective and various theorems discussed in Sec. 2.1.4, however, we can already say that weak generative capacity will stay the same, and strong generative capacity won't exceed that of hyper edge replacement grammars.

This is an excellent example of how the factorization of MGs into derivation tree languages plus a transduction from derivations to derived trees grants the formalism a tremendous amount of flexibility without jeopardizing its most appealing characteristics. Changes can be made to the feature calculus as well as the transduction to squeeze in various ideas from the syntactic literature, as long as these alterations can be expressed via MSO formulas most results established for standard MGs carry over in a straightforward manner. Where they do not, it is often apparent why. The malleability of MGs is a great advantage and, as I argue in the next section, the main reason why there is little reason to worry about how faithful a model of Minimalist syntax they are — as simplistic as standard MGs might be, they can easily be tweaked and amended without endangering their formal core.

2.3 Evaluating the Adequacy of Minimalist Grammars

A crucial criterion for every mathematical formalization is whether it is faithful to the object under study. After all, if one models bumblebees as fixed wing aircrafts they provably cannot fly (cf. Magnan 1934; Dickinson 2001). Similarly, theorems about grammar formalisms are of little use if they depend on unrealistic assumptions about said formalism. For example, SPE is easily shown to be capable of generating any recursively enumerable string language (over the largest alphabet of symbols that are representable in SPE's feature system), yet the fragment used by phonologists in their daily work turns out to be finite-state and thus significantly better behaved (Kaplan and Kay 1994).

Naturally the relevance of MG research to syntacticians depends foremost on how faithful MGs are to Minimalist syntax. From what we have seen so far, it seems that the two are far removed from each other. But these differences are merely superficial, as I argue in the next few sections. For not only is it an easy task to change aspects of the MG formalism to bring it closer to Minimalist syntax, those details are in fact immaterial once one views MGs in terms of their derivation tree languages as previously described in Sec. 1.2.

I begin with a critical dissection of the common sentiment that mathematical results are not pertinent to theoretical syntax because of profound differences in scope and methodology. After that, I quickly move on to more concrete issues (which are less likely to coax me into the kind of "methodological moaning" derided in Pullum 1983). Proceeding in a bottom-up fashion, I start out with the feature calculus, Move, and locality before turning to derived trees, generative capacity, and possible extensions of the formalism. Readers that have no qualms about the relevance of MG research to mainstream syntax may skip ahead to the next chapter.

2.3.1 Relevance of Mathematical Results to Linguistics

Nowadays there is very little overt animosity between generative grammarians and mathematical linguists. But unfortunately this isn't a product of fruitful collaboration but rather a pronounced lack thereof — their peaceful cohabitation is grounded in (mutual) indifference. Syntacticians consider the concerns of mathematical linguists orthogonal to the generative enterprise, so that mathematical results go mostly unnoticed in the community at large.

As far as I can tell, the disinterest in mathematical approaches stems from the impression that they erroneously conflate two foundational distinctions or flat-out deny their significance:

- weak generative capacity (string languages) *vs* the phrase structure trees assigned to strings, and
- generated structures (E-language) *vs* the grammars generating those structures (I-language).

Chomsky himself has repeatedly insisted on the importance of phrase structure and I-language to the detriment of strings and E-language.

> The notion of weak generative capacity [...] has almost no linguistic significance. (Huybregts and van Riemsdijk 1982:73)
>
> Strong generation is the basic notion; *weak generation* of a designated set of strings [...] is defined as an auxiliary concept, of marginal significance at best. (Chomsky 1990:143)
>
> [...] E-language has no particular significance for formal linguistics. (Chomsky 1990:145)

Because mathematical results such as Peters and Ritchie (1971, 1973b) pertain only to the string languages generated by grammars, they are neither about phrase

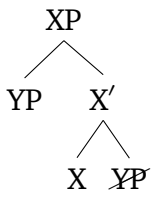
structure nor I-language and hence have nothing to contribute to generative syntax — or so the argument goes. This line of reasoning, however, does not apply to most recent formal work, and even for the research that fits this mold it only proves that mathematical linguists have not been very successful at communicating to syntacticians how their theorems bear on I-language.

In order to fully appreciate this point, we first need to clarify what it means for some result to "be about I-language". Given the equivocation of I-language and grammars, one might conclude that any result that tells us something about a grammar formalism satisfies this condition. But this is obviously not what syntacticians have in mind, since weak generative capacity results also increase our knowledge about a formalism yet are considered insufficient. Even moving from strings to phrase structure trees would keep us in the lesser domain of E-language results.

Identifying I-language with grammars entails at the very least that I-language allows for more fine grained distinctions than E-language: two distinct grammars might generate the same E-language, while the opposite does not hold. This is so because I-languages may differ in how they build certain structures.

Example 2.14 Two derivations that yield the same structure

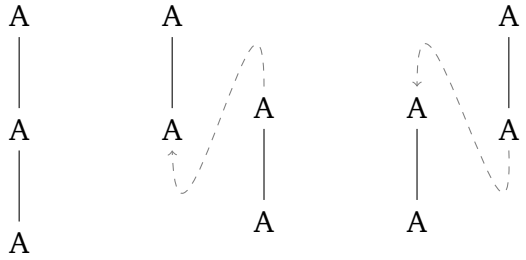
Consider the tree below, where ~~YP~~ is an unpronounced copy of YP:



Depending on our assumptions about Move, there are at least two distinct ways of generating this tree, namely overt raising of YP to Spec,XP and covert lowering of YP from Spec,XP.

This kind of ambiguity is not a peculiarity of movement-based formalisms. The

following tree can be assembled in Tree Adjoining Grammar using either substitution or adjunction:



Whatever the intended interpretation of the term I-language might be, it arguably subsumes results about how grammars assign phrase structure to strings. In technical parlance, this is tantamount to claims about derivation trees. Consequently, findings pertaining to derivation trees should qualify as I-language results.[2]

A quick glance at recent publications in mathematical linguistics shows that derivation trees take center-stage nowadays. Graf (2012b), for example, proves that if every instance of unbounded movement is decomposed into an equivalent sequence of movement steps such that each one only crosses a fixed number of maximal projections, the structural complexity of derivation trees is lowered significantly — a subtle hint that locality constraints do indeed simplify grammars. Kallmeyer (2009) successfully unifies half a dozen variants of Multi-Component TAG by treating them as standard TAGs with various constraints on their derivation trees, which raises the question if other operations, too, can be stated more insightfully in terms of derivational constraints. Stabler (2012) implements a parser for MGs that constructs

[2]There are of course components that have been considered pressing I-language issues at some point in the history of generative syntax, yet are not reflected in derivation trees as commonly construed — e.g. the structure of the lexicon and its interaction with syntax through the numeration. This is irrelevant to my point, though. If theorems grounded in derivation trees count as I-language results, then it suffices to give an example of one such theorem in the mathematical literature to disprove the claim that mathematical results are only about E-language.

derivation trees rather than derived trees and thus operates significantly faster, thereby demonstrating the practical viability of a syntactic approach that treats derivations rather than phrase structure trees as the basic data structure. For further examples see Shieber (2004), Kobele et al. (2007), Salvati (2011), and Kobele and Michaelis (2012). The notion that findings from mathematical linguistics are by necessity limited to E-language and devoid of any relevance for I-language is untenable in light of these recent developments.

In fact, the focus on I-language aspects isn't particularly novel. Equally expressive formalisms can differ along many dimensions that have been investigated in the literature, in particular parsing complexity and succinctness (where succinctness denotes the overall size of the grammar, or alternatively how easily certain structures can be derived). The good performance of the CKY-parser for CFGs, for instance, only obtains for CFGs in Chomsky normal form. For every non-deterministic finite-state automaton there exists a deterministic one that recognizes the same string language, but the blow-up in size is exponential in the worst case. For any given $k \in \mathbb{N}$, $\{a^n b^n \mid n \leq k\}$ can be generated by some regular grammar, but this grammar will be a lot more complicated than the CFG for $\{a^n b^n \mid n \in \mathbb{N}\}$; cf. Chomsky's (1957) analogous argument for favoring transformational grammar over CFGs. The copy-language $\{ww \mid w \in \Sigma^*\}$ can be defined in MGs without copying movement, but the derivations are hard to make sense of compared to those with overt copying. MGs and MCFGs have the same weak generative capacity, but the former are more succinct, supporting the idea that MGs capture certain generalizations about language missed by MCFGs. Since the very first day of formal language theory, there has been great interest in how weakly equivalent formalisms fare with respect to such properties, but for some reason this line of research garnered considerably less attention among linguists than weak generative capacity results.

Even where E-language is the object under investigation, the implications for I-language are numerous. This point is often missed in linguistic discussions because

the interesting bits of weak generative capacity results are in the proofs, which are usually too laborious for the uninitiated. Establishing the weak generative capacity of a given grammar formalism generally proceeds by exhibiting a procedure that translates it into another formalism whose expressive power is already known. It is the translations that provide us with insights about how those grammars work internally, which differences are fundamental and which but notation. For concrete examples, see Vijay-Shanker and Weir (1994), Kasper et al. (1995), Frank and Satta (1998), Fujiyoshi and Kasai (2000), Michaelis (2001, 2004), Rogers (2003), Stabler (2003), Mönnich (2006), Kasprzik (2007), Kanazawa et al. (2011), Graf (2012e), Kobele and Michaelis (2012). In particular, weak generative capacity results are of tremendous use in pinpointing a formalism's locus of power. For instance, the truly interesting insight of Peters and Ritchie (1971, 1973b) isn't that Transformational Grammar can generate all recursively enumerable languages, it is that weak generative capacity cannot be easily restrained via the base component (i.e. D-structure). This can be taken as evidence for the redundancy of the latter, which Chomsky (1993) independently argued for 20 years later. On the other hand, if the depth of derivations is linearly bounded by the length of the input string, only context-sensitive languages can be generated (Peters and Ritchie 1973b). If a specific notion of locality is introduced, the class generated by transformational grammars is even weaker and lies properly between the context-free and the context-sensitive languages (Peters and Ritchie 1973a; to my knowledge it is an open question whether all languages in said class are mildly context-sensitive). In these and many other cases, it is not the weak generative capacity itself that is of interest, but how certain aspects of the grammar, i.e. I-language, contribute to it.

Many more examples could be added to this list, but the preceding material should suffice to convince the reader that mathematical linguists seek to deepen our understanding of (I-)language just as Minimalists do. But noble intentions do not necessarily yield something good, and at least some syntacticians seem to think

that little of actual value has come out of the formally rigorous approach — not because of some fundamental failure to focus on I-language, but because some of the assumptions made in the formalization process are *ad hoc*. As is to be expected, Chomsky himself has been very vocal about this topic:

> To formalize one or another version [of X′-theory; TG] is a straightforward exercise, but [...] would require decisions that are arbitrary; not enough is understood to make them on principled grounds. The serious problem is to learn more, not to formalize what is known and make unmotivated moves into the unknown. [...] One should not be misled by the fact that computer applications require such moves. Throughout history, those who built bridges or designed airplanes often had to make explicit assumptions that went beyond the understanding of the basic sciences. (Chomsky 1990:147)

There are several assertions in this statement that one could take issue with, and at first it feels at odds with the push towards more rigorous theories that Chomsky spearheaded in the late 50s.

> Precisely constructed models for linguistic structure can play an important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact source of this inadequacy and, consequently, gain a deeper understanding of the linguistic data. More positively, a formalized theory may automatically provide solutions for many problems other than those for which it was explicitly designed. Obscure and intuition-bound notions can neither lead to absurd conclusions nor provide new and correct ones, and hence they fail to be useful in two important respects. (Chomsky 1957:5)

Upon a more nuanced reading, though, it should be clear that the passage from

Syntactic Structures is not necessarily at odds with Chomsky's later statement; the former expounds the advantages of formalization, whereas the latter addresses the issue of how rigorous a formalization is needed in order to reap those benefits. So the problem with mathematical approaches isn't one of principle, but merely practicality: given that learning mathematics has a steeper learning curve than not learning mathematics, does the initial time investment pay off in the long run?

> Even in mathematics, the concept of formalization in our sense was not developed until a century ago, when it became important for advancing research and understanding. I know of no reason to suppose that linguistics is so much more advanced than 19th century mathematics or contemporary molecular biology that pursuit of Pullum's injunction [i.e. rigorous formalization of linguistic proposals; TG] would be helpful, but if that can be shown, fine. (Chomsky 1990:146)

> It could turn out that there would be richer or more appropriate mathematical ideas that would capture other, maybe deeper properties of language than context-free grammars do. In that case you have another branch of applied mathematics, which might have linguistic consequences. That could be exciting. (Chomsky 2004b:43)

Many of the examples I gave above easily rise to Chomsky's pragmatist challenge. They show that mathematical linguistics isn't merely the study of whether formalisms fall into some computationally appealing class. Mathematical linguistics is about how the internals of a proposal interact to yield its generative power, what kind of structural dependencies it can express and which of them are instantiated in natural language, how these dependencies may be represented in a succinct fashion, how they are computed, whether they are learnable. These goals are in perfect agreement with the generative enterprise. The dividing line between mathematical linguistics and generative grammarians isn't the object of study, it is methodology.

The crucial question, then, is whether the methodological choices of mathematical linguists — characterized by abstraction and a preference for generalizations over entire classes of grammars and languages rather than specific instances thereof — mean that their work is too far removed from the ideas of syntacticians to be of any value to them. In the case of MGs, there are several technical simplifications that are apparently at odds with Minimalist proposals. In the next few sections, I show that these simplifications are merely a matter of mathematical convenience. If desired, MGs can be altered in various ways to make them more faithful to Minimalist syntax without changing their central properties (which were discussed in Sec. 2.1).

2.3.2 Feature Calculus

The most immediate difference between MGs and Minimalist syntax lies in the status and make-up of features. MG features are

- binary valued, and
- always involved in triggering a syntactic operation, and
- mandatorily erased after triggering said operation, and
- relevant to either Merge or Move, but not both, and
- linearly ordered, and
- able to occur several times on a single LI, and
- inseparable from their LIs.

The feature system of Minimalist syntax has undergone several revisions since its inception; yet at no point in that development did it exhibit a majority of the properties of the MG feature calculus.

Regarding the very first point in the list above, it is unclear whether features in Minimalism are privative (i.e. without value), binary valued, or multivalued. The only option that is ruled out is HPSG-style recursive feature structures (cf. Adger 2010). In addition, features in early Minimalism are distinguished according to their interpretability in a manner that does not line up well with the polarity bifurcation of MGs. In Minimalism, only Move needs to be triggered by uninterpretable features, but then it does not matter whether only one feature is uninterpretable or both of them. Moreover, only uninterpretable features are erased during the checking procedure (cf. example 1.1 on page 9). This means that Minimalist syntax lacks two instances of symmetry that are present in the MG feature calculus, as the latter allows only features of opposite polarities to enter a checking relation but always erases both of them.

MGs also diverge from Minimalism with respect to their basic ontology of features. As Kobele (2005) points out, MGs treat features as building blocks from which LIs are assembled. As such, they can be ordered and used multiple times for the same LI. Minimalist syntax, on the other hand, views features as properties of LIs, and since it makes little sense to establish an order on properties or claim that an object has a property several times, neither is allowed.[3] The disagreement on the metaphysics of features by itself is of little significance, but the two technical differences that arise from these opposing views — feature order and iterability — cut too deep to be ignored so easily.

[3]There is at least one notable exception to this claim. In his discussion of Icelandic, Chomsky (1995c:286,354) stipulates that a feature may have a meta-property that specifies how often the feature needs to be checked before it can be erased. This is a notational variant of allowing multiple features on the same LI, as was already observed by Nunes (2000).

To a certain degree, the modern Agree-based system with its reliance on OCC features is also inconsistent with this view, provided one adopts Chomsky's (2004a) position that phases are representational, i.e. all operations that take place inside a phase do so simultaneously. For then an LI needs multiple OCC features to trigger multiple instances of Move (which is still permitted in this system), and since there is no derivational ordering anymore, these features are all present at the same time.

Besides the major divergences regarding symmetry and the ontology of features, there are also many minor ones. In Minimalism, category features can be involved in both Merge and Move (the latter when checking an EPP/OCC feature), features may detach from their LIs in order to move on their own, and multiple features can be manipulated by a single operation if they are bundled (e.g. in the case of ϕ-features). For the sake of completeness, one could even add the mostly forgotten distinction between Delete(α) and Erasure (Chomsky 1995c:280) to this list. Finally, almost all semblance of a correspondence between the two feature systems seems to be lost once one takes the step from classic Minimalism to the contemporary system of (un)valued features introduced in Chomsky (2001). Considering the important role of features in Minimalist syntax, then, all these discrepancies cast doubt on the faithfulness of MGs.

Fortunately, there is no reason for concern. While it cannot be ruled out that the peculiarities of the MG feature calculus might cause MGs to be too far removed from Minimalism to provide insightful answers to some questions (although none come to mind), this is not the case for any of the issues covered in the MG literature so far, including this very thesis. This can be shown in two ways. The more intuitive one is to provide explicit translations from more faithful systems to the standard MG one.

Example 2.15 MGs with unordered features 

Suppose that features are no longer put in a linear order. So a given collection of LIs can now combine into several derivations, depending on the order in which the features are checked. The same behavior can be emulated by a standard grammar G in which every LI is multiplied out into all its feature permutations. That is to say, if a :: f_1 f_2 f_3 is an LI of G, then so are a :: f_1 f_3 f_2, a :: f_2 f_1 f_3, a :: f_2 f_3 f_1, a :: f_3 f_1 f_2, and a :: f_3 f_2 f_1. Syntax — rather than being forced to guess the order in which features must be checked to yield a well-formed derivation — now has to make a

choice about which of these minimally different LIs should be used when building a derivation.

In the other direction, every MG with ordered features can be emulated by one with unordered features. Intuitively, one can simply decompose a head with multiple unordered features into a hierarchy of heads that each host one of the features of the original head, similar to analyses that posit distinct probes for person and number (Béjar and Rezac 2009). The strategy works as intended due to a few key properties of MGs. First, observe that if an LI l has at least one selector feature, one of them is the feature that must be checked first as there is no other way of introducing l into the derivation. Second, l's licensor features must be checked before its category feature due to the ban against countercyclic movement. Third, the licensee features cannot be checked before the category feature because movement is to a c-commanding position, and the only way to create such a position is for some other LI to select l and thereby delete its category feature. Finally, for every MG there is an equivalent one in which every LI moves at most once.

Therefore every LI has a feature string in the regular language $(s\gamma)c(m)$, where s is a selector feature, c a category feature, m a licensee feature, and γ a possibly empty string of selector and licensor features. It is easy to decompose each LI into a hierarchy of LIs that each contain at most three features: a selector feature, a category feature, and possibly a selector, licensor or licensee feature. For example, $\mathrm{l} :: {=}\mathrm{s}\ {=}\mathrm{t}\ {+}\mathrm{x}\ \mathrm{a}\ {-}\mathrm{y}$ will be decomposed into

$$\begin{aligned}
&\mathrm{l} :: {=}\mathrm{s}\ \mathrm{a}^{\mathrm{l}}_0\\
&\varepsilon :: {=}\mathrm{a}^{\mathrm{l}}_0\ {=}\mathrm{t}\ \mathrm{a}^{\mathrm{l}}_1\\
&\varepsilon :: {=}\mathrm{a}^{\mathrm{l}}_1\ {+}\mathrm{x}\ \mathrm{a}^{\mathrm{l}}_2\\
&\varepsilon :: {=}\mathrm{a}^{\mathrm{l}}_2\ \mathrm{a}\ {-}\mathrm{y}.
\end{aligned}$$

Given our previous remarks, the features will also be checked in this order in any MG with unordered features as long as a head can still distinguish between arguments depending on whether they should be complements or specifiers — an ubiquitous assumption in the literature. Note that no LI in these refined MGs has the same feature twice, so we can also drop iterability of features without negative repercussions.

Admittedly the lexicons created this way are huge and highly redundant, and the reader would be justified to have reservations about their psycholinguistic adequacy. But our sole goal here is to ensure that results about standard MGs carry over to the variant without ordered, iterable features. The translation procedures reveal that dropping these properties does not change anything about the formalism, both systems are interchangeable. So any interesting property of standard MGs also holds for these variants.

With a little creativity, similar translations can be devised to deal with other points of divergence. Discarding with symmetric feature checking is a little bit more difficult, but see the remarks on *persistent features* in Stabler (2011). We can conclude that even though the MG feature calculus looks, at best, like a crude imitation of the original Minimalist feature system, it is in fact perfectly capable of emulating more faithful variants.

The alternate route of establishing the adequacy of the MG feature calculus is more general but also less intuitive: the feature system cannot be unfaithful because its technical details are irrelevant. Recall from Sec. 1.2 that a given MG is fully specified by its derivation tree language, from which the set of phrase structure trees is obtained by a specific transduction that moves subtrees from their base position to their final landing site. The viability of this view is tied to three formal requirements

established in Sec. 2.1:

- the derivation tree language must be a regular tree language, and
- every Move node is an occurrence for exactly one LI, and
- the transduction from derivation trees to derived trees must be MSO-definable.

The MG feature calculus is just a means towards ensuring that these three properties hold. There are infinitely many alternative systems that could be used in its place, the original Minimalist system being but one of them. Consequently, one should not attach too much importance to the peculiarities of a given feature system. For our purposes, almost all of them will do, irrespective of whether they follow standard MGs in treating features as building blocks and feature checking as invariably symmetric.

2.3.3 Movement

The discussion of the feature system immediately leads to another problem of MGs: successive cyclic movement. As the astute reader might have already observed, the restriction to a bounded number of features for each LI prevents successive cyclic movement in a system where every instance of Move must be triggered by some feature that is subsequently deleted. This is so because successive cyclic movement may involve an unlimited number of intermediate landing sites, wherefore no LI could ever carry enough features to move through all intermediate landing sites for every well-formed derivation.

Kobele (2006) offers a slight reinterpretation of successive cyclic movement that does away with this problem. He proposes that only the final target site contains a triggering feature, and all intermediate landing sites are filled with a (covert) copy of the moving subtree during the mapping from derivation trees to derived trees.

So even though successive cyclic movement behaves like one-fell-swoop movement with respect to the feature calculus, it still yields the desired phrase structure trees.

An alternative solution has already been touched upon in the previous subsection: the difficulties with successive cyclic movement stem from specific assumptions inherent in the standard MG feature system, which can be replaced by more suitable alternatives without changing anything significant about the formalism. By extension, successive cyclic movement is a challenge only for a specific MG feature calculus, not for MGs as such.

Example 2.16 Successive cyclic movement 

There are many ways of reigning in successive cyclic movement at the derivational level. As an illustration of how this can be accomplished, I present one specific strategy here.

In order to handle unlimited occurrences, the MSO transduction from derivations to derived trees needs to be modified. This can be done without endangering MSO-definability. First, we create an MSO transduction τ from standard MG derivations to derived trees with successive cyclic movement. This transduction only differs from the canonical one in that for every instance of movement crossing a landing site of successive cyclic movement (usually vP and CP), a trace is inserted at this position, following Kobele (2006). Thanks to the SMC only a bounded number of phrases move across such a landing site, wherefore the desired modification can easily be expressed in MSO terms.

In the next step, successive cyclic movement is introduced at the derivational level via a mapping τ' that inserts Move nodes in a fashion parallel to τ. An MG derivation with successive cyclic movement thus can be obtained from one with unbounded movement via τ'. Since MSO transductions are closed under composition and reversal, $\tau'^{-1} \circ \tau$ is also an MSO tranduction. This transduction translates MG

derivations with successive cyclic movement into MG derivations with unbounded movement and maps them to the intended derived trees. Hence MGs with successive-cyclic movement are the image of standard MGs under τ', and their derived tree languages are computed by $\tau'^{-1} \circ \tau$.

Another major concern about Move in MGs is its determinism. Determinism is an immediate consequence of the SMC, which does not allow for two LIs to have the same licensee feature as their first active feature at the same step in the derivation. Under this restriction there can be no scenario where two LIs are both eligible to move to a specific target site such that syntax has to make a choice as to which one gets to move. But those cases are widely attested and serve as the major motivation for various locality principles in the syntactic literature. The saving grace of MGs in this case is that the derivational determinism holds only with respect to a given choice of LIs. What might be considered a single LI in the syntactic literature may correspond to several LIs in an MG, all of which differ slightly in their feature make-up. So MGs still allow for non-determinism, but it is pushed exclusively into the initial choice of LIs from which the derivation is to be assembled.

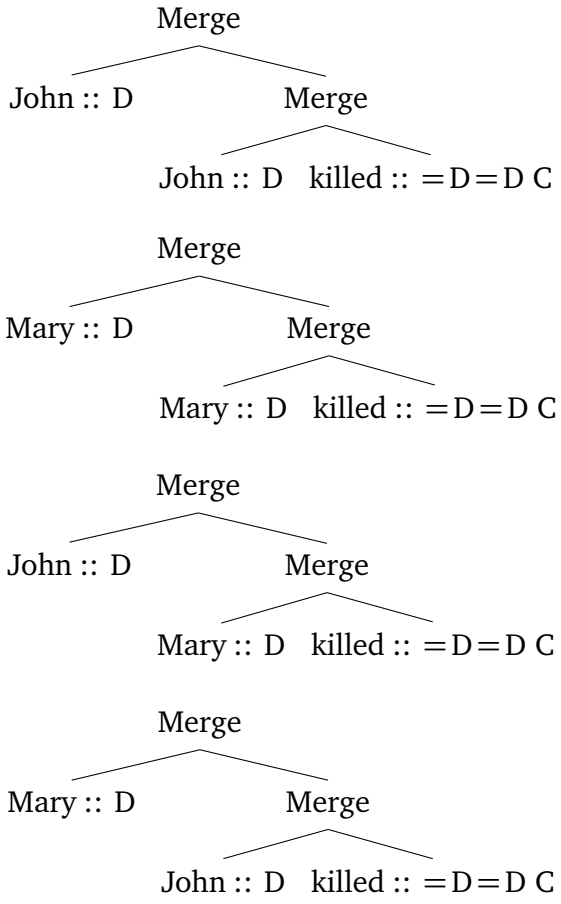
Purely lexical non-determinism is not widely entertained in the syntactic literature, though. This is motivated by the following line of reasoning: if a derivation is fully specified by the LIs that occur in it, there should be no reason for syntax to construct a derivation rather than simply pass the set of LIs directly to the interfaces. Full lexical determinism is claimed to render syntax redundant. But this is incorrect. A given set of LIs may still allow for multiple distinct derivations to be constructed. In fact, any grammar in which at least two LIs have the same feature specification has a limited amount of uncertainty so that derivation trees are indispensable for disambiguation.

Example 2.17 Disambiguation via derivation trees

Consider the MG consisting of the following three LIs.

$$\text{John} :: \text{D} \quad \text{Mary} :: \text{D} \quad \text{killed} :: {=}\text{D}{=}\text{D}\ \text{C}$$

Exactly four derivations are licensed by this grammar, two of which use all three LIs. Those two only differ in which DP is merged as the subject/object.



Case features and theta roles reduce non-determinism, but cannot fully eliminate it, e.g. in *John knows that Bill likes him* versus *Bill knows that John likes him*. Playing devil's advocate, one might propose that this kind of disambiguation could be handled by semantics, wherefore derivations are still redundant. This argument has

two crucial flaws. First, it blurs the distinction between competence and performance. There is little doubt that when planing their utterance, the speaker knows the intended LF to some degree and may use this information to limit non-determinism. But generative theories of syntax only model competence, which sentence planning and production take no part in. Second, even if the speaker had full access to all semantic information in syntax, this does not spell the end of all non-determinism because multiple derivations may yield the same LF. This is the case for *John and Mary kissed* versus *Mary and John kissed*. The idea that derivation trees provide no information over the set of LIs that they are constructed from simply isn't tenable even if Move is deterministic.

In sum, MGs allow for all the non-determinism needed in the empirical analysis of various phenomena related to locality, but they relegate said non-determinism to the choice of LIs rather than syntax proper.

2.3.4 Locality

Issues of locality have already been implicitly touched upon in the discussion of successive cyclic movement, which proved a little tricky to capture in MGs — in stark constrast to unbounded movement, which is unproblematic in MGs. This is exactly the opposite of the Minimalist postulate that local movement is the default and seemingly unbounded movement is to be reinterpreted as a sequence of local movement steps. Apparently, then, MGs are too lax regarding locality.

At the same time they are also too strong, as the SMC forbids any two LIs from having the same licensee feature as their first unchecked feature at any point in the derivation. Among other things, this precludes configurations where two DPs are capable of checking the wh-feature of a C-head, as in $[_{\text{CP}}\ \text{C}_{\text{[+wh]}}\ [_{\text{TP}}\ \text{who}_{\text{[-wh]}}\ \text{T}\ [_{\text{VP}}\ t_{who}\ \text{bought what}_{\text{[-wh]}}\ ]\ ]\ ]$. Cases like this are the prime motivation for relative notions of locality in Minimalist syntax. MGs, on the other hand, apparently throw

out the baby with the bathwater by having the SMC block all problematic structures.

Both the lack of locality and the excessive strength of the SMC can be fixed in a very general way building on the results I establish in Chap. 3. At this point it suffices for the reader to know that locality conditions can be stated as (MSO-definable) constraints over derivation trees, and once these locality restrictions are in place, the SMC can be relaxed so that it allows for multiple active instances of the same licensee feature (but only a finitely bounded number thereof). MGs with a relaxed SMC and the right choice of locality constraints behave exactly like Minimalist syntax in cases such as the one above.

Note, though, that we cannot completely do away with the SMC without losing the restriction to regular derivation tree languages (see Salvati 2011 for a detailed study of MGs without the SMC). As the well-behaved nature of MGs stems mostly from the regularity of their derivation tree languages, some variant of the SMC must remain in place. The repercussions are minimal in practice. Issues arise only for unbounded multiple wh-movement, where an unlimited number of wh-phrases may move to the same CP. But even then the problems are restricted to specific analyses of the phenomenon. The standard account treats multiple wh-movement like any other instance of wh-movement and thus requires an unbounded number of licensee features to be active at the same time (one feature per moving wh-phrase). Weakening the SMC in a manner so that it permits such cases yet still ensures that all derivation trees are regular is impossible in the general case.

Even though unbounded multiple wh-movement is incompatible with the SMC under the standard Minimalist analysis, this does not mean that the SMC precludes any viable account of this phenomenon. A clustering approach to multiple wh-movement (Gärtner and Michaelis 2010), for instance, is an attractive alternative for MGs as it only needs a bounded number of licensee features at any given step in the derivation and thus obeys the SMC. The SMC thus turns out to be rather innocent (albeit sometimes inconvenient) for linguistic purposes — by adding locality

restrictions the SMC can be relaxed in a way that is more faithful to Minimalist syntax, provides adequate empirical coverage, and preserves the regularity of Minimalist derivation tree languages.

2.3.5 Derived Trees

The derived trees of an MG closely resemble standard Bare Phrase Structure trees except for I) their interior node labels and II) the fact that they are linearly ordered.

As with the specifics of the feature calculus, the choice of interior node labels is of little relevance and can be altered as desired. Strictly speaking, one could dispense with interior node labels altogether, yielding a kind of label-free syntax as envisioned by Collins (2002). Contrary to what Collins's work might lead one to expect, this changes nothing about the internal MG mechanics. These are still determined by derivation trees, whose interior node labels are redundant anyways; they only indicate the type of operation, which can easily be inferred from a node's arity. In a sense, MGs are already label-free, and that is why tinkering with interior node labels is of no consequence.

Example 2.18 MGs are label-free

The interior node labels of the derived trees are obtained via the MSO-mapping described in Sec. 1.2. The relevant two formulas are repeated here.

$$> (x) \leftrightarrow \text{Move}(x) \vee (\text{Merge}(x) \wedge \neg \exists y [x \triangleleft y \wedge \mathit{Lex}(y) \wedge x \sim y])$$

$$< (x) \leftrightarrow \text{Merge}(x) \wedge \exists y [x \triangleleft y \wedge \mathit{Lex}(y) \wedge x \sim y]$$

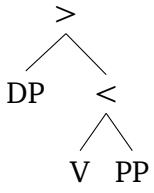
It is easy to see that any symbol could have been used instead of $<$ and $>$ without changing anything essential about the respective MSO formulas. In particular, if $<$ and $>$ are replaced by the same symbol, all interior nodes have the same label. But

if all interior nodes have the same label, this label has no information to contribute, just as if they had no label at all.

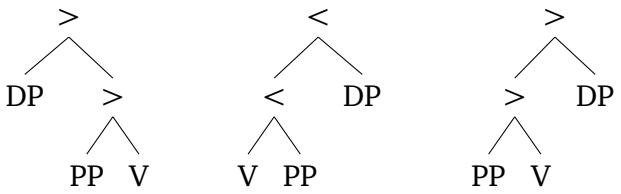
The requirement that derived trees be ordered is also mostly immaterial from a linguistic perspective, but in contrast to the labeling mechanism, there is a technical advantage to it. Unordered trees are a lot more cumbersome mathematically than ordered ones while offering no new insights of linguistic relevance. This is not meant to be a slight against Kayne's (1994) seminal work which posits that string precedence is determined by c-command rather than some precedence relation defined over nodes in a tree. Keep in mind that the import of Kayne (1994) is not whether trees are ordered, it is that string precedence is independent of any such ordering relation. Since c-command relations are invariant with respect to tree precedence, it is irrelevant for Kayne's proposal whether trees are ordered. Granted, it is slightly odd to require an ordering relation that is ultimately irrelevant for the formalism. On the other hand, an unordered tree is indistinguishable from an ordered one whose order is irrelevant, but the latter is more convenient to work with. Readers who are uncomfortable with ordered derived trees may assume that the derived tree language of an MG is actually the sibling permutation closure of the set defined in Sec. 1.2.3. That is to say, the derived tree language contains each tree in all its possible orderings.

Example 2.19 Sibling permutation closure of derived trees

Suppose the following tree is in the derived tree language of some MG G.



Then the sibling permutations of this tree are also in G's derived tree language.



2.3.6 Generative Capacity

A common complaint about any kind of formal grammar mechanism, including MGs, is that it subsumes many unnatural grammars and languages; in other words, it overgenerates both structurally and on the string level.

For example, there are infinitely many MGs whose LIs are phonetically null, so that a moment of silence could be assigned the same tree structures as the collected works of Shakespeare. In addition, nothing in the formalism commits us to at most two arguments per head, the functional hierarchy v-T-C, or that subjects aren't based-merged in T. Linear order is also too unrestrained since the class of MG string languages is closed under reversal, so for any given word order that can be derived by MGs, its mirror image is derivable, too. Other unattested patterns that can be generated by MGs are

- for every $n \in \mathbb{N}$, the language $\text{MOD}_n := \{w \in \Sigma^* \mid |w| \bmod n = 0\}$ of strings whose length is a multiple of n,
- the MIX-language $\text{MIX} := \{w \in \{a,b,c\}^* \mid |a| = |b| = |c|\}$, which contains all strings over $\{a,b,c\}$ in which all three symbols occur equally often (Salvati 2011; Kanazawa and Salvati 2012),
- the copying-language $\left\{w\#w\#w \mid w \in D_1^*\right\}$, where D_1^* (the one-sided Dyck

language over one pair of parenthesis) is the set of well-balanced bracketings over {[,]}, e.g. [] and [][[][[][]]] but not][or [[] (Kanazawa and Salvati 2010).

While MOD_2 might have a role to play in natural language insofar as the distinction between odd and even matters for stress assignment in some languages (Graf 2010a), more fine-grained distinctions are not attested (e.g. a language that requires clefted constituents to contain three, six, nine ... words). The MIX-language is an abstraction of unrealistic free word-order patterns such as a hypothetical language with Dutch-style cross-serial dependencies where the order of words is completely unrestrained. The last language represents cases where embeddings of unbounded depth are copied and fully realized in three distinct positions in the utterance. In sum, MGs are independent of well-established linguistic assumptions, fail to capture certain complexity bounds on attested patterns, and are capable of modeling elaborate dependencies that are simply unheard of. But is this actually a problem?

MGs' independence from established syntactic knowledge is perfectly acceptable. To expect otherwise is an instance of the *ferris wheel* fallacy: that there is some set of tools that is powerful enough to construct ferris wheels, but nothing else. Excessive malleability is an inevitable shortcoming of every scientific theory. Einstein could have written any formula, but he wrote $e = mc^2$. OT could model many kinds of constraint-ranking problems, but the set of posited constraints turns it into a theory of phonology. The basic formalism only has to carve out a class of grammars satisfying properties that are necessary in order for them to qualify as natural language grammars. Said properties might not be sufficient, though, which is why additional, linguistically motivated stipulations have to be put in place.

Ideally, all extra stipulations will eventually fall out from grammar-external restrictions. In Minimalist syntax, these are often thought of in terms of Bare Output Conditions imposed by the interfaces, while mathematical linguists are primarily looking at learning and parsing constraints. Parsing complexity can vary between

grammars even if they are weakly equivalent (recall that context-free grammars in Chomsky Normal Form are easier to parse), and in some cases, structurally more complex patterns are actually easier to parse (Joshi 1990; Kobele et al. 2012). This suggests that the class of efficiently parsable MGs is a proper subset of all MGs. Similarly, the classes of Gold-/PAC-/MAT-learnable MGs are necessarily proper subsets of the entire MG class. Finally, there might be certain succinctness conditions on grammars that rule out some of the more unwieldy ones. The grammar formalism is just one of several parameters that jointly pick out the class of natural language grammars, wherefore some amount of overgeneration is not too much of an issue at this point.

It should also be noted that the examples of overgeneration above only consider weak generative capacity, which — while suggestive — is far from conclusive. For example, it is far from obvious that the reversal of a given string language can be assigned a natural constituent structure in MGs. In this regard it is noteworthy that if one adopts the basic phrase structure template advocated by Minimalists, only certain word orders can be generated by MGs (Stabler 2011). Similarly, many formal string languages might have to be assigned phrase structures that are highly dubious from a linguistic perspective.

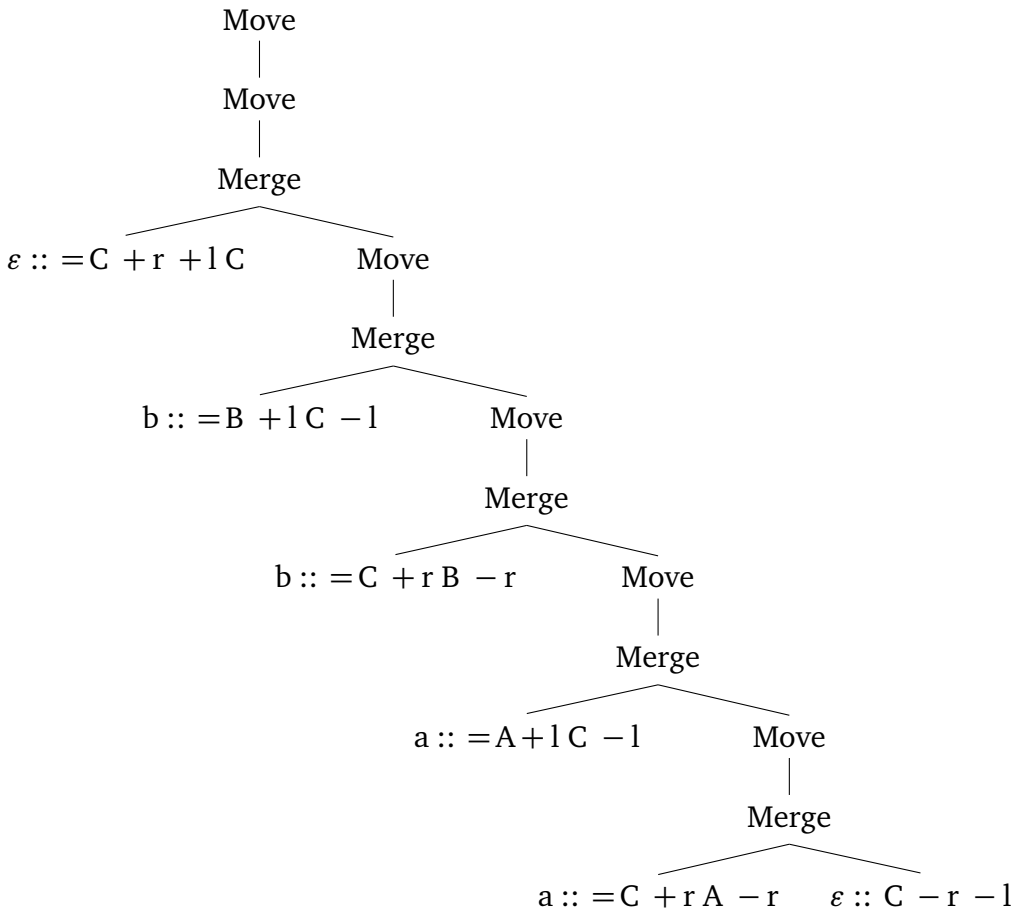
Example 2.20 An MG for the copy language

The copy language $\{ww \mid w \in \{a, b\}^*\}$ consists of all strings of the form ww, where w is some arbitrary string over symbols a and b. In other words, it contains every string that is the result of taking a string w and appending a copy of said string. Hence aa and $babbbabb$ are in the language, but a, abb and $babbbab$ are not. The empty string ε is a special case: since $\varepsilon = \varepsilon\varepsilon$, it is a member of the copy language. The copy language is not context-free, but it is mildly context-sensitive (as a matter of fact, it is a TAL). This means that standard MGs can generate the copy language

even though they lack copying movement. However, the MG for the copy language is very cumbersome.

$$\begin{array}{ll} \varepsilon :: \mathrm{C} -\mathrm{r} -\mathrm{l} & \varepsilon :: =\mathrm{C} +\mathrm{r} +\mathrm{l}\,\mathrm{C} \\ \mathrm{a} :: =\mathrm{C} +\mathrm{r}\,\mathrm{A} -\mathrm{r} & \mathrm{b} :: =\mathrm{C} +\mathrm{r}\,\mathrm{B} -\mathrm{r} \\ \mathrm{a} :: =\mathrm{A} +\mathrm{l}\,\mathrm{C} -\mathrm{l} & \mathrm{b} :: =\mathrm{B} +\mathrm{l}\,\mathrm{C} -\mathrm{l} \end{array}$$

This grammar derives the string $abab$ via 5 Merge and 6 Move steps.

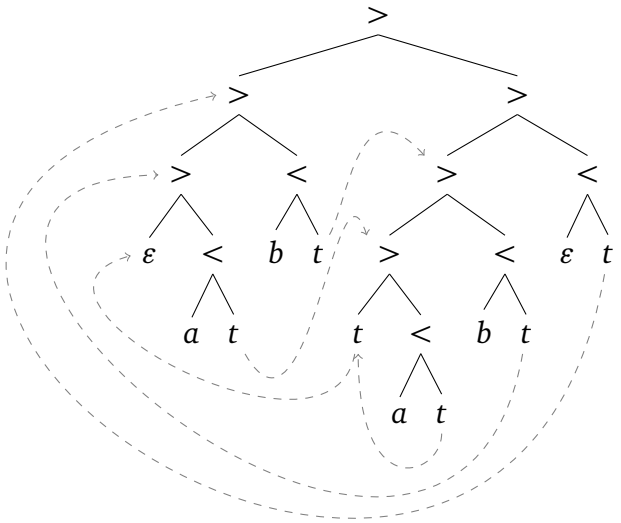


The grammar distributes the workload in a somewhat opaque manner between Merge and Move. First, the way the category features are distributed, *a*s and *b*s can only be generated in pairs of two. For instance, if a new a is to be introduced

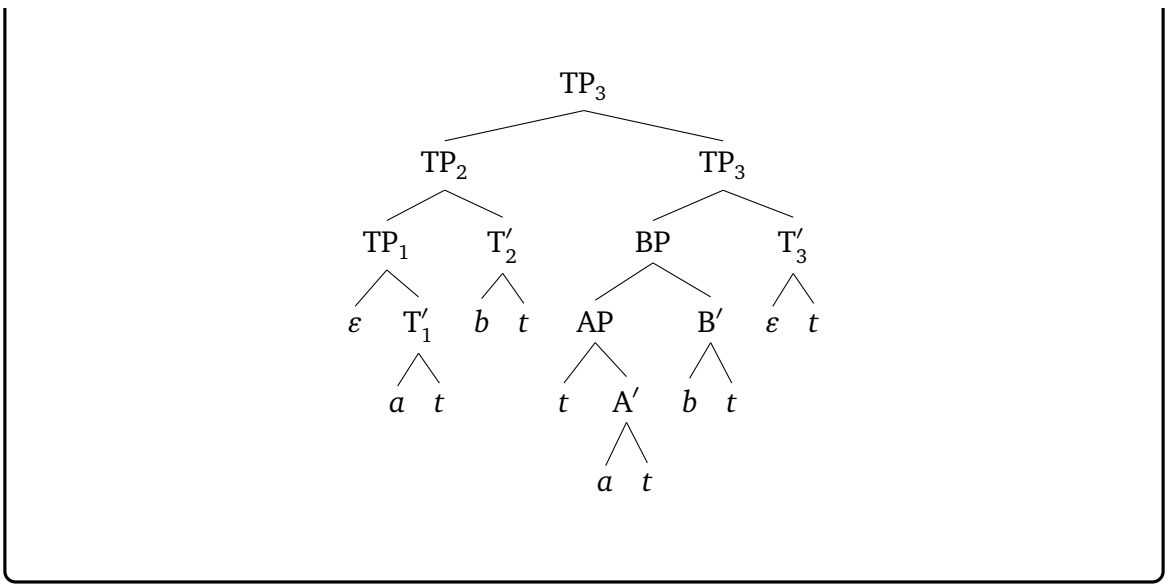
into the derivation it must be via the LI a :: =C +r A −r. This LI selects a C and is itself of category A. But the only thing that selects an A is the LI a :: =A +l C −l. So everytime an a is introduced, a second one follows right in its footsteps.

If the grammar only used Merge, then this procedure would produce $bbaa$ rather than $abab$. This is where the movement features come in. They ensure that the two as must move to different locations, one to an l licensor, the other one to an r licensor (the feature names are mnemonics for left and right). In the example above, those are provided by the two bs. This yields the desired $abab$. So while Merge creates LIs in packs of two, Move breaks them up and moves them into different positions.

The derived phrase structure tree looks fairly reasonable at first glance. But adding explicit movement arrows for each instance of Move brings to light the complexity of the underlying structure building process.



In this case the MG labeling mechanism also hides the fact that the two subtrees aren't perfect copies of each other due to the difference in category features.



One should also keep in mind that weak overgeneration is less of an issue than strong undergeneration. If a desirable increase in a formalism's strong generative capacity comes at the cost of weak overgeneration, this is a price worth paying. For example, the language in example 2.20 is straightforwardly generated by an MG with overt copying movement. Adding overt copying, however, increases the weak generative capacity of MGs to that of PMCFGs (see Sec. 2.1.2), so the number of unnatural string languages generated by these grammars is even bigger. Some mathematical linguists hence are reticent to take this step because it is more difficult to gauge what level of strong generative capacity is truly needed. However, if we reject regular and context-free grammars because of their inability to assign linguistically adequate constituent structures, it is unclear why we should hesitate to do the same for more powerful formalisms where necessary.

2.3.7 Missing Components

So far we have looked at things that MGs do differently from Minimalist syntax, but there are also many parts of the latter that are completely missing from the former: Agree, Head Movement, Sideward Movement, across-the-board movement, adjunc-

tion, Late Merger, feature percolation, multi-dominance structures, phases, binding theory, island constraints, and many more. Various movement types have been incorporated over the years (Stabler 1999, 2003, 2006; Kobele 2008), and Graf (2012c) gives a system for defining arbitrary movement types without increasing weak generative capacity. Among these movement types are standard, multi-dominance, and copying versions of raising, lowering, and sideward movement for phrases as well as heads, so that all variants of Move currently entertained in the literature can be added to MGs. For results on adjunction and feature percolation see Frey and Gärtner (2002), Gärtner and Michaelis (2007), Stabler (2011) and references therein. The status of constraints and locality conditions in MGs is the subject of this very thesis and will be explored in the next few chapters, but suffice it to say that the class of constraints that can be added to MGs without increasing their generative capacity (both weak and strong) is surprisingly big and includes the majority of constraints proposed in the literature. Overall, numerous extensions of MGs have already been explored, and with a few exceptions like Late Merger, they have little effect on the crucial formal properties of MGs: the regularity of their derivation tree languages and the MSO-definability of the mapping to derived trees.

This raises the question, though, why one should continue to work with the bare bones version of MGs rather than adopt a more faithful version as the new standard. If the computational properties of MGs are preserved by all these extensions, surely there is no harm in doing so while increasing our understanding of the actual formalism by reducing the abstractness and cumbersome nature of canonical MGs. To linguists, this line of reasoning sounds very plausible. A head movement analysis, for example, might require fewer movement steps than the corresponding remnant movement analysis. As a matter of fact, the latter might be extremely complicated. But “complicated” is not the same thing as “complex” from a mathematical perspective. A derivation with 15 phrasal movement steps could be considered more complicated than one with 5 instances of phrasal movement and 5

instances of head movement. For the purposes of mathematical inquiry, however, the grammar that only uses phrasal movement is less complex than the one that allows for both phrasal movement and head movement. The fewer movement types there are, the fewer cases need to be considered when constructing proofs. The simpler the movement types, the easier it is to reason about them. If there is only one movement type, then one does not even have to worry about how different movement types might interact. The actual number of movement steps in a derivation, on the other hand, is irrelevant thanks to various proof techniques such as induction. In other words, mathematical work profits from reducing the complexity of the grammars rather than the structures they produce.

Besides simplifying proofs, sticking with a frugal formalism also aids in analyzing the individual subcomponents. For instance, the weak equivalence of MGs and MCFGs was established for MGs with phrasal movement only, and the generative power of MGs with head movement was then established via a translation between the two MG variants. So these results were obtained via two proofs. Now suppose that the proof of the weak equivalence of MGs and MCFGs had made use of both remnant movement and head movement. In that case, the obvious question would have been whether the expressivity of MGs depends on remnant movement, head movement, or their interaction. This in turn would have required constructing two new proofs, one for MGs with remnant movement and one for MGs with head movement. Hence three proofs would have been necessary rather than just two. The additional effort is required because one cannot readily deduce the properties of specific subparts of the machinery from the behavior of the system as a whole. Quite generally, the more parameters and modules a formalism consists of, the harder it is to determine their respective contributions. By keeping the formalism as minimal as possible, one gets a good understanding of its components, which subsequently makes it easier to analyze more complicated versions.

Claims about a bare-bones formalism are also more general because most new

variant will be extensions of the original version. Since standard MGs can generate all MCFLs, MGs with more movement types can, too. The derivation tree languages of an extended MG variant cannot be less complex than the derivation tree languages of standard MGs. By sticking to a formalism F that is as simple as possible, one reduces the odds that there is some variant F' such that neither is an extension of the other. This would be the worst case scenario, since none of the theorems about one might readily carry over to the other, requiring significant extra work. Note also that there is no maximally complex variant that subsumes all others, so picking a very simple one is the safest route towards maximally general results.

The last point would be irrelevant if all syntacticians were in perfect agreement about all technical aspects of Minimalism. For then there would be exactly one universally accepted version of the theory and studying Minimalism would be tantamount to studying this very theory. Of course this is not the case, and many different competing proposals have been put forward. Even if there was only one version of Minimalism at any given point in time, this version would still have to change continuously as new data is discovered. This is why there is little point in devising a variant of MGs that encompasses every detail of Minimalism as defined in, say, Chomsky (2001). Minimalism, like any scientific theory, keeps evolving, and thus it is only prudent to set up MGs in a way that makes them compatible with as many conceivable incarnations of Minimalism as possible.

The strategy of ensuring compatibility via simplicity also obviates Chomsky's (1990) criticism that formalization involves fixing certain aspects of the theory that were deliberately left open (see the quote on page 102). A simpler theory clearly has to make fewer assumptions, and thus it is less likely to conflict with ideas in the literature. In sum, the canonical version of MGs is preferable to a more faithful version for the purpose of mathematical inquiry. The formalism is easier to understand, proofs are simpler, and results are more likely to generalize to other variants.

2.4 The Chapter in Bullet Points

- MDTLs are regular tree languages and thus can be recognized by bottom-up tree automata, a device that moves through a tree from the leafs towards the root and assigns each node a state based on its label and the states of its daughters.
- The tree automaton recognizing a given MDTL uses its states to keep track of the computations of the feature calculus. Each state is a tuple of strings of unchecked features, where the first element of the tuple lists the unchecked features on the LI that is currently the head of tree, while the others are the strings of unchecked features on the LIs encountered so far that still need to move.
- ☠ If the states of the automaton are considered part of the labels, then an MDTL decorated this way can be generated by a context-free string grammar. This shows that the feature calculus is underlyingly context-free.
- MGs are weakly equivalent to MCFGs, which puts them in the class of mildly context-sensitive grammar formalisms. These are thought to provide a good approximation of natural language, although issues remain (overgeneration and maybe undergeneration).
- The mapping from derivation trees to phrase structure trees or multi-dominance trees can be defined in MSO.
- ☠ The MSO perspective of MGs makes it very easy to modify the feature calculus or the mapping from derivations to derived trees, e.g. in order to add head movement and affix hopping.
- Even though MGs are a simplified model of Minimalist syntax, they can be made more faithful without changing any of their defining formal properties.

It is mathematically more convenient to work with the impoverished rendition, but nothing hinges on it.

Part II

The Formal Landscape of Constraints

CHAPTER 3

Constraints on Trees

Contents

This chapter establishes the central result of the thesis, first proved in Graf (2011) and Kobele (2011) independently of each other: the close connection between Merge and a particular class of constraints.

Proposition 3.1 (Merge ≡ Constraints). A constraint can be enforced purely via Merge iff it is rational iff it is MSO-definable iff it defines a regular tree language iff it can be computed with a finitely bounded amount of working memory. □

These so-called rational constraints can be converted into selectional restrictions, which are subsequently enforced via Merge. Thus adding a constraint to an MG amounts to refining the lexicon so that it imposes more fine-grained subcategorization requirements. Since a refined Minimalist lexicon still defines an MG, constraints that can be lexicalized this way do not increase the generative capacity of the MG formalism. While localization strategies of this kind have been pursued before — GPSG's slash feature percolation comes to mind (Gazdar et al. 1985) — the lexicalizability of rational constraints is surprising because almost all syntactic constraints belong to this class. On the one hand, this makes it very easy to bring MGs closer to contemporary Minimalist syntax, supporting my claim in Sec. 2.3.1 that MGs are a faithful rendition of Minimalism. On the other hand it also raises the question what purpose constraints might serve for linguistic theorizing if they do not add anything over Merge.

A lot of formal legwork has to be done before we are in a position to discuss these and related issues in an insightful way. The conceptual backdrop is provided in Sec. 3.1. I start with a brief overview of previous work on constraints, in particular Müller and Sternefeld's (2000) classification of constraints. Their approach is then complemented by the logical view of constraints, also known as model-theoretic syntax (Blackburn and Meyer-Viol 1994; Rogers 1998; Pullum 2007). Synthesizing the two traditions, I propose a general taxonomy of constraints for MGs that classifies them according to the type of object they apply to (phrase structure tree, multi-

dominance tree, derivation tree) and the size of their locality domain. In particular, constraints operating on a single tree are distinguished from transderivational constraints, also known as economy conditions. Only the former are discussed here, with the latter relegated to the next chapter.

Sec. 3.2 then establishes the connection between rational constraints and Merge. Starting with the observation that adding a constraint C to an MG G is tantamount to intersecting G's MDTL with the set of derivation trees satisfying C, I show that the result of such an intersection step usually won't be an MDTL. Nonetheless an equivalent MDTL can be obtained by refining category and selector features, i.e. the features controlling the application of Merge. An intuitive description of this procedure is given in Sec. 3.2.2, while Section 3.2.3 contains a mathematically rigorous presentation of these ideas (which differs from the original approach in Graf 2011). The latter section is optional and recommended only to readers that wish to verify the validity of my proofs. Everybody else may immediately advance to Sec. 3.2.4, which gives an overview of all the formal consequences for MGs.

The linguistic ramifications are explored in Sec. 3.3 and 3.4. I explain why the type of tree a constraint is stated over is irrelevant in most cases and why locality restrictions do not limit the power of rational constraints. I also give a few examples for how MGs can be rendered more faithful via rational constraints.

This chapter draws heavily on concepts introduced in the first part of the thesis. Readers should already be familiar with MDTLs and how they can be described in terms of slices (Sec. 1.1.2 and 1.1.5, optionally 1.2.1 and 1.2.2), and it is advantageous to be aware of the connection between MDTLs, tree automata, and context-free string grammars (Sec. 2.1.1 and 2.1.3). Quite generally tree automata feature prominently, as do regular tree languages (cf. Sec. 2.1.1). For the unabashedly mathematical Sec. 3.2.3, familiarity with the closure properties of regular tree languages is also presupposed.

3.1 A Taxonomy of Constraints

3.1.1 The Role of Constraints in Linguistics

Since the early days of generative grammar constraints have been the primary tool for syntactic reasoning alongside operations. One of the earliest constraints is probably the A-over-A principle (Chomsky 1964), and only a few years later we already find an abundance of island conditions (Ross 1967) and the highly influential notion of Subjacency (Chomsky 1973), the spirit of which still underlies many locality principles put forward in the literature. Given the prominent role constraints have enjoyed for over five decades now, any attempt at a comprehensive overview would inevitably exceed the few pages alloted here. Still a few overarching themes can be readily identified.

Two conceptual questions about constraints are commonly explored in the literature: their relation to operations on the one hand, and their explanatory power on the other. The status of constraints with respect to operations is discussed under various guises. For example, many theories differ in how much of the explanatory burden they put on each component, so that comparisons of these proposals necessarily involve weighing the pros and cons of constraints *versus* operations. A recent example of this is the discussion between strictly derivational Minimalists (Epstein et al. 1998; Epstein and Seely 2002, 2006) and representational Minimalists (Brody 1995, 2000, 2002). Derivationalists seek to explain constraints in terms of operations, arguing that operations are a more fundamental, indispensable mechanism while constraints are tacked on, superfluous and do not fit naturally into the framework. Representationalists share this sentiment of conceptual parsimony, but turn the argument on its head. They claim that operations necessarily require access to the structure they assemble, which in turn implies that these structures are represented in some way. But if representations are indispensable, one might just as well make them the center of the theory and limit their shape via representational

constraints. Whatever one might think about these arguments, they highlight that there is some conceptual tension between constraints and operations.

Of course this sort of discussion extends beyond the boundaries of the Minimalist community. HPSG and LFG, for example, are heavily constraint-based, and advocates of these frameworks have argued extensively in favor of such a move. An even more egregious example is the switch from rule-based to constraint-based formalisms in phonology, brought about by OT. Phonology also furnishes one of the best known arguments (Kisseberth 1970) that a purely operation-based theory might be observationally adequate in the sense of Chomsky (1965) but nonetheless fail to capture important generalizations. In these cases the main concern isn't so much the relation between operations and constraints as which one of the two is better suited to express the right generalizations, i.e. the second issue identified above.

What all these examples have in common is that they base their conclusions on empirical case studies. Save for a few exceptions such as Pullum and Scholz (2001), the role of constraints is always evaluated with respect to a specific incarnation of a theory and a fixed set of phenomena. And rather than entire classes of constraints, it is individual constraints whose adequacy is put under scrutiny. That is a reasonable strategy if one's primary interest lies in the properties of these specific constraints — after all, it is their usefulness for empirical work that ultimately matters to linguists, and this usefulness has to obtain with the theory the constraint has been proposed for.

Unfortunately, the exceptional degree of specificity these studies depend on renders their results highly volatile. Theories are constantly in flux, and superficially minor changes may undermine the validity of earlier findings. For instance, there were many reasons that motivated the switch from the operational extended standard theory to GB with its pronounced reliance on principles, filters, and constraints, but few of them could be invoked verbatim to argue for a stronger role of constraints in Minimalism today. Besides theories, our understanding of empirical phenomena

also evolves, with new data calling earlier claims into doubt. There also seems to be no easy way of generalizing the findings of empirical case studies to new constraints, even if they closely resemble the ones that have been studied. Finally, two constraints might work well in conjunction but yield horrible results in isolation.

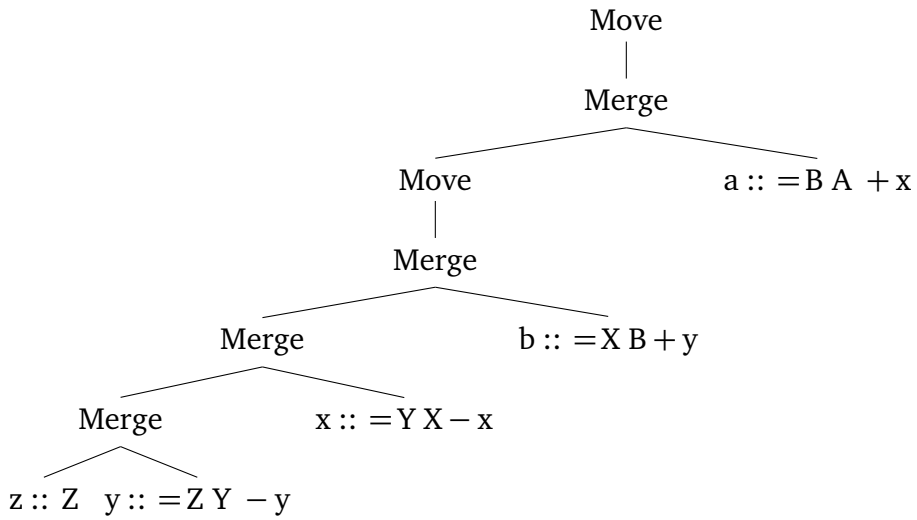
Example 3.1 Non-monotonic effects of the Specifier Island Constraint

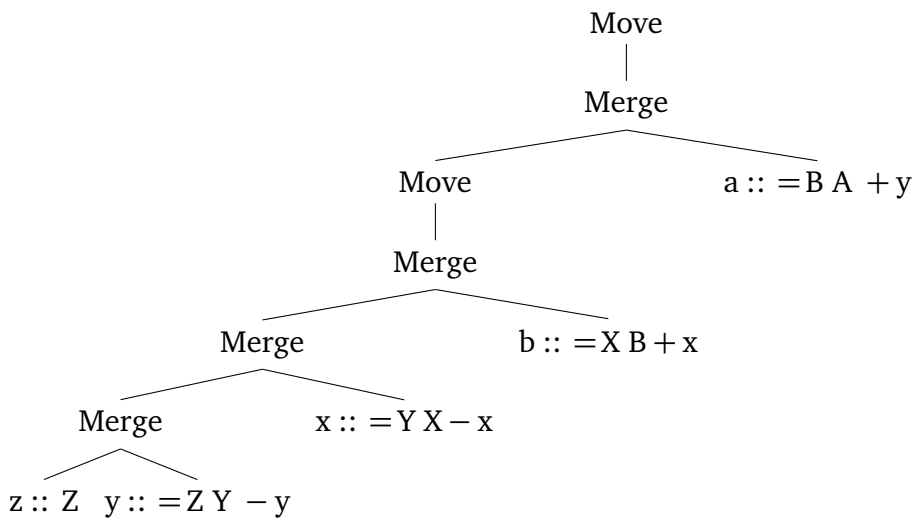
The SMC is essential in limiting the generative capacity of MGs to the class of MCFLs. Without the SMC, MGs are significantly more powerful, although their string languages are still context-sensitive and thus not completely unrestricted (Salvati 2011). Another constraint that has been studied in the MG literature is the Specifier Island Constraint (SPIC), which blocks extraction of a subconstituent from a specifier (Stabler 1999). In conjunction with the SMC, the SPIC lowers the generative capacity of MGs as one would expect (Michaelis 2004, 2009). What is surprising, though, is that the SPIC increases the power of MGs that lack the SMC (Gärtner and Michaelis 2005; Kobele and Michaelis 2005, 2011). As it turns out, MGs with the SPIC but without the SMC are Turing complete. This means that they can generate any recursively enumerable string language, putting them in the most powerful class of formal grammars. We see, then, that the effects of the SPIC are not monotonic and highly dependent on other parameters of the grammar. It might be a reasonable constraint in conjunction with the SMC, but it has disastrous effects without it.

The Turing completeness proof relies on the fact that any recursively enumerable string language can be computed by an automaton with a single queue as its memory, a so-called queue automaton. A queue is a particular kind of memory similar to the stack used by pushdown automata. Both stacks and queue provide an infinite storage for symbols, but the symbols are in a linear order representing the temporal

order in which they were put into memory. For instance, an automaton might start with an empty storage and put symbols A and B in storage every time it reads an a or b in the string. So for the string $caababd$, say, the string of symbols in the storage would be $AABAB$ at the point when d is encountered. What sets queues apart from stacks is how symbols might be read from memory. In a stack, only the rightmost symbol in memory can be read. If the automaton wanted to check if it has seen an A at some point, it would first have to remove the B from memory, making A the rightmost symbol. In a queue, on the other hand, only the leftmost symbol in memory is accessible. Hence the automaton could immediately verify that it has seen an a before, but would have to remove two As from memory to make the same check for b. Essentially, then, stacks are a "last one in is the first one out" model of memory, whereas queues are a "first one in is the first one out" model.

The lack of the SMC and the presence of the SPIC conspire in a way that makes it possible to simulate a queue automaton via an MG. Consider first a standard MG with both the SMC and the SPIC. Now take a look at the two derivations below, each of which involves two LIs in a head argument relation such that both undergo movement.





The derivation on the top is well-formed. First y moves to the specifier of b, taking along z. Then x moves by itself to the specifier of a. The second derivation, on the other hand, contains a violation of the SPIC. This time x moves first, turning it into a specifier of b. As an immediate consequence the SPIC no longer allows any extraction to take place from x. But y is contained by x and still needs to get its licensee feature −y checked. At this point the derivation has become unsalvageable.

This example shows that the SPIC limits the order in which movement steps may take place. If LI x enters the derivation before LI y, then x must move before y. The similarity to the "first one in is first one out" strategy of queues should be apparent. Merging an LI is the analogue of putting a symbol in the queue, and moving it is tantamount to removing the symbol. Now as long as the SMC holds, only a finite number of LIs can move at any given point, so the size of the queue is finitely bounded. However, once the SMC is dropped, this limit disappears, allowing MGs to simulate queue automata via Merge and Move.

In sum, most of the approaches in the literature focus on the evaluation of specific constraints with respect to contemporary theories and a small selection

of empirical phenomena. While this maximizes the immediate applicability of the results, it limits their scope, breadth and long-term viability. More abstract and far-reaching questions cannot be answered this way to a satisfying degree:

- What kinds of constraints are there?
- Are there principled distinctions between them regarding expressivity, learnability or psychological reality?
- Can constraints of one type be translated into another?
- How do constraints interact?
- What factors limit the power of constraints?
- Are there phenomena that can only be explained by constraints?
- What is the relation to operations?
- Why are there constraints in syntax at all?

Everyone who wishes to explore these topics must move away from specific constraints towards a perspective that emphasizes classes of constraints. The next section presents two important steps in this direction.

3.1.2 The Müller-Sternefeld Hierarchy

Linguists have accumulated a rich set of terms over the years to distinguish various types of constraints, but there is little consensus on what the relevant types are, how they should be defined, and if these distinctions should even matter.

The most elaborate attempt at systematizing these notions and put them into relation is undertaken in Müller and Sternefeld (2000). Müller and Sternefeld rely on two parameters in their classification of constraints. First, constraints may apply to phrase structure trees or to derivations. Second, the application domain of the

constraint may be a locally bounded subtree, the entire tree, or a collection of trees. Out of the six logically possible combinations, they identify five as correspondents of well-known constraints from the literature. Any constraint that applies to a single phrase structure tree is called *representational*, irrespective of whether it is locally bounded or not. A *derivational* constraint, on the other hand, is a locally bounded constraint on derivation trees, and its unbounded counterpart is called *global*. Constraints that apply to collections of trees are *translocal* or *transderivational*, depending on whether they apply to phrase structure trees or derivations.

While these definitions sound clearcut, deciding which class a specific constraint belongs to tends to be a muddled affair. Consider the PBC, which states that every trace must be properly governed at LF. During the era of GB, the PBC was thought of as a filter on phrase structure trees produced by Move α, which makes it a representational constraint. But a plausible reinterpretation of the PBC could also phrase it as a constraint on when movement may take place during the derivation, putting it in the class of derivational constraints. This kind of duality also holds for other constraints such as Subjacency and Relativized Minimality. But at least constraints that under a Minimalist conception apply at the interfaces seem to be good candidates for purely representational constraints. Kayne's (1994) LCA, for instance, states that no two leaves in the final phrase structure tree may c-command each other. While there might be ways of enforcing this derivationally, the relevant constraint would presumably look very different. So despite the strong overlap between derivational and representational constraints, there still seem to be a few distinguishing cases.

Global constraints are also difficult to pin down. Intuitively, a global constraint is a constraint on derivations that requires unbounded look-ahead or look-back. The standard example is GB's Projection Principle, which states that the Θ-criterion (Fillmore 1968) holds at all steps during the derivation. The Projection Principle thus ensures that at any given point in the derivation, every argument is assigned

exactly one Θ-role and every Θ-role is assigned to exactly one argument. But it is relatively easy to see that this requires neither look-ahead nor look-back. Instead, one only needs a derivational constraint that ensures that an argument receives a Θ-role when it enters the derivation, that every Θ-role is discharged at some point, and that no structure manipulating operation may affect the assignment of Θ-roles. But not only can global constraints usually be recast in derivational terms, they are also readily subsumed by transderivational constraints.

A transderivational constraint takes as input a set of derivations, ranks those derivations according to some economy metric, and then discards all but the highest ranked trees. A prominent example is the Shortest Derivation Principle (Chomsky 1995b), which out of a set of competing derivations picks the one that involves the fewest instances of Move. Rizzi's (1997) Avoid Structure is a close relative that militates against trees that contain more functional heads than are necessary for the derivation to converge. The Accord Maximization Principle of Schütze (1997) optimizes in the other direction by forcing as many morphological features as possible to enter into agreement relations. Since transderivational constraints have access to the entire derivation during the evaluation phase, global constraints are easily rephrased as transderivational ones. For example, the Merge-over-Move principle (Chomsky 1995b, 2000) enforces that Merge is always preferable to Move unless this causes the derivation to become ill-formed at some — possibly much later — point. In Müller and Sternefeld's (2000) classification, Merge-over-Move is a global constraint since it applies to a single derivation but requires unbounded look-ahead. In the syntactic literature, however, it is often recast in transderivational terms such that given a choice between two well-formed derivations, Merge-over-Move prefers the one that delays Move the longest. This kind of reinterpretation isn't too surprising, though, seeing how transderivational constraints are by definition an extension of global constraints.

Translocal constraints are transderivational constraints that apply to phrase

structure trees rather than derivations. They aren't nearly as common, presumably because they, too, can easily be stated as transderivational constraints instead. The best known example of a translocal constraint is the Avoid Pronoun principle (Chomsky 1981), which establishes a general dispreference for overt pronouns. Given two representations that differ only on whether a given pronoun is overt or covert, the Avoid Pronoun principle chooses the one with the covert pronoun. Once again it is easy to see how this principle could be phrased as a transderivational ban against the introduction of overt pronouns unless necessary at some later point, similar to the Merge-over-Move principle. As Merge-over-Move is also global, its similarity to Avoid Pronoun suggests that the latter might be global, too.

It seems, then, that the boundaries between the constraint classes are rather blurry. Representational and derivational constraints show strong overlap, global constraints can be viewed as derivational or transderivational, and translocal constraints as global or transderivational. This does not mean that the classification is misguided, though. For one thing, that specific constraints can be shifted between two or more classes does not mean that these classes are equivalent. Maybe the original formulation of the constraint just happens to employ a mechanism that is more powerful than necessary, so that the constraint can be pushed into a weaker class. Moreover, the kind of shifting seems to be restricted to more closely related classes. For instance, the Avoid Pronoun principle can easily be made global, but it is less obvious what a representational analogue would look like. On the other hand, every constraint can be made transderivational. There seems to be a power difference between the respective constraint classes that affects which constraints can be moved into which classes. Based on our observations so far, the interdependencies might be something along the following lines:

- representational $\cap$ derivational $\neq \emptyset$,
- representational $\neq$ derivational,

- derivational $<$ global $<$ transderivational,
- global $\cap$ translocal $\neq \emptyset$,
- translocal $<$ transderivational

This is remarkably close to the hierarchy proposed in Müller (2005), which I call the Müller-Sternefeld (MS) hierarchy: representational $=$ derivational $<$ global $<$ translocal $<$ transderivational. As I did in this section, Müller derives the hierarchy purely from careful examination of specific representative constraints. Therefore we would be wise to treat it merely as an intriguing conjecture for now, a conjecture that needs to be proved through more rigorous means.

3.1.3 Logic and Constraints

Even though constraints have been a cornerstone of linguistic theories since the sixties, they did not attract much attention among mathematical linguistics until thirty years later. While transformations and other operations could readily be reinterpreted as various computational devices such as formal grammars and automata, there was no such model for constraints. This changed in the early nineties when a series of papers pointed out the close relation between linguistic constraints and logical formulas (Blackburn et al. 1993; Blackburn and Meyer-Viol 1994; Backofen et al. 1995; Kracht 1995a,b; Rogers 1997, 1998). A constraint c is a statement that must be satisfied by a structure in order to be well-formed with respect to c. A logical formula ϕ is a statement that must be satisfied by a structure in order to be a model of ϕ.

Example 3.2 Constraints as logical formulas

Take a simple constraint that forbids any node labeled B to be (properly) dominated by a node labeled A. This constraint can be expressed more accurately as "for all

nodes x and y, if x dominates y and y has label B, then x does not carry label A". Now suppose that $x \triangleleft^+ y$ and $A(x)$ are true iff x dominates y and x is labeled A, respectively. Then our constraint can be translated into the first-order formula $\forall x, y\,[x \triangleleft^+ y \wedge B(x) \rightarrow \neg A(x)]$.

With just a small number of binary predicates such as proper dominance ($x \triangleleft^+ y$) and identity ($x \approx y$), unary predicates for labels (VP(x), John(x)), and the standard logical connectives *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\rightarrow$) and *iff* ($\leftrightarrow$) it is possible to define increasingly complicated predicates and relations to express very elaborate constraints in logical terms.

Example 3.3 A logical definition of c-command

Assume that the constraint above involves c-command rather than proper dominance. That is to say, no node labeled A may c-command a node labeled B. This change is trivial if there is a predicate c-com(x, y) that holds iff x c-commands y. In that case the new formula is $\forall x, y\,[\text{c-com}(x, y) \wedge B(x) \rightarrow \neg A(x)]$.

But for various reasons it is unappealing to enrich a logic with more and more primitive, unanalyzed predicates. The number of basic predicates should be small, and all additional predicates one might need must be defined in terms of those basic predicates. For c-command, this merely involves using proper dominance and equivalence to restate the standard definition in logical terms.

$$
\begin{array}{lr}
\text{c-com}(x,y) \iff & x \text{ c-commands } y \text{ iff} \\
\quad \neg(x \approx y) \wedge & x \text{ and } y \text{ are not the same node, and} \\
\quad \neg(x \lhd^{+} y) \wedge & x \text{ does not dominate } y, \text{ and} \\
\quad \forall z \Big[& \text{for every node } z \text{ it holds that} \\
\quad z \lhd^{+} x \rightarrow & z \text{ dominating } x \text{ implies that} \\
\quad z \lhd^{+} y \Big] & z \text{ also dominates } y
\end{array}
$$

A more common definition of c-command demands instead that the mother of x c-command y. This can also be stated in logical terms, of course, but we first have to define the *mother-of* predicate $\lhd$.

$$
\begin{array}{lr}
x \lhd y \iff & x \text{ is the mother of } y \text{ iff} \\
\quad x \lhd^{+} y \wedge & x \text{ dominates } y, \text{ and} \\
\quad \neg\exists z \Big[& \text{there is no } z \text{ such that} \\
\quad x \lhd^{+} z \wedge & x \text{ dominates } z \text{ and} \\
\quad z \lhd^{+} y \Big] & z \text{ dominates } y
\end{array}
$$

The new definition of c-command differs only minimally from the previous one.

$$
\begin{array}{lr}
\text{c-com}(x,y) \iff & x \text{ c-commands } y \text{ iff} \\
\quad \neg(x \approx y) \wedge & x \text{ and } y \text{ are not the same node, and} \\
\quad \neg(x \lhd^{+} y) \wedge & x \text{ does not dominate } y, \text{ and}
\end{array}
$$

$$\begin{array}{ll} \forall z \Big[& \text{for every node } z \text{ it holds that} \\ z \lhd x \rightarrow & \text{if } z \text{ is the mother of } x \\ z \lhd^{+} y \Big] & \text{then } z \text{ dominates } y \end{array}$$

By explicitly defining c-command through proper dominance, we ensure that the properties of a logic using both dominance and the c-command predicate do not differ from the corresponding logic without c-command. The predicate c-com(x, y) is merely a convenient shorthand for the formula on the right-hand side. So the formula c-com$(x, y) \wedge A(x)$ is actually an abbreviation of $(x \approx y) \wedge (x \lhd^{+} y) \wedge \forall z[z \lhd^{+} x \rightarrow z \lhd^{+} y] \wedge A(x)$.

There are many different logics, and they all differ with respect to their expressivity. For example, propositional logic is significantly weaker than first-order logic as it is incapable of quantifying over nodes. The power of first-order logic, in turn, depends on the kind of predicates being used. A logic with binary predicates is more powerful than one with monadic predicates only. Similarly, a variant of first-order logic in which every formula may only use the variables x and y is weaker than one that may use an arbitrary number of variables. Things get even more involved once one considers the plethora of modal logics that have been proposed over the years (cf. Blackburn et al. 2002). The obvious question, then, is which logic is best suited to expressing linguistic constraints.

Different proposals have been made (see Kepser 2008 and references therein), but arguably the most commonly adopted logic is monadic second-order logic (MSO). I briefly discussed MSO in Sec. 1.2.3, where it wasn't used for constraints but rather the definition of the MG mapping from derivation trees to derived trees. MSO is

a minimal extension of first-order logic that can also quantify over sets of nodes (see Sec. B.4.1 for the full definition). Hence it can express dependencies between individual nodes as well as conditions on specific domains.

Example 3.4 A logical definition of Principle A

Binding principles incorporate the notion of a binding domain that can be formalized in terms of MSO.

Principle A (simplified) Every anaphor must be c-commanded by some DP within its binding domain.

C-command was already defined in example 3.3, and the notion of anaphor can be formalized in various ways. A crude yet efficient way is to simply list all LIs that are anaphors. For English, this would be *himself*, *herself*, and *itself*.

$$\begin{aligned} \text{anaphor}(x) \iff & && x \text{ is an anaphor} \\ & \text{himself}(x) \vee && x \text{ is labeled } \textit{himself}, \text{ or} \\ & \text{herself}(x) \vee && x \text{ is labeled } \textit{herself}, \text{ or} \\ & \text{itself}(x) \vee && x \text{ is labeled } \textit{itself} \end{aligned}$$

Only the notion of binding domain still needs to be defined. The binding domain of α is the smallest category that contains α, a governor of α, and an abstract subject accessible to α. A completely faithful implementation of this definition is a laborious task (see Rogers 1998:Ch.12), so for the sake of exposition I use a simplified version here: the binding domain of α is the smallest TP containing α. Using MSO, this can be captured as follows.

First, the predicate $\text{TP}(X)$ holds of a set X of nodes iff it is a subtree whose root

is labeled TP.

$$
\begin{array}{ll}
\mathrm{TP}(X) \iff & \text{the set } X \text{ of nodes is a TP iff} \\
\quad \forall x, y \Big[(& \text{for all nodes } x \text{ and } y \\
\quad X(x) \wedge x \lhd y \rightarrow X(y) & X \text{ contains the daughters of every } x \in X \\
\quad \Big] \wedge & \text{and} \\
\quad \exists x \Big[& \text{there is a node } x \\
\quad X(x) \wedge \mathrm{TP}(x) \wedge & \text{that belongs to } X, \text{ is labeled TP, and} \\
\quad \forall y \big[& \text{for every node } y \text{ it holds that} \\
\quad X(y) \rightarrow & \text{if } y \text{ also belongs to } X \\
\quad x \approx y \vee x \lhd^{+} y \big] \Big] & \text{then it is } x \text{ or dominated by } x
\end{array}
$$

Then the predicate b-dom(X, x) holds iff the set X of nodes is the smallest TP containing node x. Finding the smallest set requires the notion of proper subset, which is also MSO-expressible.

$$
\begin{array}{ll}
X \subset Y \iff & X \text{ is a subset of } Y \text{ iff} \\
\quad \forall x \big[& \text{it holds for every } x \text{ that} \\
\quad X(x) \rightarrow Y(x) \big] \wedge & \text{if } X \text{ contains } x, \text{ so does } Y, \text{ and} \\
\quad \exists x \big[& \text{there is some } x \\
\quad Y(x) \wedge \neg X(x) \big] & \text{that is contained by } Y \text{ but not } X
\end{array}
$$

Now we can finally define binding domains.

$$
\begin{aligned}
\text{b-dom}(X,x) \iff & \quad X \text{ is the binding domain of } x \text{ iff} \\
\text{TP}(X) \wedge & \quad X \text{ is a TP, and} \\
X(x) \wedge & \quad X \text{ contains } x\text{, and} \\
\neg\exists Y \Big[& \quad \text{and there is no } Y \text{ such that} \\
\text{TP}(Y) \wedge & \quad Y \text{ is a TP, and} \\
Y(x) \wedge & \quad Y \text{ contains } x\text{, and} \\
Y \subset X \Big] & \quad Y \text{ is a subset of } X
\end{aligned}
$$

With this battery of predicates in place, Principle A only takes a single line.

$$\forall x \left[\text{anaphor}(x) \rightarrow \exists X \exists y \left[\text{b-dom}(X,x) \wedge \text{DP}(y) \wedge X(y) \wedge \text{c-com}(y,x)\right]\right]$$

Example 3.5 An MSO formula for recognizing strings of odd length 

Principle A can actually be stated as a first-order formula without set quantification (the reader is invited to come up with the corresponding formula). The same does not hold for the constraint $^*\textsc{even}$ which deems all strings of even length ill-formed. Let $\prec$ denote string precedence such that $x \prec y$ iff x is immediately to the left of y. Then the following MSO-formula defines $^*\textsc{even}$:

$$\exists O \left[\forall x \left[\Big(\big(\neg\exists y \left[y \prec x \vee x \prec y \right]\big) \rightarrow O(x) \Big) \wedge \forall y \Big[x \prec y \rightarrow \big(O(x) \leftrightarrow \neg O(y) \big) \Big] \right] \right]$$

This formula is much more complicated than the ones we have encountered so far.

Intuitively, it states that there is a set O such that the first node of the string belongs to O, the last node of the string belongs to O, and a node belongs to O iff the nodes neighboring it do not. In other words, O contains the first, third, fifth, ... node in the string, but none of the nodes in an even position. Hence the first and the last node of the string belong to O iff the string's length is odd. This correctly rules out all strings of even length.

MSO's ability to define dependencies between nodes as well as specific domains within which dependencies must be met grants them a great amount of power when it comes to stating syntactic constraints. Rogers (1998) is an impressive demonstration of that. Rogers defines a variant of MSO over tree structures similar to the one we have been using so far and then proceeds with a full implementation of standard GB, complemented with Rizzi's (1990) Relativized Minimality. This shows that MSO formulas over trees are indeed a viable model for syntactic constraints.

Only a few conditions come to mind that might be of interest to syntacticians yet cannot be stated via MSO. The most glaring limitation is that MSO-formulas cannot determine whether two subtrees are identical (unless there is a fixed upper bound on the size of these subtrees). So an analysis of ellipsis as deletion under structural identity might not be MSO-definable, depending on what ancillary assumptions are made. For example, if structural identity is actually encoded via multi-dominance so that the tree in question does not literally contain two identical subtrees, but just one that is attached to two positions, an MSO-treatment is still in the realm of possibility.

A bigger problem is posed by semantic and pragmatic constraints. Even though MSO is a logic, it cannot make explicit use of notions such as semantic entailment or satisfiability. Hence a constraint that, say, needs to check whether two clauses

are truth conditionally equivalent cannot be expressed in MSO unless there is an equivalent, purely structural generalization.

Example 3.6 Semantic constraints and MSO

Fox (2000) proposes that QR is banned in cases where it does not affect meaning. As this requires checking entailment patterns, it cannot be done in MSO. However, Fox later simplifies this condition such that a quantifier may QR over another one only if this could in principle create a new reading. So a universal may QR over an existential because there is at least one formula ϕ such that $\forall x \exists y \phi$ differs semantically from $\exists y \forall x \phi$. QR of a universal over a universal, on the other hand, never yields a new reading and thus is blocked. If only finitely many different types of quantifiers need to be distinguished (e.g. two for existential and universal), then the weakened condition can be expressed in simple structural terms and thus should be MSO-definable. This issue will be taken up again in Sec. 4.4.3.

Note that the question of MSO-definability isn't one of whether semantics plays a role in the constraint, but whether the semantic aspects can be described in structural terms. The licensing of NPIs in a negative polarity context, for instance, has a semantic flavor to it, yet it can be expressed as a distributional limitation to certain structural configurations such as in the scope of a downward entailing quantifier (which in turn can be described as combinations of specific LIs). The type system of Montagovian semantics, although closely related to meaning, can be treated like syntactic selection. So it isn't semantic constraints as such that pose a challenge for MSO, it is semantic constraints that necessarily involve the inspection of the denotation of formulas of possibly unbounded size (if the size were bounded, we could compile a list to void the need for interpretation). It is safe to say that syntactic constraints do not require this kind of power, nor does a significant number of constraints at the syntax-semantics interface. Overall, then, MSO provides a

reasonable description language for the kind of constraints syntacticians are likely to entertain.

When restricted to trees, MSO is also very appealing from a formal perspective thanks to its expressive equivalence to bottom-up tree automata, which we encountered in Sec. 2.1.1 (Doner 1970; Rabin 1969; Thatcher and Wright 1968). Every closed MSO-formula ϕ (i.e. a formula without free variables) can be converted into an equivalent bottom-up tree automaton A. That is to say, the set of trees satisfying ϕ is identical to the tree language recognized by A. This conversion also works in the other direction, establishing the equivalence of bottom-up tree automata and MSO over trees.

The connection between MSO and tree automata has two important implications. First, since tree automata use a finite set of states as their memory, computing MSO formulas takes only a finitely bounded amount of working memory. This means that any constraint for which there exists an equivalent MSO formula can be computed with a finitely bounded amount of working memory, too. At least from a cognitive perspective, then, MSO-definable constraints are very appealing. Second, MSO can only define regular tree languages. The string yield of a regular tree language is context-free. But as briefly mentioned in Sec. 2.1.2, not all natural languages are context-free, so there are some phenomena in natural language that cannot be described in terms of MSO-definable constraints.

The mismatch in the weak generative capacity of MSO and natural language syntax admittedly raises concerns about the suitability of MSO as a formal model of linguistic constraints. Then again, MGs offer the right kind of generative power but are also fully specified by their MTDLs, which are regular and thus MSO-definable. So if every linguistic constraint can be paraphrased as an MSO-constraint over Minimalist derivation trees, the expressive limitations of MSO do not matter after all. But is such a paraphrase possible? This brings us back to the issues discussed in the previous section, in particular how powerful derivational and global constraints

are in comparison to other types.

3.1.4 Formalizing the Research Problem

We are now in a position to synthesize the distinct threads that have been running through this thesis into one coherent view that will allow us explore the status of constraints in syntax in a rigorous fashion. First, I adopt the MS-hierarchy (Sec. 3.1.2) as a succinct taxonomy of the different types of constraints employed in the syntactic literature. For the sake of clarity, an explicit distinction is made between constraints over phrase structure trees and constraints over multi-dominance trees, and constraints that do not compare multiple structures are subsumed under the macro-class of *tree-local* constraints. I only investigate tree-local constraints in this chapter, with reference-set constraints relegated to Ch. 4.

	tree-local		**reference-set**
	local	*global*	
phrase structure tree	representational		translocal
multi-dominance tree	multirepresentational		
derivation tree	derivational	global	transderivational

Table 3.1: Refined parameterization of the Müller-Sternefeld hierarchy

In order to make constraints amenable to mathematical inquiry, I only consider constraints that can be expressed as MSO-formulas over the relevant type of tree structure (Sec. 3.1.3). All such tree-local constraints are called *rational*.

Definition 3.2. A constraint defining a language L of trees is *rational* iff L is the set of models of some MSO-formula without free variables.

Finally, MGs provide the formal model of Minimalist syntax. The original definition in Sec. 1.2.4 is expanded with the option of adding constraints over derivations as well as derived trees. I express this in terms of intersection with the tree languages defined by the constraints. The tree language defined by a constraint over

derivations is intersected with the MG's MDTL, that of a constraint over derived trees with the image of the MDTL under the mapping to derived trees (multi-dominance trees or bare phrase structure trees depending on the type of mapping).

Definition 3.3 (MGs with Rational Constraints). A *Minimalist Grammar with Rational Constraints* (MGRC) is a 5-tuple $G := \langle \Sigma, Feat, Lex, \mathscr{D}, \mathscr{R} \rangle$ such that

- Lex is a $(\Sigma, Feat)$-lexicon, and
- $\mathscr{D}$ and $\mathscr{R}$ are finite sets of regular tree languages.

The MDTL of G is the largest subset of $\text{FSL}(Lex) \cap \bigcap_{D \in \mathscr{D}} D$ that obeys **Final**, **Merge**, **Move**, and **SMC**. The tree language $L(G)$ generated by G is the image of its MDTL under the MSO transduction Φ_{gr} intersected with $\bigcap_{R \in \mathscr{R}} R$.

The primary question is how MGRCs fare with respect to standard MGs regarding weak and strong generative capacity, which allows for conclusions to be drawn with respect to the impact of constraints on syntax. It will also be interesting to see to which extent the power of MGRCs depends on the presence of specific types of constraints.

3.2 Tree-Local Constraints as Merge

3.2.1 Operations on Minimalist Derivation Tree Languages

As pointed out at the end of Sec. 3.1.3, rational constraints over phrase structure trees are arguably too weak to be empirically adequate. I suggested that this might not be a problem, though, if all such constraints can be translated into constraints over derivation trees. Restricting the shape of derivations should be sufficient because each MG is uniquely specified by its MDTL. Let us then ignore constraints over derived trees for now and focus solely on the question whether MGRCs with constraints over derivations are more powerful than standard MGs.

This can be rephrased in more technical terms as whether the class of MDTLs is closed under intersection with regular tree languages. That is to say, is the result of intersecting an MDTL with a regular tree language itself an MDTL? If so, this kind of intersection is but a convenient way of converting one MG into another and adds no power to the formalism. By extension, rational constraints would be merely a different way of specifying MGs. As the reader will see now, the class of MDTLs is not closed under intersection with regular tree languages, but it is closed under intersection with MDTLs. The second result follows from a principled connection between MDTLs and Minimalist lexicons, and in the next section we will see how this connection can be exploited to get around the lack of closure under intersection with regular tree languages.

Recall from Sec. 1.2 that the slice of an LI l is a partial derivation tree that consists of l and all the interior nodes associated to one of l's positive polarity features. Intuitively, the slice of LI l contains exactly the nodes in the derivation tree that correspond to the projections of l in the phrase structure tree (with l as the 0-th projection). The free slice language FSL(*Lex*) of lexicon *Lex* consists of all (possibly partial) derivation trees that can be obtained by combining the slices of *Lex*. The MDTL over *Lex*, in turn, is the largest subset of FSL(*Lex*) that satisfies the requirements of the feature calculus, which are expressed by four constraints on the shape of derivation trees: **Final**, **Merge**, **Move**, and **SMC**. As explained in Sec. 2.1.1, every MDTL is a regular tree language.

A proper subset S of an MDTL is not necessarily itself an MDTL. The maximality requirement of MDTLs entails that the property of being an MDTL is in general not preserved under removal of derivation trees. As a consequence, the intersection of an MDTL M with a tree language L is not guaranteed to be an MDTL, even if L is regular.

Theorem 3.4. The class of MDTLs is not closed under intersection with regular tree languages. □

Proof. We have to exhibit a case where the intersection of an MDTL with a regular tree language fails to be an MDTL. Consider the MG G given by the lexicon $Lex := \{\text{a} :: =\text{c c}, \varepsilon :: \text{c}\}$. This grammar generates the string language a^*, and its MDTL L_G consists of all derivation trees of the form $\underbrace{\text{Merge}(\cdots\text{Merge}(}_{n}\varepsilon :: \text{c}, \underbrace{\text{a} :: =\text{c c})\cdots,\text{a} :: =\text{c c})}_{n}$, $n \geq 0$. For every $n \geq 1$, let L_n be the singleton set containing G's derivation tree t_n for a^n. Note that t_n is built from the same slices as every other t_m, $m \geq 1$. As every singleton set is regular, so is L_n, yet $L_G \cap L_n = L_n$ is not an MDTL because it fails maximality. ■

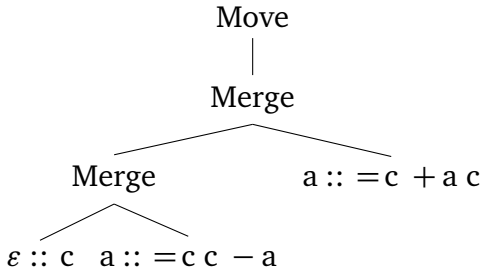
Although the proof above uses singleton sets as a particular example, the underlying problem immediately extends to regular tree languages in general. MDTLs, by virtue of their maximality, are continuous in a particular sense: if an MDTL contains derivation tree t, then it also contains every subtree of t that obeys all conditions of the feature calculus. Regular tree languages, on the other hand, are not bound by the feature calculus and may destroy this kind of continuity when intersected with an MDTL.

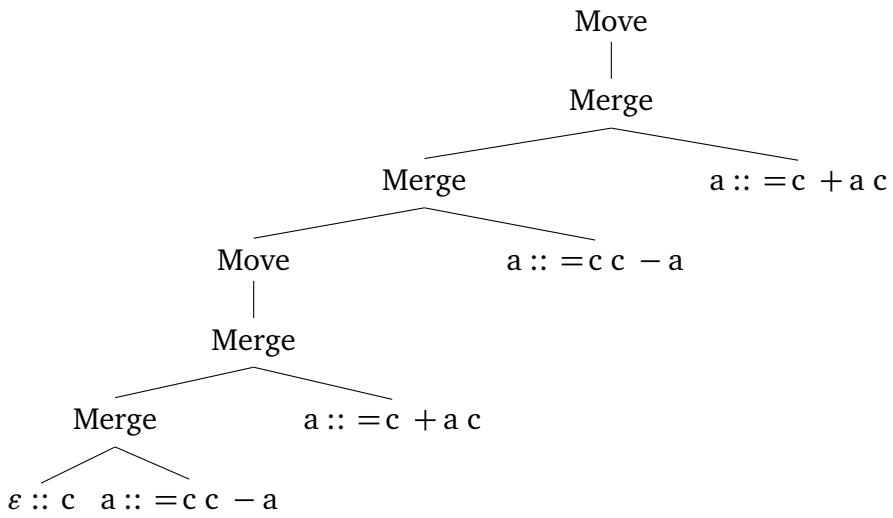
Example 3.7 Modulo counting destroys MDTLs

For a more interesting example of the destructive force of regular tree languages, consider a variant of MG G with movement:

$$\varepsilon :: \text{c} \qquad \text{a} :: =\text{c c} -\text{a}$$
$$\text{a} :: =\text{c c} \qquad \text{a} :: =\text{c} +\text{a c}$$

The corresponding MDTL contains the two derivations below, among others:





Now let L_o be the set of at most binary branching trees such that I) each node is labeled by Merge, Move, or one of G's LIs, and II) every tree has an odd number of nodes. We already saw in example 2.3 on page 57 that bottom-up tree automata can distinguish trees according to whether they contain an odd or an even number of nodes, so L_o is regular. Yet intersecting L_o with the MDTL of our example grammar G clearly does not yield an MDTL. Although the two derivation trees above are built from the same slices, only the larger one of the two is contained in the intersection as its number of nodes is odd. This provides us with yet another regular tree language that can destroy the maximality property of MDTLs under intersection.

Given our observations so far, one would expect that any intersection step that does not destroy the maximality of MDTLs produces an MDTL. In particular, the intersection of two MDTLs should also be an MDTL. This is indeed the case.

Theorem 3.5. The class of MDTLs is closed under intersection. □

The theorem is a corollary of a slightly stronger result that reduces the intersection of MDTLs to the intersection of Minimalist lexicons. So rather than intersecting the MDTLs of two MGs G and G', it suffices to take the intersection of their lexicons.

Since every subset of a Minimalist lexicon is also a Minimalist lexicon, the equality between MDTL intersection and lexicon intersection implies the intersection closure theorem above.

Lemma 3.6. Given MGs $G := \langle \Sigma, Feat, Lex \rangle$ and $G' := \langle \Sigma, Feat, Lex' \rangle$ with MDTLs L and L', respectively, $L \cap L'$ is the MDTL $L_\cap$ of MG $G_\cap := \langle \Sigma, Feat, Lex \cap Lex' \rangle$. □

Proof. Observe first that $\text{FSL}(Lex) \cap \text{FSL}(Lex') = \text{FSL}(Lex \cap Lex')$. Clearly both $\text{FSL}(Lex \cap Lex') \subseteq \text{FSL}(Lex)$ and $\text{FSL}(Lex \cap Lex') \subseteq \text{FSL}(Lex')$, so $\text{FSL}(Lex \cap Lex') \subseteq \text{FSL}(Lex) \cap \text{FSL}(Lex')$. In the other direction, $\text{FSL}(Lex) \cap \text{FSL}(Lex')$ contains only trees built from slices belonging to both Lex and Lex', so $\text{FSL}(Lex) \cap \text{FSL}(Lex') \subseteq \text{FSL}(Lex \cap Lex')$, establishing the equality of the two.

By the definition of MDTLs, L is the largest subset of $\text{FSL}(Lex)$ that satisfies the constraints **Final**, **Merge**, **Move**, and **SMC**. Overloading our notation, we identify each constraint with the set of trees that obey it, so that $L := \text{FSL}(Lex) \cap \textbf{Final} \cap \textbf{Merge} \cap \textbf{Move} \cap \textbf{SMC}$ (and analogously for L'). Hence

$$\begin{aligned} L \cap L' &= \text{FSL}(Lex) \cap \text{FSL}(Lex') \cap \textbf{Final} \cap \textbf{Merge} \cap \textbf{Move} \cap \textbf{SMC} \\ &= \text{FSL}(Lex \cap Lex') \cap \textbf{Final} \cap \textbf{Merge} \cap \textbf{Move} \cap \textbf{SMC} \\ &= L_\cap \end{aligned}$$ ■

It must be pointed out that the same kind of reduction does not work for union — in general, the MDTL of $G_\cup := \langle \Sigma, Feat, Lex \cup Lex' \rangle$ is a superset of $L \cup L'$. As a matter of fact $L \cup L'$ is not even an MDTL in many cases.

Example 3.8 MDTLs are not closed under union

Suppose that the lexicon of MG G contains a :: c and b :: =c c, that of G' d :: c and e :: =c c. These are just variants of the MG we encountered in the proof of Thm 3.4. The union of their respective MDTLs L and L' thus contains derivation trees of the

shape $\underbrace{\text{Merge}(\cdots\text{Merge}(}_{n} x :: \text{c}, \underbrace{y :: =\text{c c})\cdots, y :: =\text{c c})}_{n}$, $n \geq 0$, where either $x = a$ and $y = b$, or $x = d$ and $y = e$.

Now consider $G_\cup$, whose lexicon contains all four LIs. Its derivation trees have the same basic shape, except that $x \in \{a, b\}$ and $y \in \{d, e\}$. That is to say, a can now occur in the same derivation as e, and similarly for b and d. Upon reflection it should be clear that the MDTL of $G_\cup$ is the smallest MDTL that contains $L \cup L'$ (both are built with the same slices), wherefore the latter is not an MDTL.

The difference between lexical intersection and lexical union is that the latter allows for new combinations of slices, so that $\text{FSL}(Lex \cup Lex')$ will often be a proper superset of $\text{FSL}(Lex) \cup \text{FSL}(Lex')$, which is why the proof of Lem. 3.6 does not carry over to union.

Theorem 3.7. The class of MDTLs is not closed under union. □

Corollary 3.8. The class of MDTLs is not closed under complement. □

Proof. By De Morgan's law, intersection and union are interdefinable via complement: $L \cup L' = \overline{\overline{L} \cap \overline{L'}}$. If the class of MDTLs was closed under complement, closure under intersection would entail closure under union. ■

The results so far seem discouraging. The class of MDTLs is not closed under union or complement, let alone intersection with regular tree languages. While it is closed under intersection, this isn't very helpful for the purpose of adding constraints to MGs — intersection of MDTLs merely amounts to removing LIs from the lexicon, which clearly isn't enough to express the majority of constraints entertained in the literature. Still, the proof of intersection closure is illuminating in that it highlights the connection between MDTLs and Minimalist lexicons. The proof strategy does not readily extend to union or intersection of regular tree languages, but maybe

this is because the correspondence between MDTLs and lexicons is more abstract in these cases. What is needed at this point, then, is a more general lexicalization strategy that also works for the other operations.

3.2.2 Constraints as Category Refinement: The Basic Idea

Let us go back to some of the examples involving non-MDTLs. The proof of Thm. 3.4 involved intersection with a singleton set, and example 3.7 intersection with the regular language L_o of trees that contain an odd number of nodes. In each case the resulting set is not an MDTL. However, the astute reader might have already noticed that in both cases it is rather easy to construct an MG that generates the intended phrase structure language. As a matter of fact, even the derivation trees are almost the same, except for the names of some category and selector features.

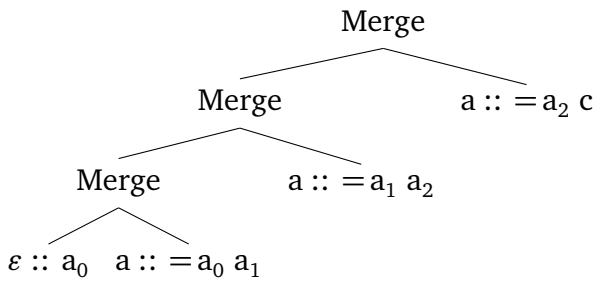
Example 3.9 An MG for a singleton language

The MG G presented in the proof of Thm. 3.4 is given by the lexicon $Lex := \{\mathrm{a} :: =\mathrm{c}\ \mathrm{c}, \varepsilon :: \mathrm{c}\}$. As before, L_n denotes the singleton set containing G's derivation tree t_n for a^n, where n is some fixed natural number. Intersecting G's MDTL with L_n yields a non-MDTL that contains only t_n.

Since every singleton string language is an MCFL, there is an MG that generates the string a^n (in fact, there are infinitely many). More importantly, though, there is an MG G_n whose derivation tree for a^n is structurally identical to t_n, except for some category features.

The lexicon of G_n contains $n+1$ LIs: $\varepsilon :: \mathrm{a}_0$, $\mathrm{a} :: =\mathrm{a}_0\ \mathrm{a}_1$, …, $\mathrm{a} :: =\mathrm{a}_{n-2}\ \mathrm{a}_{n-1}$, $\mathrm{a} :: =\mathrm{a}_{n-1}\ \mathrm{c}$.

The derivation tree for the case where $n = 3$ is given below. The reader is invited to verify that *modulo* subscripts, the tree is identical to G's derivation t_3.



Example 3.10 An MG for counting modulo 2

The strategy of refining categories also furnishes an MG for the non-MDTL in example 3.7. There we were given an MG G with four LIs:

$$\varepsilon :: \mathrm{c} \quad \mathrm{a} :: =\mathrm{c}\ \mathrm{c} \quad \mathrm{a} :: =\mathrm{c}\ \mathrm{c}\ -\mathrm{a} \quad \mathrm{a} :: =\mathrm{c}\ +\mathrm{a}\ \mathrm{c}$$

The regular tree language L_o contained all trees whose number of nodes is odd. Intersecting the MDTL of G with L_o did not produce an MDTL.

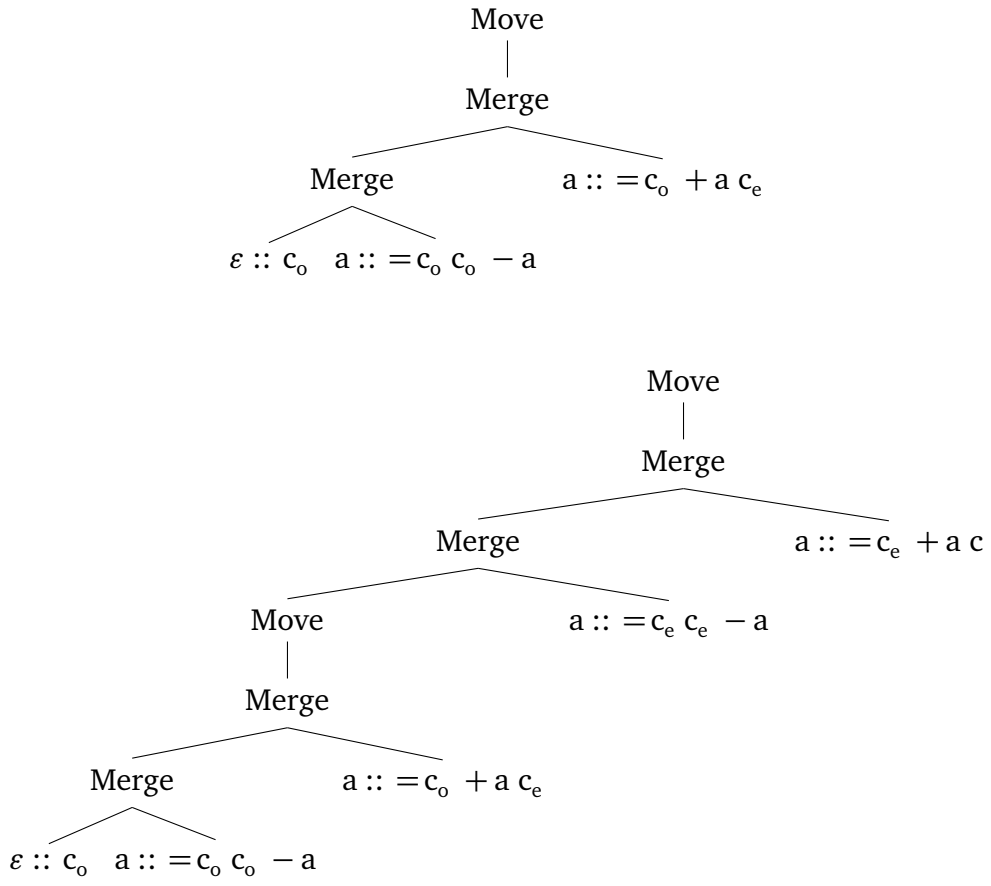
Once again there is a nearly identical MDTL which only differs in the category and selection features. This MDTL is generated by the following MG:

$$\begin{array}{llll} \varepsilon :: \mathrm{c_o} & \mathrm{a} :: =\mathrm{c_o}\ \mathrm{c_o} & \mathrm{a} :: =\mathrm{c_o}\ \mathrm{c_o}\ -\mathrm{a} & \mathrm{a} :: =\mathrm{c_o}\ +\mathrm{a}\ \mathrm{c_e} \\ & \mathrm{a} :: =\mathrm{c_e}\ \mathrm{c_e} & \mathrm{a} :: =\mathrm{c_e}\ \mathrm{c_e}\ -\mathrm{a} & \mathrm{a} :: =\mathrm{c_e}\ +\mathrm{a}\ \mathrm{c_o} \\ \varepsilon :: \mathrm{c} & \mathrm{a} :: =\mathrm{c_o}\ \mathrm{c} & & \mathrm{a} :: =\mathrm{c_e}\ +\mathrm{a}\ \mathrm{c} \end{array}$$

This grammar implicitly keeps track of the number of nodes via the category system. For instance, the original $\varepsilon :: \mathrm{c}$ corresponds to a single leaf node in the derivation tree, so its category is changed to c_o to indicate that the subtree headed by this LI contains an odd number of nodes. In the same vein, $\mathrm{a} :: =\mathrm{c}\ +\mathrm{a}\ \mathrm{c}$ is replaced by two LIs. If its complement is of category c_o, then the LI heads a subtree with an even

number of nodes: an odd number of nodes in the complement, the LI itself, a Merge node, and the Move node associated to the licensor feature +a. Consequently, the LI must have category feature c_e. Note that we still need LIs of category c in order for the derivations to be well-formed. To this end, each LI of category c_o is duplicated without the subscript (as listed in the third row).

A quick inspection of the refined versions of the two derivations in example 3.7 reveals that the derivations indeed differ only with respect to the presence of subscripts.



Category refinement establishes the necessary connection between MDTLs and

lexicons that makes it possible to incorporate constraints. The idea is very simple. Applying a constraint, i.e. intersecting an MDTL with another tree language, filters out certain derivations. As derivations are assembled from slices, this is tantamount to blocking certain combinations of slices. But MGs already provide Merge as a mechanism for determining which slices may combine. Merge is regulated by category features and selector features. The greater the number of distinct category and selector features, the more fine-grained distinctions can be made as to which slices may be joined by Merge. Hence, whenever a constraint blocks certain combinations of slices, this effect can be emulated in the lexicon by fine-tuning the set of Merge features.

There are two possible problems with the idea of using feature refinement in order to enforce constraints purely via Merge. First, Minimalist lexicons are finite by definition, so the number of new features introduced by the refinement must be finite. Second, how does one find a suitable refinement to mimic a given constraint? In the case of rational constraints, both issues are easily addressed thanks to their connection to bottom-up tree automata.

As mentioned before, rational constraints and bottom-up tree automata have the same generative capacity, so that one can be translated into the other. Bottom-up tree automata, in turn, have a finite set of states that they use to determine whether a tree is well-formed. Given an algorithm for incorporating the states of an automata directly into the category features, then, both issues disappear immediately: the finiteness of the lexicon is guaranteed by the finiteness of the state set, and the procedure for precompiling a rational constraint C directly into a given MG consists of converting C into an equivalent automaton A followed by the refinement algorithm for automata.

Before we move on, a slightly more abstract perspective is helpful in bringing out the connection between constraints and category refinement. Regular tree languages and the derivation tree languages of context-free string grammars are related in a

peculiar way that was first pointed out by Thatcher (1967). The class of derivation tree languages of CFGs is properly included in the class of regular tree languages. Yet Thatcher realized that the states of a tree automaton can be viewed as an alphabet, which the automaton "adds" to the original node labels. Therefore it is possible to refine any regular tree language R over alphabet Σ to a degree where it can be described as the derivation tree language of a context-free grammar over the alphabet $\Sigma \times Q$, where Q is the set of states of the automaton computing R.[1]

Example 3.11 Regular tree languages as projections of CFGs

The following CFG generates some basic English sentences, but lacks agreement between the subject and the finite verb.

$$\begin{array}{lcl} \text{S} & \rightarrow & \text{NP VP} \\ \text{NP} & \rightarrow & \text{John} \mid \text{children} \mid \text{NP and NP} \\ \text{VP} & \rightarrow & \text{laugh} \mid \text{laughs} \end{array}$$

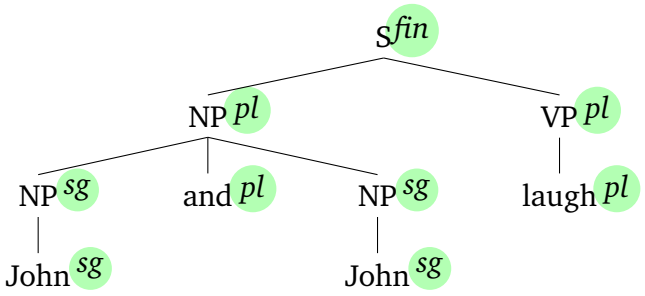
Subject verb agreement is easily enforced by a tree automaton with states *sg*, *pl*, *fin*, and these transition rules:

$$\begin{array}{lcl} \text{John} & \rightarrow & \mathit{sg} \\ \text{children} & \rightarrow & \mathit{pl} \\ \text{laugh} & \rightarrow & \mathit{pl} \\ \text{laughs} & \rightarrow & \mathit{sg} \\ \text{and} & \rightarrow & \mathit{pl} \end{array}$$

[1] Strictly speaking, Thatcher's (1967) result applies to *local sets* rather than CFG derivation tree languages. A tree language L is a local set iff there is some finite set S of trees of depth 1 such that L is the smallest set containing every tree t for which it holds that all subtrees of depth 1 are members of S. For example, the set of strictly binary branching trees with nodes labeled a or b is local — let $S := \{a(a,a), a(a,b), a(b,b), b(a,a), b(a,b), b(b,b)\}$. Every CFG derivation tree language is local, as each rule $A \rightarrow \alpha$ defines a tree $A(a_1, \ldots, a_n)$, where $\alpha = a_1 \cdots a_n$. However, local sets do not enforce a strict distinction between terminal and non-terminal symbols, and thus they constitute a slightly relaxed version of CFG derivation tree languages.

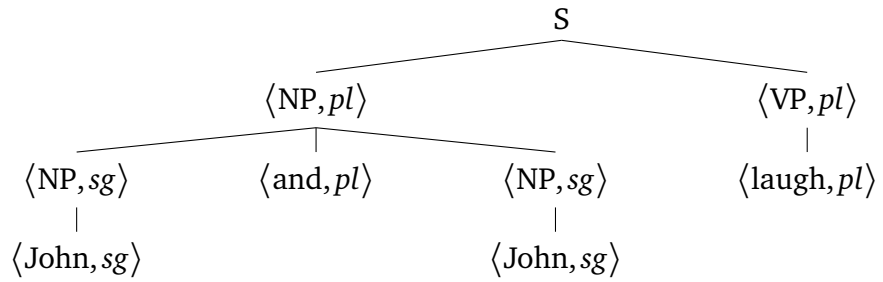
$$
\begin{array}{lcll}
\sigma(q) & \to & q & \text{for } \sigma \in \{\text{NP}, \text{VP}\} \text{ and } q \in \{sg, pl\} \\
\text{NP}(q, pl, q') & \to & pl & q, q' \in \{sg, pl\} \\
\text{S}(q, q) & \to & fin & q \in \{sg, pl\}
\end{array}
$$

A CFG derivation tree with state annotations is given below.



Incorporation of the states into the alphabet yields a CFG with subject verb agreement. Notice that the derivations still have the same shape.

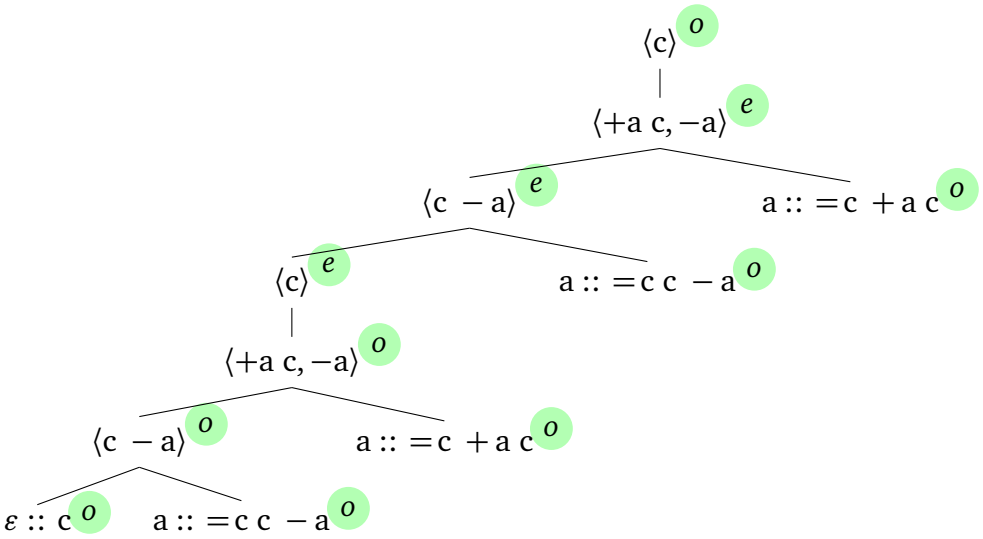
$$
\begin{array}{lcll}
\text{S} & \to & \langle \text{NP}, sg \rangle \ \langle \text{VP}, sg \rangle & \\
\text{S} & \to & \langle \text{NP}, pl \rangle \ \langle \text{VP}, pl \rangle & \\
\langle \text{NP}, sg \rangle & \to & \langle \text{John}, sg \rangle & \\
\langle \text{NP}, pl \rangle & \to & \langle \text{children}, pl \rangle & \\
\langle \text{NP}, pl \rangle & \to & \langle \text{NP}, q \rangle \ \langle \text{and}, pl \rangle \ \langle \text{NP}, q \rangle & q \in \{sg, pl\} \\
\langle \text{VP}, sg \rangle & \to & \langle \text{laughs}, sg \rangle & \\
\langle \text{VP}, pl \rangle & \to & \langle \text{laugh}, pl \rangle &
\end{array}
$$



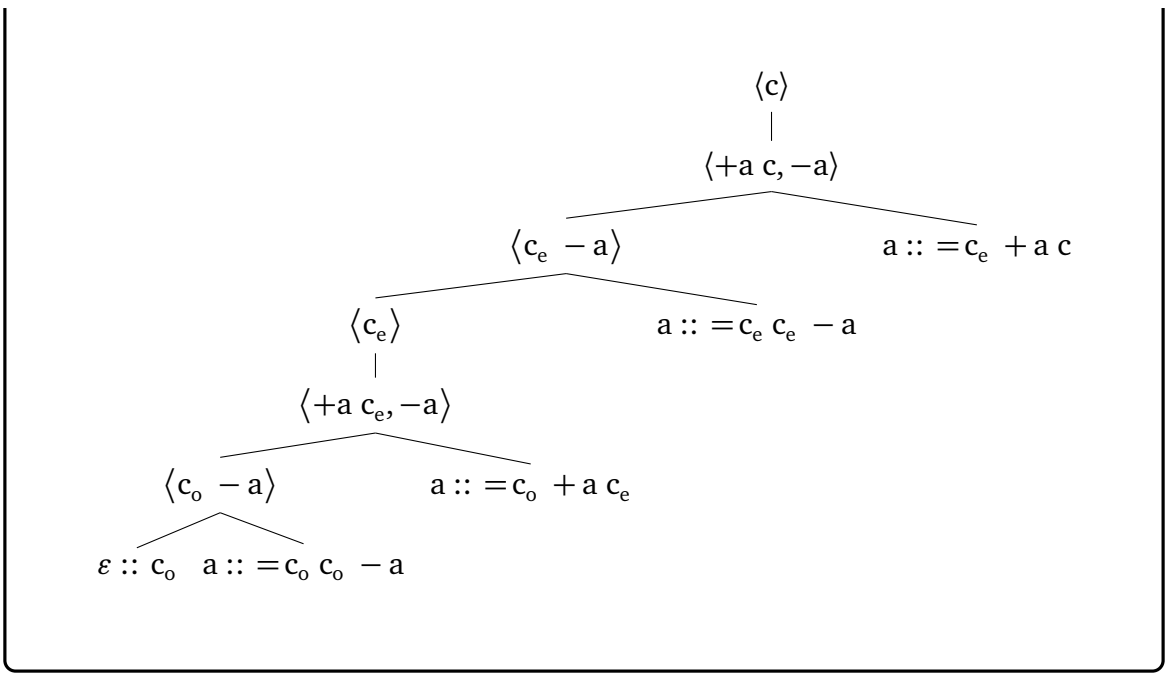
The connection between regular tree languages and CFGs extends to MGs, too. In Sec. 2.1.1 and 2.1.3 we saw that the MG feature calculus is context-free. That is to say, if we annotate the nodes of derivation trees with tuples whose components keep track of the strings of features that still need to be checked, then each MDTL can be described by a CFG. The gerrymandering of the category features indirectly refines the non-terminals of the "feature calculus CFG" in the spirit of Thatcher (1967).

Example 3.12 Modulo counting revisited

The longer one of the two derivations in example 3.7 is repeated here with interior nodes labeled by tuples. In addition, the states assigned by the automaton are also indicated.



Looking at the tuple-based version of the corresponding refined derivation, one can see how subscripting the category features with states of the automaton affects all interior node labels, parallel to the CFG in example 3.11. In contrast to Thatcher's strategy, however, interior node labels aren't combined with the state assigned to the same node. Instead, each LI l has its category feature subscripted with the state of its slice root. The selector features are also subscripted to match the new category features of the respective arguments.



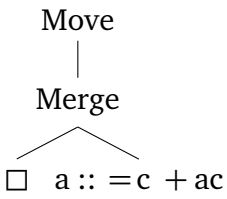
Thatcher's (1967) strategy of incorporating states into the alphabet makes it possible to express rational constraints via context-free means such as the MG feature calculus. However, the procedure as originally formulated cannot be applied to MDTLs. The Thatcher-algorithm would produce labels like $\langle \text{Merge}, q \rangle$. But the interior nodes of a derivation tree must be labeled by Merge or Move. This leaves only the option of pushing the state into the features of the LIs. Since every interior node in a derivation is associated to a specific positive polarity feature, this is doable in principle. But note that most of the states in the previous examples are actually redundant for the refinement procedure. As a matter of fact, it always suffices for the category feature of each LI l to be subscripted by the state assigned to its slice root.

The redundancy of all states except those belonging to slice roots follows from the fact that bottom-up tree automata can be made deterministic (cf. Comon et al. 2008:Sec. 1.1). Suppose we are given a bottom-up tree automaton A and a derivation t whose root is the slice root of LI l. The derivation is accepted by automaton A iff its root is assigned a final state. The root of t is the slice root of l, so in order to

encode t's grammaticality it suffices to subscript l with the state of its slice root. At least for the highest slice only the state at the slice root is important, then. As for all other slices, we may freely assume that A is deterministic because every bottom-up tree automaton can be determinized. That is to say, there is no ambiguity as to what state a node may be assigned — there is exactly one state it can receive given its label and the states of its daughters. Now if A is deterministic, then the state that A assigns to the slice root of an LI l' can be reliably predicted from the states assigned to the slice roots of the arguments of l'.

Example 3.13 Predicting the state of a slice root

Consider the slice for the LI a :: =c +a c from the MG in example 3.7.



The state of the Move node depends on the state of the Merge node, which in turn depends on the state of the LI and the root of whatever slice s is substituted for □. A deterministic automaton always assigns the same state to the LI, so that the state of the Move node can inferred from the root of the slice s. For the automaton presented in example 3.7, the state of Move is o if the root of s has state e, and e otherwise.

In sum, the state of the root of the derivation, which is the root of the highest slice, is the decisive one in determining its well-formedness. This state in turn, can be reliably predicted from the states assigned to the slice roots of the arguments, which in turn depend on the states assigned to the slice roots of each argument's arguments, and so on. Hence it suffices to subscript every LI's category feature by the state q assigned to its slice root, and its selector features by the corresponding

states that produce q. The complete procedure for lexicalizing rational constraints can be carried out in four steps:

- Given a rational constraint C, convert it into a deterministic tree automaton A that defines the same regular language.
- Decorate the derivation trees of MG G with the states A assigns to each node.
- Subscript the category feature of each LI l with the state of l's slice root.
- Add corresponding subscripts to all selector features so that the feature checking requirements of Merge are still satisfied for each derivation.

Note that this strategy depends purely on the symmetry of the feature checking mechanism underlying Merge. All other properties of MGs, in particular those pertaining to Move, are immaterial. Hence once could easily add new movement types, add the SPIC from example 3.1, or even drop the SMC — the lexicaliation procedure would still work as intended. We thus have a very strong result: rational constraints over derivations can be expressed purely via Merge, wherefore they can freely be added to the formalism without increasing its generative capacity.

3.2.3 Formal Specification of the Refinement Algorithm 

As just explained, the refinement algorithm only has to decorate selector and category features with the states assigned to the relevant slice roots. This still leaves open the question how these states should be determined.

One strategy, presented in Graf (2011), is to first add states to all nodes of all derivation trees using the original algorithm of Thatcher (1967). Lexical refinement is straight-forward then. This approach has the advantage that all ill-formed derivations can be disregarded, so that the refined lexicon contains no LIs that never occur in a well-formed derivation. However, specifying the algorithm in a

way so that it won't loop indefinitely while looking at more and more trees drawn from a potentially infinite MDTL is rather tricky. The regularity of both MDTLs and the language defined by the automaton guarantees that only a finite number of trees need to be inspected in order to produce the full lexicon of the refined MG, so an algorithm that smartly picks a representative sample of trees will terminate eventually. Graf (2011) makes no effort at defining such a smart algorithm, seeing how he is mostly concerned with proving certain closure properties rather than practical implementability. The intuitive appeal and mathematical simplicity of the derivation tree route thus comes at the cost of reduced practicality.

I opt for an alternative approach here which completely omits derivation trees. Instead, the algorithm operates directly on the lexicon by creating all possible refinements of LIs that are in line with the state calculus of the automaton. In most cases this will produce many redundant LIs that do not occur in any well-formed derivations. Fortunately, efficient CFG algorithms can be used to prune away these useless LIs (see Grune and Jacobs 2008:49-52).

The lexical refinement algorithm takes as input a Minimalist lexicon *Lex* and the *canonical* automaton $A := \langle \Sigma, Q, F, \Delta \rangle$ of some regular tree language R. An automaton for language L is canonical iff it is the smallest deterministic automaton recognizing L. For every LI $l \in Lex$ it constructs the slice of l, denoted slice(l), via the function ζ defined in Sec. 1.2.1 on page 33. It then simulates all possible runs of A over slice(l) by replacing each $\square_i$ in slice(l), $i \geq 0$, by some $q \in Q$ and applying the relevant transition rules in Δ to determine the state of the slice root. Each such slice is then translated back into a refined LI, producing the refined lexicon Lex_R. For every $l := \sigma :: \gamma \mathrm{c}_\mathrm{q} \delta \in Lex_R$ such that γ and δ are strings of features and $q \in F$ a final state of A, the LI $l' := \sigma :: \gamma \mathrm{c} \delta$ is also added to Lex_R. Note that l must be kept in addition to l' to allow for derivations where l is selected by another LI. For L and L_R the MDTLs of the MGs defined by *Lex* and Lex_R, respectively, L_R is identical to $L \cap R$ *modulo* state subscripts.

Algorithm 3.1: Lexical Refinement

Input : Minimalist lexicon $Lex \subseteq \Sigma \times Feat^*$,
deterministic bottom-up tree automaton $A := \langle \Sigma, Q, F, \Delta \rangle$
Output : Refined Minimalist lexicon $Lex_R \subseteq \Sigma \times (Feat^* \cup (Feat \times Q)^*)$

`/* remove ill-formed LIs */`

$Lex' := Lex \cap \{ \gamma f \delta \mid \gamma \in (\textsc{Base} \times Op \times \{+\})^*,$
$f \in \textsc{Base} \times \langle merge, - \rangle,$
$\delta \in (\textsc{Base} \times \langle move, - \rangle)^* \}$

`/* pool LIs that behave the same wrt refinement */`

max := maximum number of positive polarity features for every $l \in Lex'$
foreach $q_l \in Q$ *and* $\gamma \in (\textsc{Base} \times Op \times \{-\})^{\max}$ **do**
  $Lex[\gamma, q_l] := \{ l \in Lex' \mid l := \sigma :: \gamma \mathrm{f} \delta, \sigma \in \Sigma, \delta \in Feat^*,$
  $\mathrm{f} \in \textsc{Base} \times \langle merge, - \rangle,$
  γ a fixed string of positive polarity features,
  A assigns state q_l to l $\}$

`/* instantiate refined lexicon */`

$Lex_R := \emptyset$

`/* compute refined LIs */`

foreach $Lex[\gamma, q_l]$ **do**
  pick one arbitrary $l := \sigma :: \mathrm{f}_1 \cdots \mathrm{f}_n \in Lex[\gamma, q_l]$
  k := number of selector features of l
  foreach $\overrightarrow{q_k} := \langle q_1, \ldots, q_k \rangle \in Q^k$ **do**
    $l[\overrightarrow{q_k}] := \zeta(l, \overrightarrow{q_k})$ `/* see Alg. 3.2 */`
    compute state q that A assigns to root of $l[\overrightarrow{q_k}]$ according to Δ
    if q *exists* **then**
      foreach $l \in Lex[\gamma, q_l]$ **do**
        add refined LI $\rho(l, \overrightarrow{q_k} \cdot \langle q \rangle)$ to Lex_R `/* see Alg. 3.3 */`

`/* create LIs with final category C */`

foreach $l := \sigma :: \mathrm{f}_1 \cdots \mathrm{f}_i \langle \mathrm{C}, \mathrm{q} \rangle \cdots \mathrm{f}_n \in Lex_R$ **do**
  if $q \in F$ **then**
    add $\sigma :: \mathrm{f}_1 \cdots \mathrm{f}_i \, \mathrm{C} \cdots \mathrm{f}_n$ to Lex_R

return Lex_R

Algorithm 3.2: Slice function ζ for computing state-annotated slices

Input : Minimalist LI $l := \sigma :: \mathrm{f}_1 \cdots \mathrm{f}_n$ and vector $v := \langle v_1, \ldots, v_k \rangle$
Output : Slice of l with states instead of ports

```
/* Find number j of selector features and index i of last
   positive polarity feature                                  */
```
1 $i := 1$
2 $j := 0$
3 **while** $1 \leq i \leq n$ **do**
4 **if** $\mathrm{f}_i \in \textsc{Base} \times \langle merge, - \rangle$ **then**
5 $i := i - 1$
6 break
7 **else if** $\mathrm{f}_i \in \textsc{Base} \times \langle merge, + \rangle$ **then**
8 $i := i + 1$
9 $j := j + 1$
10 **else**
11 $i := i + 1$

```
/* sanity check length of vector                              */
```
12 **if** $j \neq k$ **then** raise exception

```
/* Construct slices top-down, iterating from f_i to f_1       */
```
13 **function** $\zeta(l,\ v,\ i,\ j)$ **def**
14 **if** $\mathrm{f}_i = \langle f, merge, + \rangle$ **then**
15 $t := \mathrm{Merge}(v_j, \zeta(l, v, i-1, j-1))$
16 **else if** $\mathrm{f}_i = \langle f, move, + \rangle$ **then**
17 $t := \mathrm{Move}(\zeta(l, v, i-1, j))$
18 **else if** $i = 0$ **then**
19 $t := l$

20 **return** $\zeta(l,\ v,\ i,\ k)$

Algorithm 3.3: Pairing function ρ for states and features

Input : Minimalist LI $l := \sigma :: \mathrm{f}_1 \cdots \mathrm{f}_n$ and vector $v := \langle v_1, \ldots, v_k \rangle$
Output : Refined LI l_r

`/* sanity check length of vector */`
if $k \neq 1 +$ *number of selector features in* l **then** raise exception

`/* pair selector features with states */`
function $\mathrm{zip}(\mathrm{f}_i \cdots \mathrm{f}_n,\ j)$ **def**
if $\mathrm{f}_i \in \textsc{Base} \times \langle merge, + \rangle$ **then**
  $\langle \mathrm{f}_i, v_j \rangle \cdot \mathrm{zip}(\mathrm{f}_{i+1} \cdots \mathrm{f}_n,\ j+1)$
else if $\mathrm{f}_i \in \textsc{Base} \times \langle merge, - \rangle$ **then**
  $\langle \mathrm{f}_i, v_j \rangle$
else
  $\langle \mathrm{f}_i \rangle \cdot \mathrm{zip}(\mathrm{f}_{i+1}, j)$

return $\sigma :: \mathrm{zip}(\mathrm{f}_1 \cdots \mathrm{f}_n, 1)$

Algorithm 3.1 gives a pseudo-code implementation of the procedure. Note that I opted for clarity over efficiency, leaving lots of room for optimization. In particular, each slice is processed twice: first during its creation, then in order to compute the state of its root. The same holds for every LI: it is accessed once during the construction of its slice, and a second time during the feature refinement. All these steps can be consolidated into one such that each LI is refined at the same time as its state-annotated slice is constructed. The assignment of states could also be improved, seeing how it relies on checking all logically possible combinations, many of which might not allow for any state to be assigned to the slice root. The set of state combinations can be restricted top-down — the slice root must receive some state — and bottom-up via the state of the LI.

Note that even though the number of state combinations explodes quickly and is the biggest factor in determining run-time behavior — for an LI with k selector features, there are $|Q|^k$ — the algorithm still runs in linear time of the size of the MG lexicon. Let *max* be the maximum number of positive polarity features per LI. Then every $l \in Lex$ spawns at most $|Q|^{max-1}$ state annotated slices (as the automaton is

deterministic, exactly one state is a valid subscript for the category feature). Each such slice is processed only a fixed number of times, and the maximum size of slices is linearly bounded by *max*. In practice, the size of the lexicon is negligible as long as the algorithm can pool most LIs into a small number of equivalence classes $Lex[\gamma, q_l]$ that each contain all LIs whose slices are isomorphic and receive state q_l at the lexical leaf node. Hence the algorithm does not depend on the size of the lexicon itself but rather on the degree of distinctness between individual LIs. Of course the latter is bounded by the former, so knowing the lexicon size suffices for evaluating the worst case scenario.

The size of the refined lexicon Lex_R is also linearly bounded by the size of *Lex*.

Proposition 3.9. Let *Lex* be an MG lexicon and $A := \langle \Sigma, Q, F, \Delta \rangle$ the canonical acceptor of some regular tree language. Then the size of the refined lexicon Lex_R is less than

$$2 \cdot \sum_{i=0} \left(|Lex^{(i)}| \cdot |Q|^i \right)$$

where $Lex^{(n)} := \{l \in Lex \mid l \text{ has exactly } n \text{ selector features}\}$. □

Keep in mind that $|Lex^{(i)}| = 0$ for all $i > max$, so no explicit upper bound needs to be put on the sum operator.

The factor 2 in the formula is due to MGs considering only C a final category. As a consequence, each LI with a category feature $\langle \mathrm{C}, q \rangle$, $q \in F$, has a duplicate in which $\langle \mathrm{C}, q \rangle$ is replaced by C. Graf (2011) proposes a relaxed form of MGs in which each grammar comes with an explicitly defined set F of final categories. This requires only a minimal change in the definition of **Final** provided in Sec. 1.2.2 and does not affect any of the formal properties of MGs and MDTLs. When the set of final categories is no longer limited to C and A assigns a state to the root of every state annotated slice, the size of the refined lexicon Lex_R generated by the algorithm is exactly $\sum_{i=0} \left(|Lex^{(i)}| \cdot |Q|^i \right)$.

In most cases Lex_R contains many LIs that cannot occur in a well-formed deriva-

tion by virtue of their feature make-up. While these superfluous LIs do no harm, it would be preferable if there was an algorithm that automatically removed all useless LIs from Lex_R. Fortunately various algorithms for context-free string grammars can be co-opted toward this end. In Sec. 2.1.3 I sketched a way of translating an MG G_m into a context-free string grammar G_c whose derivation tree language is identical to the MDTL of G_m, except that interior nodes are decorated with tuples of feature strings. As a matter of fact, this translation was just a special case of Thatcher's (1967) procedure for converting regular tree languages into CFGs where the interior nodes of the derivation trees correspond exactly to the states of the feature automaton presented in example 2.4 on page 59. Now if an LI never occurs in a derivation of G_m, then it is an unreachable terminal in G_c. Trimming unreachable terminals from a CFG is a common problem with well-known, efficient solutions (see Grune and Jacobs 2008:49-52). Given such a trimmed CFG G_c^r, it is easily converted back into an MG G_m^r; the lexicon is the smallest set containing every LI $\sigma :: \mathrm{f}_1 \cdots \mathrm{f}_n$ such that $\langle \mathrm{f}_1 \cdots \mathrm{f}_n \rangle \to \sigma$ is a terminal production of G_c^r. Hence, in order to remove all superfluous LIs from an MG, it suffices to convert it into a CFG, prune all unreachable terminals, and translate the resulting CFG back into an MG.

We can conclude so far that the algorithm for refining MGs via their slices rather than their MDTL can be stated in a few lines of pseudo-code, runs in linear time of the size of the lexicon, and yields a refined lexicon whose size is also linearly bounded by the original lexicon. The refined lexicon may contain unusable LIs, but these can be discarded automatically. The most important question, though, has not been addressed yet: is the algorithm correct? That is to say, given an automaton A recognizing a regular tree language R and MGs G and G_R with MDTLs L and L_R, respectively, are $L \cap R$ and L_R identical *modulo* the names of category and selector features?

The notion of projection is helpful in making this question slightly more precise. A *projection* is a surjective map $\pi : \Sigma \to \Omega$ between two (possibly ranked)

alphabets. The extension $\hat{\pi}$ of π to trees is given in the natural way such that for $t := \sigma(t_1, \ldots, t_n)$ a Σ-labeled tree, $\hat{\pi}(t) := \pi(\sigma)(\hat{\pi}(t_1), \ldots, \hat{\pi}(t_1))$. We overload our notation such that π may also denote $\hat{\pi}$, and we say that t' is a projection of t iff there is some projection π such that $t' := \pi(t)$. The terminology carries over from trees to languages in the familiar fashion: L is a projection of L' iff $L := \bigcup_{t \in L'} \pi(t)$. We can now paraphrase the correctness question as follows: let $A := \langle \Sigma, Q, F, \Delta \rangle$ be the canonical automaton recognizing some regular tree language R, G an MG with lexicon Lex and MDTL L, and G_R an MG with lexicon Lex_R and MDTL L_R. Is $L \cap R$ a projection of L_R?

It isn't too difficult to prove that L is a projection of L_R under the extension of the map π that sends every feature $\langle f, o, p, q \rangle \in \textsc{Base} \times Op \times \textsc{Polarity} \times Q$ to $\langle f, o, p \rangle$.

Theorem 3.10. $\pi(L_R) = L \cap R$. □

Proof. It suffices to show that $t \in L_R$ implies $\pi(t) \in L \cap R$, and that for every $t \in L \cap R$ there is some $t_R \in L_R$ such that $t = \pi(t_R)$. We prove by induction on the depth of t.

If $t \in L_R$ contains exactly one node, it is an LI $\sigma :: \mathrm{f}$ consisting of a single category feature. In this case $\mathrm{f} := \langle \mathrm{f_i}, \mathrm{q} \rangle$ iff the canonical acceptor A assigns q to $\pi(\sigma :: \langle \mathrm{f_i}, \mathrm{q} \rangle)$ iff A assigns q to $\sigma :: \mathrm{f_i}$. Note that since A is deterministic, q is unique. Therefore

$$\begin{aligned} \sigma :: \langle \mathrm{f_i}, \mathrm{q} \rangle \in L_R \quad & \text{iff } \mathrm{f_i} = \mathrm{C},\ q \in F, \text{ and } \sigma :: \langle \mathrm{f_i}, \mathrm{q} \rangle \in Lex_R \\ & \text{iff } \sigma :: \mathrm{C} \in Lex \text{ and } A \text{ assigns it state } q \in F \\ & \text{iff } \mathrm{f_i} = \mathrm{C},\ \sigma :: \mathrm{f_i} \in Lex \text{ and } \sigma :: \mathrm{f_i} \in R \\ & \text{iff } \sigma :: \mathrm{f_i} \in L \cap R. \end{aligned}$$

If $t \in L_R$ contains more than one node, then its root is the slice root of some LI $l := \sigma :: = \langle \mathrm{f_1}, \mathrm{q_1} \rangle \cdots = \langle \mathrm{f_k}, \mathrm{q_k} \rangle \ \langle \mathrm{f_{k+1}}, \mathrm{q} \rangle \cdots \mathrm{f_n}$, $k \geq 1$. For all $1 \leq i \leq k$, the i-th argument a_i of l has category feature $\langle \mathrm{f_i}, \mathrm{q_i} \rangle$. By our induction hypothesis, then, A assigns q_i to the slice root of a_i in $\pi(t)$. As A is deterministic, it also assigns a unique state to the slice root of $\pi(l)$, which must be q. Since $t \in L_R$ only if $\mathrm{f_{k+1}} = \mathrm{C}$ and

$q \in F$, it follows that $t \in L_R$ implies $\pi(t) \in L \cap R$. A simple variation of this argument shows that for every $t \in L \cap R$ there is a $t' \in L_R$ such that $t = \pi(t')$. ■

This shows that all rational constraints can indeed be lexicalized and thus do not increase the power of MGs.

3.2.4 The Power of Lexical Refinement

The lexical refinement strategy sketched and formally defined in Sec. 3.2.2 and 3.2.3, respectively, opens up many new ways of manipulating MGs through their derivation languages without increasing their strong generative capacity. Whatever additional restrictions one would like to put on Minimalist derivation trees, as long as they are rational, they can be expressed in a local fashion via Merge.

Actually, the constraints can even be stated over the derived trees rather than the derivations. This is so because every rational constraint over derived trees can be automatically translated into a rational constraint over derivation trees. In Sec. 1.2.3 I explained that Minimalist derivations can be mapped to multi-dominance trees via an MSO-definable transduction Φ_{gr} — or alternatively to phrase structure trees via Φ_{tr}. The direction of MSO transductions can easily be reversed, so the reversal of Φ_{gr}, denoted Φ_{gr}^{-1}, translates derived trees into derivation trees. These reversed MSO transductions have the peculiar property that they preserve regularity. That is to say, if L is a regular tree language, then so is its image under Φ_{gr}^{-1}. Now if C is some rational constraint defining some language L of derived trees, then $\Phi_{gr}^{-1}(L)$ is the language of derivation trees corresponding to these derived trees and is guaranteed to be regular. In the terminology of Müller and Sternefeld, locally unbounded representational constraints reduce to locally unbounded constraints over derivations, i.e. global constraints.

Proposition 3.11 ((multi)representational ≤ global). For every rational constraint over derived trees there exists an equivalent rational constraint over derivation

trees. □

This answers one of the main questions raised in Sec. 3.1: Are MGRCs, that is MGs with rational constraints over derivations and/or representations, more powerful than MGs? The answer is no, because both kinds of constraint can be lexicalized.

Theorem 3.12 (MG ≡ MGRC). For every MG with rational constraints there exists a strongly equivalent standard MG (and the other way round). □

Note that this result only holds for rational constraints — non-rational constraints can be lexicalized only if they could also be expressed as rational constraints.

Theorem 3.13. Let M be an MDTL and L a tree language defined by some constraint C. Then C can be lexicalized only if there is some regular tree language R such that $M \cap L = M \cap R$. □

Proof. Suppose that C can be lexicalized, and that the tree language L defined by C is not regular (for otherwise R could be chosen to be L). Since C can be lexicalized, $M \cap L$ is equivalent to some MDTL M_R *modulo* lexical refinement. As MDTLs are regular and regularity is preserved under the state-removing projection π (see Sec. 3.2.3), undoing the lexical refinement of M_R yields a regular subset R of M. Clearly $M \cap R = R = \pi(M_R) = M \cap L$. ■

Not only then can Merge express rational constraints, it can only express constraints that are rational with respect to the MDTL they apply to.

Proposition 3.14 (Merge ≡ Rational Constraints). A constraint can be expressed via Merge iff it is rational. □

Even though Merge and rational constraints — and by extension MGs and MGRCs — are expressively equivalent, the latter are definitely easier to work with.

Rather than writing a grammar completely by hand, one might now start with a bare bones MG that is subsequently refined via easily defined rational constraints. Applications of this strategy will be explored later in Sec. 3.4.

Curiously, rational constraints aren't the only way to manipulate MDTLs while staying inside the realm of MG-definability. If one ignores the subscripts introduced by lexicalization, MDTL-status is preserved under a variety of set-theoretic operations. In these cases, we say that MDTLs are *projection-closed* (p-closed) under these operations: MDTLs are p-closed under operation O iff it holds for every MDTL L that applying O to it yields some set that can be turned into an MDTL via lexical refinement (readers wondering about the meaning of "projection" here are referred to the end of Sec. 3.2.3).

Theorem 3.15. The class of MDTLs is p-closed under intersection with regular tree languages. □

Corollary 3.16. The class of MDTLs is p-closed under intersection, union, relative complement, and symmetric difference. □

Proof. By Thm 3.5 MDTLs are closed under intersection, which immediately entails their p-closure under intersection. Independently of this fact, though, p-closure also follows from Thm. 3.15 due to every MDTL being regular.

Given two MDTLs L and M, their union is given by $L \cup M := S \cap (L \cup M)$, where S is the MDTL built from the union of the lexicons for L and M; note that $L \cup M$ is regular. Their relative complement $R := L \setminus M$ is also regular language, so $L \setminus M = L \cap (L \setminus M) = L \cap R$ is a projection of some MDTL by Thm. 3.15. This furthermore implies closure under symmetric difference as $L \triangle M := (L \cup M) \setminus (L \cap M)$. ■

Note that p-closure under relative complement does not imply p-closure under complement with respect to the class of all MDTLs. The complement of some set S is its relative complement with respect to its largest superset S'. For example, the

complement of a language L of strings over alphabet Σ is $\Sigma^* \setminus L$. Since Σ^* is the set of all strings over Σ, $\Sigma^* \setminus L$ is the set of all strings that are not contained by L, which is indeed the complement of L. Complement is not a well-defined notion for the class of all MDTLs because there is no largest MDTL with respect to which it could be defined.

However, if we restrict our attention to the class of all MG whose lexicon is a subset of some finite set *Lex* over Σ, *Feat*, there will be one MDTL that subsumes all others and complement can be expressed in terms of relative complement as usual.

Corollary 3.17. Let *Lex* be some finite subset of $\Sigma^* \times Feat^*$, and MG_{Lex} the class of MGs whose lexicon is a subset of *Lex*. Then the class of MDTLs of grammars $G \in \text{MG}_{Lex}$ is p-closed under complement. □

The general upshot of all these p-closure properties is that we need not limit ourselves to just intersection; union and relative complement are viable alternatives in the specification of grammars. This opens up an entirely new box of tools that linguists haven't made use of so far. It will be interesting to see if they can foster new generalizations.

3.3 The Relative Power of Constraint Classes

3.3.1 A Revised Müller-Sternefeld Hierarchy

The p-closure properties of MDTLs have important repercussions for the extended MS-hierarchy discussed in Sec. 3.1.2. Recall that Müller and Sternefeld (2000) distinguish five classes of constraints, based on two parameters: the type of object it applies to (representations or derivations), and its locality degree (local, global, transductive). I proposed a slight refinement in order to address whether derivations are mapped to standard phrase structure trees or multi-dominance trees (see Fig. 3.1 on page 151). The MS-hierarchy makes two important claims about tree-local

constraints: locality matters for constraints over derivation trees, and local constraints over derivations are exactly as powerful as constraints over representations (whose locality degree is irrelevant). Neither claim is entirely correct when rational constraints are considered.

Take the issue of locality. The application domain of rational constraints is unbounded because they coincide with MSO formulas, whose validity is evaluated with respect to the entire tree. But rational constraints are also strictly local because every MSO formula can be converted into a bottom-up tree automaton, which assigns states in a maximally local fashion — the state of a node depends only on its label and the states of its daughters. The locality of the tree automaton is preserved even when it is precompiled into the grammar using the lexicalization strategy described in the previous section. Every rational constraint, no matter what its locality domain, can be mediated locally via the symmetric feature checking mechanism of Merge. In the parlance of Müller and Sternefeld (2000), derivational ≡ global. Contrary to Müller (2005), then, locality is irrelevant for all rational constraints over derivations.

Proposition 3.18 (derivational ≡ global). For every locally unbounded rational constraint over derivation trees, there is an equivalent locally bounded one (and the other way round). □

In order to see whether locality is a meaningful parameter for rational constraints over phrase structure trees or multi-dominance trees, the general relation between derivations and derived structures must be properly understood first. As mentioned many times before (Sec. 1.1.3 and 1.2.3), the difference between derivation trees and multi-dominance trees is minimal. Except for some simple relabelings, the two are distinguished merely by the presence of a few additional branches indicating the target sites of movement. Note that this information is already present in the derivation tree, so the branches are redundant for rational constraints. The same holds for the headedness information encoded by the interior node labels, and the

linear order of constituents. Multi-dominance trees thus do not add any information, so multirepresentational $\leq$ derivational (which was already established in a more formal way in the previous section).

On the other hand, multi-dominance trees do not destroy any information, either. Every multi-dominance tree can easily be turned into its corresponding derivation tree (up to isomorphism) by deleting all movement branches and restoring the original labels of all nodes. Both tasks are quickly accomplished once one has found all Move nodes, which is simple: a node is a Move node iff it properly dominates a mother of one of its daughters. So multi-dominance trees and derivation trees provide the same amount of information.

Proposition 3.19 (multirepresentational $\equiv$ derivational). For every rational constraint over multi-dominance trees, there exists an equivalent rational constraint over derivation trees, and the other way round. □

The power of constraints over phrase structure trees depends crucially on how traces are implemented. If traces are indexed, then they provide exactly the same information as multi-dominance trees and the equivalence of representational and derivational constraints obtains as claimed by Müller (2005). However, in the MG literature traces are not indexed, nor is there a mechanism of chain formation that could keep track of which traces belong to which mover. As a result, traces in MG phrase structure trees only indicate that something has moved, but not what and where. Obviously this constitutes a loss of information, a loss that is significant enough to render representational constraints strictly weaker than derivational ones.

Lemma 3.20. Let Φ_{tr} be the natural mapping from Minimalist derivation trees to phrase structure trees with index-free traces (cf. Sec. B.4.6). Then there is a regular language R (of derivation trees) and MDTL M such that there is no regular language L (of phrase structure trees) for which $\Phi_{tr}(M) \cap L = \Phi_{tr}(M \cap R)$. □

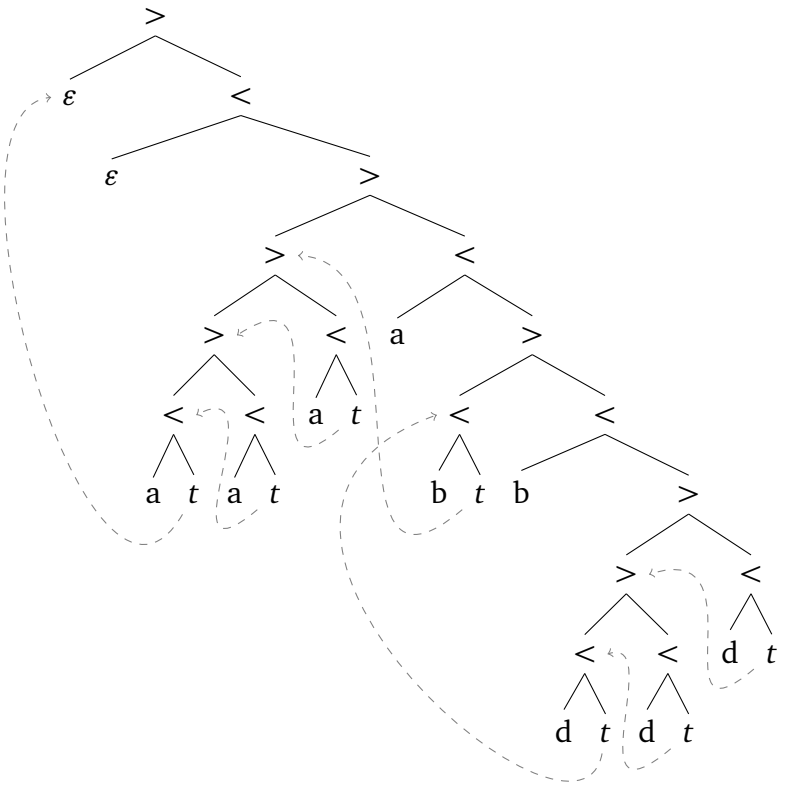


Figure 3.1: Phrase structure tree generated by MG G_+ for the string $aaaabbddd$

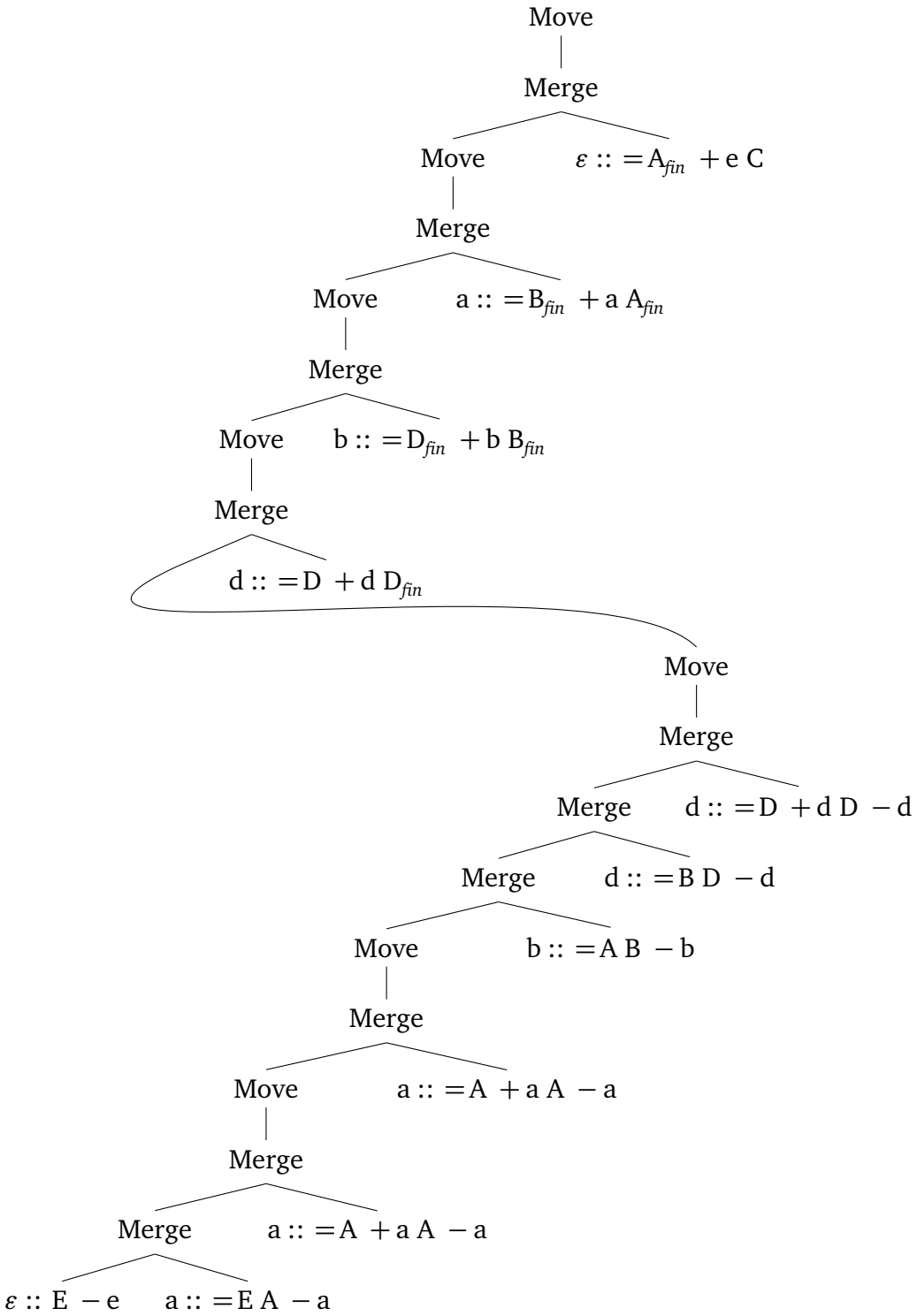


Figure 3.2: Derivation tree of G_+ for the string $aaaabbddd$

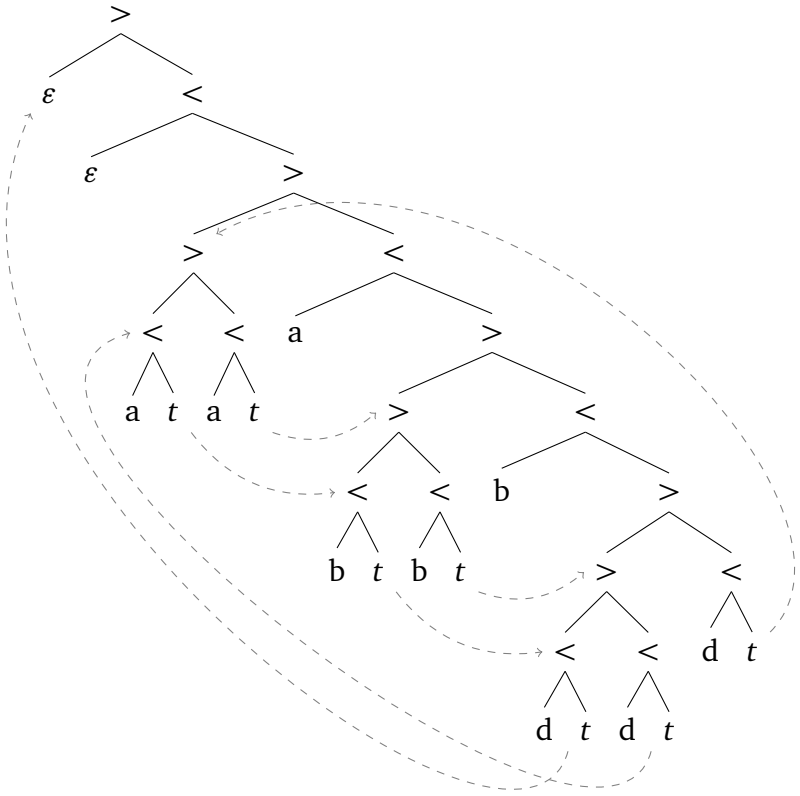


Figure 3.3: Phrase structure tree generated by MG G_n for the string $aaabbbddd$

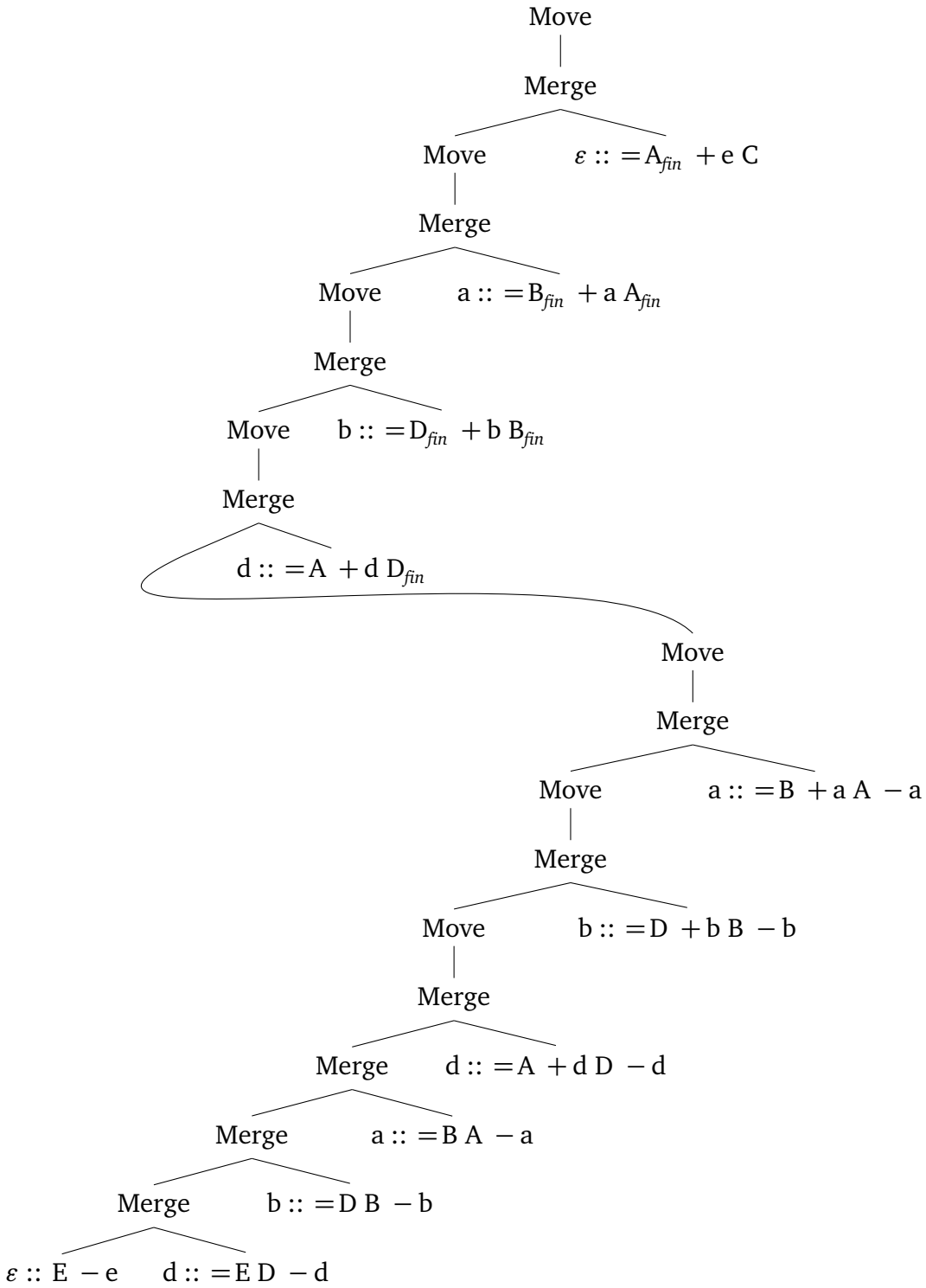


Figure 3.4: Derivation tree of G_n for the string $aaabbbddd$

Proof. Let G_+ be the following MG for the regular string language $a^i b^j d^k$, $i, j, k \geq 2$:

ε :: E $-$e	a :: $=$E A $-$a	b :: $=$A B $-$b	d :: $=$B D $-$d
	a :: $=$A $+$a A $-$a	b :: $=$B $+$b B $-$b	d :: $=$D $+$d D $-$d
ε :: $=$$A_{fin}$ $+$e C	a :: $=$$B_{fin}$ $+$a A_{fin}	b :: $=$$D_{fin}$ $+$b B_{fin}	d :: $=$D $+$d D_{fin}

This grammar uses roll-up movement to create "towers" of as, bs, and cs, respectively. The set T_x of well-formed x-towers, $x \in \{a, b, c\}$, can be specified recursively: $<(x, t) \in T_x$ and for all $y \in T_x$, $>(y, <(x, t)) \in T_x$. Each x-tower resides in the specifier of an LI of category X_{fin}. Hence every phrase structure tree in the derived tree language L_+ generated by G_+ is of the form $>(\varepsilon, <(\varepsilon, >(T_a, <(a, >(T_b, <(b, >(T_d, <(d, t))))))))$. See Fig. 3.1 on page 181 for an annotated example tree and Fig. 3.2 on page 182 for the corresponding derivation tree.

Now consider the MG G_n, which generates the mildly context-sensitive $a^n b^n d^n$, $n \geq 2$:

ε :: E $-$e	a :: $=$B A $-$a	b :: $=$D B $-$b	d :: $=$E D $-$d
	a :: $=$B $+$a A $-$a	b :: $=$D $+$b B $-$b	d :: $=$A $+$d D $-$d
ε :: $=$$A_{fin}$ $+$e C	a :: $=$$B_{fin}$ $+$a A_{fin}	b :: $=$$D_{fin}$ $+$b B_{fin}	d :: $=$D $+$d D_{fin}

Even though G_n uses remnant movement, every member of its set L_n of phrase structure trees fits the structural pattern given for G_+, with the important restriction that all towers have the same height. Example trees are given in Fig. 3.3 on page 183 and 3.4 on the previous page.

Since the string yield of L_n isn't a context-free string language, L_n is not a regular tree language. The language L_+, on the other hand, is regular, as can easily be gleamed from the pattern presented before: the set of x-towers is regular, and L_+ is simply the result of inserting x-towers at specific leafs of a finite template. Hence L_n is a non-regular proper subset of the regular set L_+. As regular languages are closed under intersection, it follows that there is no regular L such that $L_+ \cap L = L_n$.

However, let G be the MG whose lexicon is the union of the lexicons of G_+ and G_n. Now fix M and R to be the MDTLs of G and G_n, respectively. Clearly $R \subseteq M$, so $\Phi_{tr}(M \cap R) = \Phi_{tr}(R) = L_n$. ■

Proposition 3.21 (representational < derivational). Rational constraints that are stated over derivation trees are properly more powerful than rational constraints over index-free phrase structure trees. □

Putting all our findings together, we see that almost all tree-local constraint classes in the MS-hierarchy conflate into one if measured by their relative power.

Proposition 3.22 (Revised MS-Hierarchy). Within the class of rational constraints, representational $<$ multirepresentational $\equiv$ derivational $\equiv$ global □

In the next chapter, we will see that even transderivational constraints do not improve on the expressivity of derivational constraints, putting them in the same group as the majority of the other constraint classes.

3.3.2 Why use Constraints at all?

It is important to keep in mind that Prop. 3.22 only evaluates the expressive power of the respective classes. The complexity of extensionally equivalent constraints can still vary.

Example 3.14 The PBC over derivations and derived trees

The PBC is easily formulated as a constraint over trace-indexed phrase structure trees: if x and y belong to the same chain, then either x c-commands y or y c-commands x. This condition is clearly satisfied by PBC-obeying movement. Remnant movement, on the other hand, creates configurations where not all chain members are related by c-command anymore. We already encountered such a case in example 1.6 on

page 18, where the DP *das Buch* is moved into Spec,*v*P followed by movement of the VP containing its trace into Spec,CP. As a result, neither c-command each other anymore.

Stating the PBC over derivation trees is more involved. One has to ensure for every LI l that none of its occurrences are dominated by an occurrence of some LI l' such that some node of the slice of l' dominates l (the derivational equivalent of phrasal containment). While this is still conceptually transparent, it does not match the simplicity of the phrase structure constraint.

So different constraints still excel in different areas when it comes to expressing generalizations in as simple a form as possible. This is particularly noticeable when scrutinizing the refined grammar in example 3.10. Without prior knowledge of the constraint that has been lexicalized, it is difficult to make out what kind of tree-geometric property is encoded in the subscripts. Linguistic constraints are a lot more complex on average than a simple ban against trees with an even number of nodes, which can be computed using only two states. A faithful implementation of binding theory, for example, is near incomprehensible from an automaton perspective (cf. Graf and Abner 2012). This problem becomes even more pressing once multiple constraints are combined. It is safe to assume that even with only a handful of constraints, the number of distinct states pushed into the category features will quickly reach three digits, causing an enormous blow-up in the size of the refined lexicon.

Example 3.15 Feature refinement and lexical blow-up

Suppose we are given an MG consisting of 10 LIs, 5 of which select no arguments, 3 take one, and 2 two. Furthermore, there are three constraints whose respective

automata have 2, 5, and 8 states. For all intents and purposes, these numbers are unrealistically low. Yet in the worst case the fully refined lexicon will contain 13,045 distinct LIs.

Let us quickly verify how this number falls out, using the formula from Prop. 3.9 on page 172. First, instead of carrying out three refinement steps for the individual automata, they can be combined into one big automaton with $2 \cdot 5 \cdot 8 = 80$ states. The automaton accepts exactly those trees that are deemed well-formed by the original three automata. It then suffices to carry out the refinement only once for the big automaton (the eventual size of the refined lexicon does not depend on whether one uses the big automaton or the three original ones).

Out of the 10 LIs, 5 have no selector features, so they yield only $5 \cdot 80^0 = 5 \cdot 1 = 5$ refined LIs. Then there are 3 LIs with 1 selector feature, from which one obtains $3 \cdot 80^1 = 3 \cdot 80 = 240$ refines LIs. The lion's share, however, is contributed by the remaining 2 LIs with 2 selector features. In the worst case, the refinement algorithm produces $2 \cdot 80^2 = 2 \cdot 6400 = 12800$ LIs. The total number might even double if the sole unchecked feature in a well-formed derivation is always C, because in this case all LIs of category C_q must have a duplicate entry with category feature C if q is a final state. Deducing the original constraints from over twenty thousand LIs is an impossible task for any mortal (and it isn't much easier with ten thousand).

It is also interesting to observe the relative contribution of the number of states and selector features to the overall size of the refined lexicon. Increasing the number of LIs without selector features from 5 to 500 increases the total of 13,045 by only 495, so LIs without selector features are negligible. Similarly, adding 3 new LIs with a single selector feature leads to the creation of merely 240 new LIs. While 240 is a lot more than 3, it is but a fraction of the sum of $13,045 + 240 = 13,285$ LIs. The blow-up is significantly more noticeable when 2 new LIs with two selector features

are added, as this almost doubles the size to $25,845$.

The truly decisive factor, however, is the number of states needed for the constraints. Introducing one more constraint that can be computed with 2 states almost quadruples the size of the lexicon to $51,685$. This effect would be even larger if the lexicon contained LIs with more than two selector features. For instance, if the LIs with one selector feature had three such features instead, the total size would be $1,548,805$ with three constraints and $12,339,205$ after the addition of the fourth constraint, an increase by factor 8. We see that the greater the number of states and the maximum of selector features per LI, the bigger the relative blow-up.

For further illustration of this difference, take a grammar with 6 constraints, each computed by a non-deterministic automaton with 10 states (a conservative estimate for a faithful implementation of Minimalist syntax). Refinement with a non-deterministic automaton means that the state on the category feature is no longer uniquely determined by the selector features and the LI's phonetic exponent. Consequently, an LI with n selector features may yield as many new LIs under refinement via a non-deterministic automaton as an LI with $n+1$ selector features under refinement via a deterministic automaton. Then the size of the refined lexicon could exceed the number of seconds since the big bang ($\approx 4.3 \cdot 10^{17}$) if the original lexicon contains at least one LI with two selector features ($1 \cdot (10^6)^3 = 10^{18}$), while it would require at least $430,000$ LIs with one selector feature to reach the same threshold ($4.3 \cdot 10^{17}/(10^6)^2 = 4.3 \cdot 10^5$). Yet if the grammar employs only 2 out of the given 6 constraints, the number of LIs with two selector features would have to be close to 430 billion ($= 4.3 \cdot 10^{17}/10^{(2 \cdot 3)}$).

The reader should keep in mind, though, that these numbers represent the worst case where all logically possible ways of assigning states to selector features yield an LI that occurs in at least one well-formed derivation. It remains to be seen whether

linguistically plausible grammars and constraints ever come close to the worst case scenario. So while the lexical blow-up will in general be significant, it might not be quite as astronomical in practice as in this example.

The explosion of the number of LIs and the fact that there isn't a single class of tree-local constraints that is best suited to expressing all linguistic generalizations show convincingly that Prop. 3.22 only talks about the generative power of these constraint classes. Their succinctness, intuitiveness, psychological reality and explanatory adequacy are independent of this result. One might think that this deprives the equivalences of all linguistic relevance, turning them into a mathematical curiosity at best. But just like one should not overstate the importance of Prop. 3.22, one should not outright dismiss it either.

The interchangeability of the constraint classes and their expressibility via Merge still have important implications for generative syntax. For one thing, we are guaranteed that all these constraints show monotonic behavior, no matter in which way they are combined. That is to say, there are no cases analogous to the one in example 3.1, where a constraint increased the power of MGs rather than limiting it. Rational constraints are also appealing from a cognitive perspective because their computation requires only a finitely bounded amount of working memory. Moreover, the choice in favor of a specific constraint type cannot be made purely on empirical grounds, because with respect to their empirical coverage, all classes of rational constraints are equivalent (except over index-free phrase structure trees). Preferring a constraint type over others thus is a matter of explanatory adequacy.

As a matter of fact, using constraints at all — rather than relying just on Merge — is a matter of explanatory adequacy, too. Once again, there is no syntactic empirical data that could serve to distinguish the two approaches. Extra-syntactic domains like processing might provide evidence in favor of one of the two, but even if that should

turn out to be the case it would only pertain to how the grammar is implemented in the parser, not how its internals are specified. Finally, the duality of Merge and constraints does away with one of the most basic questions in Minimalism, namely why syntactic constraints like the Person Case Constraint exist at all. As Merge already provides all the power needed to express constraints, it would be puzzling if there were none. From this perspective it isn't the existence of constraints that is bewildering, but rather that the majority of rational constraints is not instantiated in any known natural language (see Graf 2012a for an extended discussion of this issue with respect to the Person Case Constraint).

3.4 Increasing the Faithfulness of MGs with Constraints 

3.4.1 Locality Conditions

In my discussion of the faithfulness of MGs in Sec. 2.3 I claimed that many missing aspects of Minimalist syntax can be incorporated via rational constraints. I now give a few examples how this might be accomplished, starting with locality constraints.

Intervention conditions and island effects such as the one below are completely absent from canonical MGs despite their ubiquity in the syntactic literature.

(4) a. Who$_i$ did John say that Bill adores t_i?

b. ?/* Who$_i$ did John doubt whether Bill adores t_i?

Without further precautions, an MG that derives (4a) will also derive (4b) as movement is restricted only by the feature calculus, not the shape of the phonological strings. An MSO-constraint can easily militate against such locality violations.

Recall from Sec. 1.2.2 and 1.2.3 that a Move node m is an occurrence of LI l iff it is involved in checking one of l's licensee features. Moreover, the predicate $occ(m,l)$ holds iff m is an occurrence of l, and $x \triangleleft^+ y$ denotes that x (properly) dominates y. Similarly, $sliceroot(y,l)$ iff y is the root of the slice of l. Finally, the

predicate whether(x) holds of x iff it is labeled by an LI with *whether* as phonetic exponent. This is simply a shorthand for $l_1(x) \vee l_2(x) \vee \cdots \vee l_n(x)$, where $l_1, \ldots, l_n$ are the lexical entries for *whether*. With these ancillary predicates, movement in (4b) is blocked by the following constraint:

$$\forall m \forall l \Big[\mathit{occ}(m,l) \rightarrow \neg \exists x \forall y \big[\text{whether}(x) \wedge m \triangleleft^{+} x \wedge (\mathit{sliceroot}(y,x) \rightarrow y \triangleleft^{+} l) \big] \Big]$$

This MSO formula ensures that no slice headed by an LI with string exponent *whether* occurs between an LI and any of its movement nodes — which is tantamount to blocking movement across a phrase headed by *whether*.

Other locality conditions follow an analogous pattern. Take the Complex NP Constraint, which blocks extraction from a CP that is complement of a noun, and the Specified Subject Constraint, which rules out movement originating from inside a DP in subject position.

(5) * Who$_i$ did John reject the claim that the lobbyists bribed t_i?

(6) a. Where$_i$ is it likely that John went t_i?

b. * Where$_i$ is that John went t_i likely?

Once again some auxiliary predicates must be defined. We use CatX(x) as a shorthand for "x is an LI with category feature X". Like whether(x), this is merely a shorthand for a disjunction $l_1(x) \vee l_2(x) \vee \cdots \vee l_n(x)$, but this time $l_1, \ldots, l_n$ are all the items with category feature X. Furthermore, $\mathit{compl}(x, \text{XP})$ holds iff x is the complement of an XP:

$$\mathit{compl}(x,l) \iff \exists y \big[\mathit{sliceroot}(y,x) \wedge \text{c-com}(l,y) \wedge \text{c-com}(y,l) \big]$$

$$\mathit{compl}(x, \text{XP}) \iff \exists y \big[\mathit{compl}(x,y) \wedge \text{CatX}(y) \big]$$

This definition uses the standard c-command definition as specified in example 3.3

on page 142, but employs it over derivation trees instead. In a derivation tree an LI c-commands the slice root of its complement, and the other way round. The same trick can be used to pick out specifiers, whose slice roots are dominated by the slice root of the LI they c-command.

$$spec(x, \text{XP}) \iff \exists y \exists z \exists z' \big[sliceroot(y, x) \wedge sliceroot(z', z) \wedge \text{CatX}(z) \wedge \text{c-com}(y, z) \wedge z' \triangleleft^{+} y \big]$$

The Complex NP Constraint and Specified Subject Constraint now can be expressed as minimal variants of each other (assuming that subjects are simply DPs residing in Spec,TP).

$$\forall m \forall l \Big[occ(m, l) \rightarrow \neg \exists x \forall y \big[\text{C}(x) \wedge compl(x, \text{NP}) \wedge (sliceroot(y, x) \rightarrow m \triangleleft^{+} y \wedge y \triangleleft^{+} l) \big] \Big]$$

$$\forall m \forall l \Big[occ(m, l) \rightarrow \neg \exists x \forall y \big[\text{D}(x) \wedge spec(x, \text{TP}) \wedge (sliceroot(y, x) \rightarrow m \triangleleft^{+} y \wedge y \triangleleft^{+} l) \big] \Big]$$

The *that*-trace filter combines the structural component of the last two constraints with the sensitivity to phonetic material we saw with respect to *whether*. In general, a wh-word can be extracted out of a CP headed by *that*, but not if it is the subject of the clause.

(7) a. What$_i$ did you say that John ate t_i?

b. * Who$_i$ did you say that t_i ate my burrito?

$$\forall m \forall l \Big[occ(m, l) \rightarrow \exists x \exists y \big[(\text{that}(x) \wedge sliceroot(y, x) \wedge m \triangleleft^{+} y \wedge y \triangleleft^{+} l) \rightarrow \neg spec(l, \text{TP}) \big] \Big]$$

Locality conditions a general pattern of the form $\forall m \forall l [occ(m, l) \rightarrow \phi]$, where ϕ is some MSO-formula (with m and l as its only free variables, if it has any). Many more constraints can be captured this way, for instance the Coordinate Structure

Constraint, the Left Branch Condition, and barriers or phases (but see my remarks on successive cyclic movement in Sec. 2.3.3). Many of the principles formalized in Rogers (1998) can also be adapted for MGs, although the change from derived trees to derivation trees will require some marginal revisions in certain cases, in particular binding and control, which rely on representational c-command and thus need to be computed slightly differently on a derivational level.

3.4.2 Agreement and Pied-Piping

Through the use of constraints we can also reduce the number of movement steps in our grammars. In early Minimalism Chomsky (1995c), satisfying feature dependencies between non-adjacent phrases invariably required movement, an assumption inherited by MGs. In such a setup, subject-verb agreement, say, is assumed to be an effect of the subject moving into the specifier of the TP and checking its person features. But other instances of agreement, e.g. between determiners or adjectives on the one hand and nouns on the other, are rather cumbersome to handle this way. This brought about a major revision of the feature calculus in order to make feature checking apply at a distance in certain cases Chomsky (2001). As long as we do not allow unbounded nesting and crossing of the checking paths defined by this operation, rational constraints can yield the same effect by associating every lexical item with "pseudo-features" that encode properties not pertinent to movement.

For instance, the Icelandic adjective *rauðan* 'red', which is masculine, singular, accusative, and strongly inflected, could be assigned the corresponding pseudo-features, represented by the predicates $masc(x)$, $sing(x)$, $acc(x)$, and $strong(x)$. As before, each such predicate is simply a disjunction of LIs. Assume for the sake of exposition that adjectives take a single complement which is either a noun or another adjective. Then the constraint below enforces basic number agreement between

nouns and adjectives.

$$\forall x, y \left[(\text{CatA}(x) \wedge \mathit{compl}(y, x)) \rightarrow (\mathit{sing}(x) \leftrightarrow \mathit{sing}(x)) \wedge (\mathit{pl}(x) \leftrightarrow \mathit{pl}(x)) \right]$$

The structural condition in the antecedent of the implication can be altered freely to accommodate other configurations, and the agreement dependencies on the right hand of the implication can reach any empirically desirable degree of complexity.

A more interesting case is long-distance subject-verb agreement in English expletive constructions.

(8) a. * There seems to John to be several men in the garden.

b. There seem to John to be several men in the garden.

In the case of adjective-noun agreement above one could easily refine categories by hand to ensure the right agreement patterns, but for long-distance dependencies the automatic refinement via MSO-constraints is the preferable solution. The general format can remain the same, only the subformula $\text{CatA}(x) \wedge \mathit{compl}(y, x)$ needs to be replaced by a suitable structural description.

But the application of pseudo-features need not stop here. They also make it possible to add a restricted version of pied-piping to MGs. In pied-piping, a constituent containing some element with a movement licensee feature seems to be stuck to it for the purposes of movement.

(9) a. [Which famous linguist]$_i$ did Robert write a book [about t_i]?

b. [About which famous linguist]$_i$ did Robert write a book t_i?

In the literature this is sometimes analyzed as the movement licensee feature of the DP percolating upwards into the PP. This strategy is shunned by some syntacticians as too stipulative and overly permissive, and rightly so, for MGs with such a feature percolation mechanism can generate any recursively enumerable language (Kobele 2005). But at least for the example above, only a very limited kind of feature

percolation is required. We could have two lexical entries for *about* and *which*, one each with a wh-licensee feature and one each without said feature. In order to prevent overgeneration, one then requires that *about* may have a licensee feature only if its complement is headed by *which*. More generally, one may stipulate that the complement is headed by an LI that could in principle carry a wh-licensee feature. This is trivial if one uses the pseudo-feature mechanism described above. (For an alternative implementation of pied-piping as a new type of movement see Graf (2012c).)

3.4.3 Relaxing the SMC

Dynamic restrictions on the distribution of features also allows us to work around certain shortcomings of the SMC. The SMC — albeit essential for keeping MDTLs within the confines of regular tree languages — comes with its fair share of linguistic inadequacies, in particular with respect to wh-movement. Since English allows for wh-phrases to stay *in situ*, every wh-phrase in an MG must come in two variants, one with a wh-licensee feature, and one without it. But given this duality, nothing prevents superiority violations like the one in (10b) (for the sake of simplicity, only wh-movement is indicated by traces).

(10) a. Who$_i$ t_i prefers what?

b. * What$_i$ does who prefer?

The ungrammatical (10b) can be derived because *who* need not carry a wh-licensee feature, in which case the MG will treat it like any other DP. Consequently, nothing prevents *what* from carrying a wh-licensee feature and moving to Spec,CP.

Instances of overgeneration like this can also be blocked via pseudo-features: rather than checking whether any LI along the movement path has the same active licensee feature, one must determine whether it could potentially have an active licensee feature. In (10b) above, this condition is violated because *who* could be a

carrier of an active wh-licensee feature.

This strategy can even be extended to multiple wh-movement. MGs struggle with this kind of movement, because either the grammar has only one wh-feature, in which case multiple wh-movement is impossible, or there are several types of wh-features, in which case those features are incorrectly regarded as distinct by the SMC and even the most basic superiority violations are no longer captured. A grammar with multiple distinct wh-features but a single pseudo-wh-feature gets us the best of both worlds. The SMC no longer applies to these features and all desired locality restrictions on wh-movement can be enforced as MSO-constraints that are stated with respect to pseudo-features instead.

3.5 The Chapter in Bullet Points

- *Logic and Constraints*
  - Constraints are modeled as logical formulas without free variables.
  - The power of a constraint is reflected by the power of the weakest logic(s) that can express it.
  - MSO allows for quantification over nodes and sets of nodes and thus is powerful enough to state the majority of constraints proposed in the syntactic literature (demonstrated by the MSO-implementation of GB in Rogers 1998).
  - MSO-definable constraints are called rational.
  - Every MSO-formula (and thus every rational constraint) can be converted into an equivalent deterministic bottom-up tree automaton, and *vice versa*.
- *Constraints and Merge*

- Once a rational constraint has been converted into an equivalent deterministic automaton, the automaton can be precompiled into the grammar.
- For every LI, its category feature is subscripted with the state that the automaton assigns to its slice root in some derivation. If there are several such states, multiple copies of the LI are created that differ only in their subscript.
- The selector features must also be refined accordingly, except that the subscript is determined by the state of the slice root of the selected argument.
- The result of the refinement is an MG that enforces the constraint through the symmetric feature calculus governing Merge. The derived structures generated by this grammar are indistinguishable from the ones generated by the original grammar with the constraint, and their derivations differ only in the names of category and selector features.
- It follows that a constraint over derivations can be expressed by Merge iff it is rational.
- As the result depends only on the symmetric feature checking of Merge, it is independent of the types of movement in the grammar. In particular, it is irrelevant whether the SMC holds.

- *Classes of Constraints*
  - In the spirit of Müller and Sternefeld (2000), constraints are classified according to the structure they operate on (phrase structure tree with no indexed traces, phrase structure tree with indexed traces, multi-dominance tree, derivation tree) and the size of their application domain (bounded tree-local, unbounded tree-local, set of competing trees).
  - Rational constraints over phrase structure trees without indexed traces

are properly weaker than local rational constraints over derivations. All other types of tree-local rational constraints are equally powerful.

CHAPTER 4

Transderivational Constraints

Contents

The previous chapter established that a constraint can be lexicalized and expressed via Merge iff it is rational, i.e. MSO-definable. From this the interdefinability of all the subclasses of tree-local constraints could be derived, with the exception of constraints over phrase structure trees without indexed traces. A basic understanding of MSO also allows one to see that the majority (if not all) of tree-local constraints in the syntactic literature are rational. This still leaves open the issue of translocal and transderivational constraints, which according to the classification in Tab. 3.1 on page 151 aren't tree-local but rather reference-set constraints (RCs). Are these constraints more powerful than tree-local ones? Or equivalently, can they be expressed via Merge?

The main problem in tackling this issue is that the formalization of tree-local constraints in terms of MSO-formulas does not extend elegantly to RCs. These constraints are relative and violable. Rather than enforcing absolute well-formedness conditions that are checked against single trees, an RC takes as input a set of trees and determines which one of them satisfies a given condition best. The tree in question might still violate the condition several times, and for arbitrary sets there might not be an upper bound on the number of violations even among the best matches. MSO-formulas are ill-suited to capture this behavior. Under their standard interpretation, they apply to single trees, and they are either true in a tree or not. Illicit trees aren't valued differently depending on how close they come to being a model of the formula. Hence RCs need a new model that can measure gradients in well-formedness between trees and translate those back into the categorical system of grammaticality.

OT-style grammars are a natural candidate as they, too, rank competing structures and dismiss all but the best among them as ungrammatical. Frank and Satta (1998) propose to model OT in terms of transducers, i.e. automata with output. A transducer reads an input structure and creates an output structure guided by the shape of the

former. In the case of a transderivational constraint, the transducer would transform a suboptimal derivation into an optimal one. Notice how this strategy subverts the standard linguistic dichotomy of constraints versus operations. RCs are turned from filters that weed out suboptimal forms into rewriting rules that transform suboptimal outputs into optimal ones.

Counterintuitive as a procedural view of RCs might be to linguists, it is the more fruitful one for mathematical purposes. The central result of this chapter arguably could not have been obtained otherwise.

Proposition 4.1. A transderivational constraint can be expressed by Merge iff it can be modeled by a regularity-preserving transduction from an MDTL into itself. □

This means that if the output of an RC is both regular and a subset of the MDTL the constraint applies to, then the constraint can be expressed by Merge. Putting it more bluntly, one might say that a transderivational constraint can be expressed by Merge iff it can be converted into an equivalent derivational constraint. The line of reason leading to this result is formally developed in Sec. 4.1. In order to show that many RCs actually fall into this subclass (and thus are reducible to purely derivational constraints) I implement three well-known specimen from the literature: Focus Economy (4.2), Merge-over-Move (4.3), and the Shortest Derivation Principle (4.4). Most constraints in the syntactic literature are merely variants or combinations of these constraints from a mathematical perspective.

As this chapter builds a new formal model, few prerequisites carry over from the previous ones. Familiarity with MGs and MDTLs is still necessary, though, and knowledge of the regularity of MDTLs and their p-closure under intersection with regular tree languages is indispensable for understanding the validity of the formal argument. Seeing how transducers are essentially automata with output, it is also helpful to have worked through the examples of bottom-up tree automata in Sec. 2.1.1, and a little bit of MSO is used for the implementation of Focus Economy

in Sec. 4.2.

4.1 Transderivational Constraints as Rewriting Rules

4.1.1 Examples of Reference-Set Constraints

Out of all the items in a syntactician's toolbox, RCs are probably the most peculiar one. When handed some syntactic tree, an RC does not determine its well-formedness from inspection of the tree itself. Instead, it constructs a *reference set* — a set containing a number of trees competing against each other — and chooses the optimal candidate from said set.

Consider *Fewest Steps*, also known as the *Shortest Derivation Principle* (Chomsky 1991, 1995b). For any derivation tree t, the Shortest Derivation Principle constructs a reference set that consists of t itself and all the derivations that were assembled from the same LIs as t. The derivations are then ranked by the number of Move nodes they contain such that the tree(s) with the fewest instances of movement is (are) chosen as the winner. All other trees are flagged as ungrammatical, including t if it did not emerge as a winner.

Another RC is *Focus Economy* (Szendrői 2001; Reinhart 2006), which is supposed to account for the empirical fact that neutral stress is compatible with more discourse situations than shifted stress. Take a look at the utterances in (11), where main stress is indicated by **bold face**. Example (11a) can serve as an answer to various questions, among others "What's going on?" and "What did your neighbor buy?". Yet the virtually identical (11b), in which the main stress falls on the subject rather than the object, is compatible only with the question "Who bought a book?". These contrasts indicate a difference as to which constituents may be *focused*, i.e. can be interpreted as providing new information.

(11) a. My neighbor bought a **book**.

b. My **neighbor** bought a book.

Focus Economy derives the relevant contrast by stipulating that first, any constituent containing the node carrying the sentential main stress can be focused, and second, in a tree in which stress was shifted from the neutral position a constituent may be focused only if it this would not be possible in the original tree with unshifted stress. In (11a), the object, the VP and the entire sentence can be focused, since these are the constituents containing the main stress carrier. In (11b), the main stress is contained by the subject and the entire sentence, yet only the former may be focused because focusing of the latter is already a licit option in the neutral stress counterpart (11a).

The application domain of RCs includes narrow syntax as well as the interfaces. In syntax, one finds Fewest Steps (Chomsky 1995c), Merge-over-Move (Chomsky 1995c, 2000), Pronouns as Last Resort (Hornstein 2001), resumption in Lebanese Arabic (Aoun et al. 2001), phrase structure projection (Toivonen 2001), the Person Case Constraint (Rezac 2007), Chain Uniformization (Nunes 2004), object extraction in Bantu (Nakamura 1997), and many others. The most prominent interface constraints are Rule I (Grodzinsky and Reinhart 1993; Heim 1998; Reinhart 2006; Heim 2009), Scope Economy (Fox 1995, 2000), and the previously mentioned Focus Economy, but there are also more recent proposals such as Situation Economy (Keshet 2010). We will see later on that syntactic RCs are usually reducible to tree-local constraints, while some prominent constraints from semantics such as Rule I pose a challenge that might be insurmountable. This mirrors earlier observations in Sec. 3.1.3 regarding the limitations of MSO when it comes to semantics. As a rule of thumb, it must be possible to describe a constraint in purely structural terms in order for it to be expressible via Merge — the meaning of a subtree cannot be taken into account.

The viability and adequacy of RCs, in particular transderivational ones, is a contentious issue. It has been conjectured that they are computationally intractable (Collins 1996; Johnson and Lappin 1999), too powerful (Potts 2001), or make the

wrong empirical predictions (Sternefeld 1996; Gärtner 2002). The next section is a refutation of these claims. We will see that reference-set constraints can be implemented as finite-state devices; linear bottom-up tree transducers (lbutts), to be precise. Lbutts are of interest for theoretical as well as practical purposes because regular tree languages are closed under linear transductions, so applying a linear transducer to a regular tree language yields a regular tree language again. As MDTLs are p-closed under intersection with regular tree languages, they are also closed under linear transductions that map an MDTL to a subset of itself. It follows immediately that every reference-set constraint fitting this criterion can be equivocated with the output language defined by the transduction, which in turn can be represented by an automaton that can be lexicalized. Hence reference-set constraints can be expressed by Merge, too.

4.1.2 Introducing Tree Transducers

As just noted, RCs will be modeled in terms of tree transducers. Tree transducers can be viewed as a particular kind of SPE- or Aspects-style rewriting system, and this was indeed the linguistic motivation for their introduction in Rounds (1970). The connection between transducers and rewriting systems brings about an intriguing shift in perspective regarding RCs: rather than filtering out suboptimal trees in a rather arcane, non-local way, RCs rewrite them into optimal ones using what may be considered a subclass of syntactic transformations.

The rewriting procedure can be viewed as a generalization of the assignment of states to nodes carried out by bottom-up tree automata. When handed a tree as its input, a transducer moves through said tree in a bottom-up fashion, from the leaves towards the root, possibly relabeling nodes, deleting subtrees, or inserting new structural material. Once it reaches the root, the end result of its modifications is either thrown away or returned as an output tree, depending on whether the end result is deemed well-formed. Sometimes there are several ways to manipulate an

input tree, and in these cases the transducer might create multiple output trees, just like a non-deterministic automaton might have several states it could assign to a given node.

Table 4.1 on the following page depicts the rules of a transducer for simple instances of wh-movement. Each one consists of a left-hand side, a rewrite arrow, and a right-hand side. The left hand side varies depending on whether the transducer is at a leaf node or a non-terminal node. In the former case, it comprises only the label of said node, as in rules (1) and (2). Otherwise, it specifies the label of the current node, plus the states of the daughters of the node. But in contrast to the states of an automaton, these states immediately dominate the subtrees rooted by the daughters of the current node and will be removed once the transducer moves to the next higher node. Like an automaton, a transducer may only consider (I) the label of node it is currently at, and (II) the state symbols said node dominates in order to determine which rewrite rule to apply. So rule (5), for instance, may be applied if and only if the current node is labeled TP, its left daughter is q_*, and its right daughter is q_{wh}.

On the right-hand side of rules (3), (4) and (5), the states dominated in the left-hand side are gone. Instead, a new state was added on top of the new output subtree. Similarly, the right-hand sides of rules (1) and (2) each contain a state dominating the leaf node. This setup allows left-hand sides and right-hand sides to interact as follows in order to push states upwards through the tree during the rewrite steps: First, the transducers reads the currents node label and the states it dominates, if they exist. Depending on the applicable rewrite rules, it may leave this part of the tree unaltered (rule (1)), change the label (rule (2)), insert new structure (rule (5)), or delete a subtree (the last option is not featured in our example, but could easily be obtained from, say, rule (4) by removing one of the two subtrees in the right-hand side). Irrespective of how the subtree is rewritten, though, the transducer must put a state symbol on top of it, i.e. closer to the root. Note that

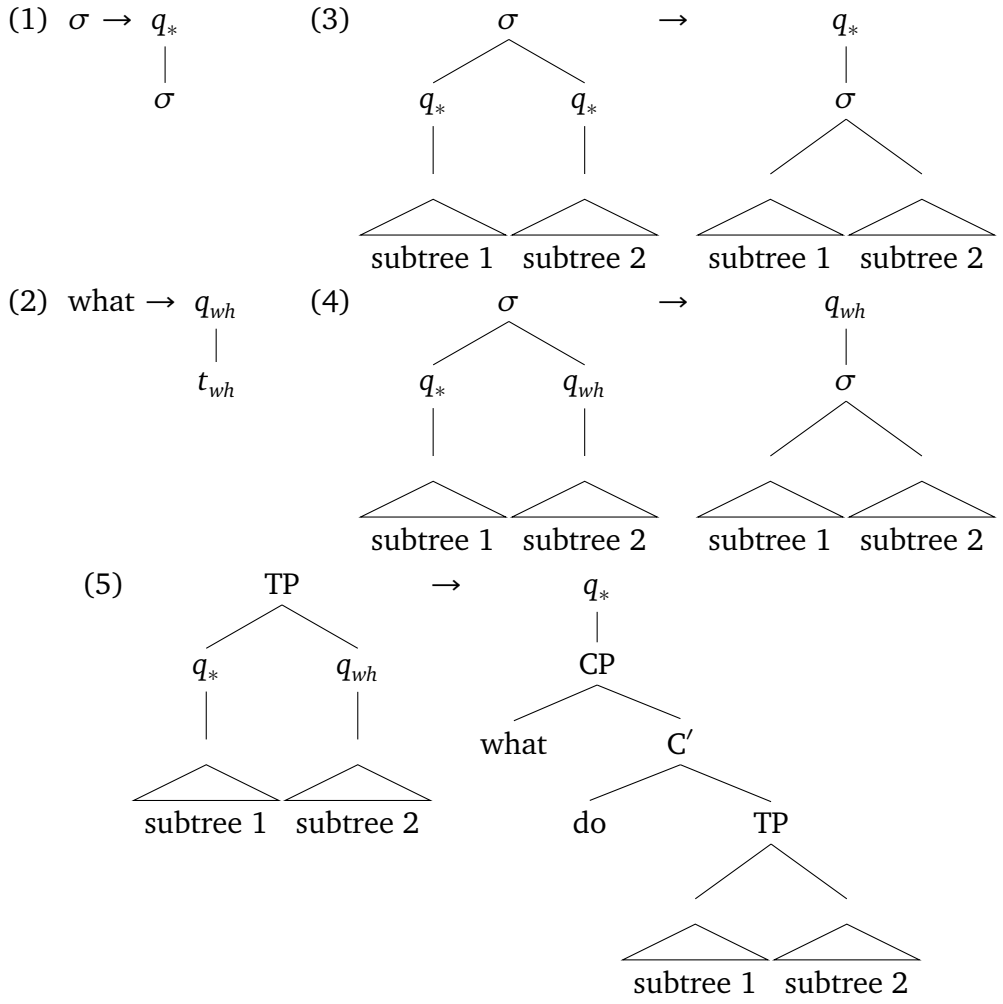


Table 4.1: Rules of a transducer for simplified wh-movement; only q_* is a final state

this way of rewriting makes it necessary to traverse trees bottom-up. One starts by rewriting leaves adding states on top of them, then one rewrites the mothers of the leaves (which involves removing the old states and again adding a new one on top), after that the next higher mothers, and so on, until one finally reaches the root. The transducer deems the output tree well-formed only if the state dominating the root node is a final state, a notion that once again should already be familiar from our earlier discussion of tree automata. For each transducer one has to specify in advance which states are final states.

Let us work through the example in Tab. 4.1 in greater detail now. As I mentioned

in passing before, the transducer is supposed to model very simple instances of wh-movement (wh-movement was chosen because it should be sufficiently familiar to all readers that they can fully focus their attention on the mechanics of the transducer). Only two states are used, q_* and q_{wh}. The former indicates that nothing was changed in the subtree dominated by q_* — we may call it the identity state — whereas q_{wh} signals that somewhere in the subtree it dominates, a wh-word was replaced by a trace.

The five rules of the transducer can now be paraphrased as follows. First, note that σ is used as a shorthand for any label, so an instruction to rewrite σ as σ instructs the transducer to keep the current label. Consequently, rule (1) tells the transducer that if it is at a leaf node, it should leave said node unaltered and record this decision by adding the state q_* on top. Rule (2), on the other hand, allows for leaf nodes labeled *what* to be rewritten by wh-traces. Naturally we add the state q_{wh} this time. Crucially, the two rules are not in conflict, and the transducer may choose freely between rule (1) and (2) whenever it encounters a leaf labeled *what* (since σ matches any label). Hence the transducer creates several output trees for inputs with wh-words, some with wh-in-situ and some with wh-movement.

Rule (3) and (4) are fairly unremarkable insofar as they merely ensure that the transducer does not manipulate non-terminal nodes and that q_{wh} is percolated upwards as necessary. If we did not take care to carry along q_{wh} at every step of the rewriting process, then the transducer would "forget" that it had replaced a wh-word by a trace earlier on. That is to say, it would merely remove wh-words rather than displace them. Finally, rule (5) tells the transducer to add a CP with the wh-word on top of a TP if rule (2) was applied at some earlier point. Note that if q_{wh} is a final state, rule (5) need never apply since output trees in which the transducer failed to switch back from q_{wh} into q_* before reaching the root node would also be considered acceptable. Hence only q_* may be a final state if we want wh-words to be reinserted into the tree after they have been replaced by traces.

A transduction using all five rules is given in Fig. 4.1 on the following page. Except for deletion, it shows off all the capabilities of transducers that will be needed for RCs, in particular relabelings, the insertion of new material, and the ability to use states to both memorize limited amounts of structural information and decide whether output trees should be accepted or discarded.

Three different RCs will be implemented as transducers in this chapter, so in order to save space while maintaining maximum clarity I will provide the full mathematical definitions. This requires that all the notation for transducers is in place.

Definition 4.2. A *bottom-up tree transducer* is a 5-tuple $\mathscr{A} := \langle \Sigma, \Omega, Q, F, \Delta \rangle$, where Σ and Ω are finite ranked alphabets, Q is a finite set of states, $F \subseteq Q$ the set of final states, and Δ is a set of productions of the form $f(q_1(x_1), \ldots, q_n(x_n)) \rightarrow q(t(x_1, \ldots, x_n))$, where $f \in \Sigma$ is of rank n, $q_1, \ldots, q_n, q \in Q$, and $t(x_1, \ldots, x_n)$ is a tree with the node labels drawn from $\Omega \cup \{x_1, \ldots, x_n\}$.

Note that the trees in the transition rules are represented in functional notation such that $f(q_1(x_1), \ldots, q_n(x_n))$ denotes a tree whose root f dominates the states $q_1, \ldots, q_n$, which in turn dominate the subtrees represented by the variables $x_1, \ldots, x_n$. Using this format, rule (5) could be written as $\text{TP}(q_*(x), q_{wh}(y)) \rightarrow q_*(\text{CP}(\text{what}, \text{C}'(\text{do}, \text{TP}(x, y))))$.

Later in this section, it will sometimes be more convenient to specify a transduction top-down rather than bottom-up. In this case, the tree on the left-hand side consists of a state dominating a node with n subtrees, and the right-hand side is a tree with states dominating the subtrees from the left-hand side. For example, $q(f(x, y)) \rightarrow g(a, b, q'(x))$ would be a transduction step that applies if state q dominates a subtree with root f. This tree is then rewritten as a tree rooted by g, which has three daughters: a, b, and the original subtree, dominated by state q'.

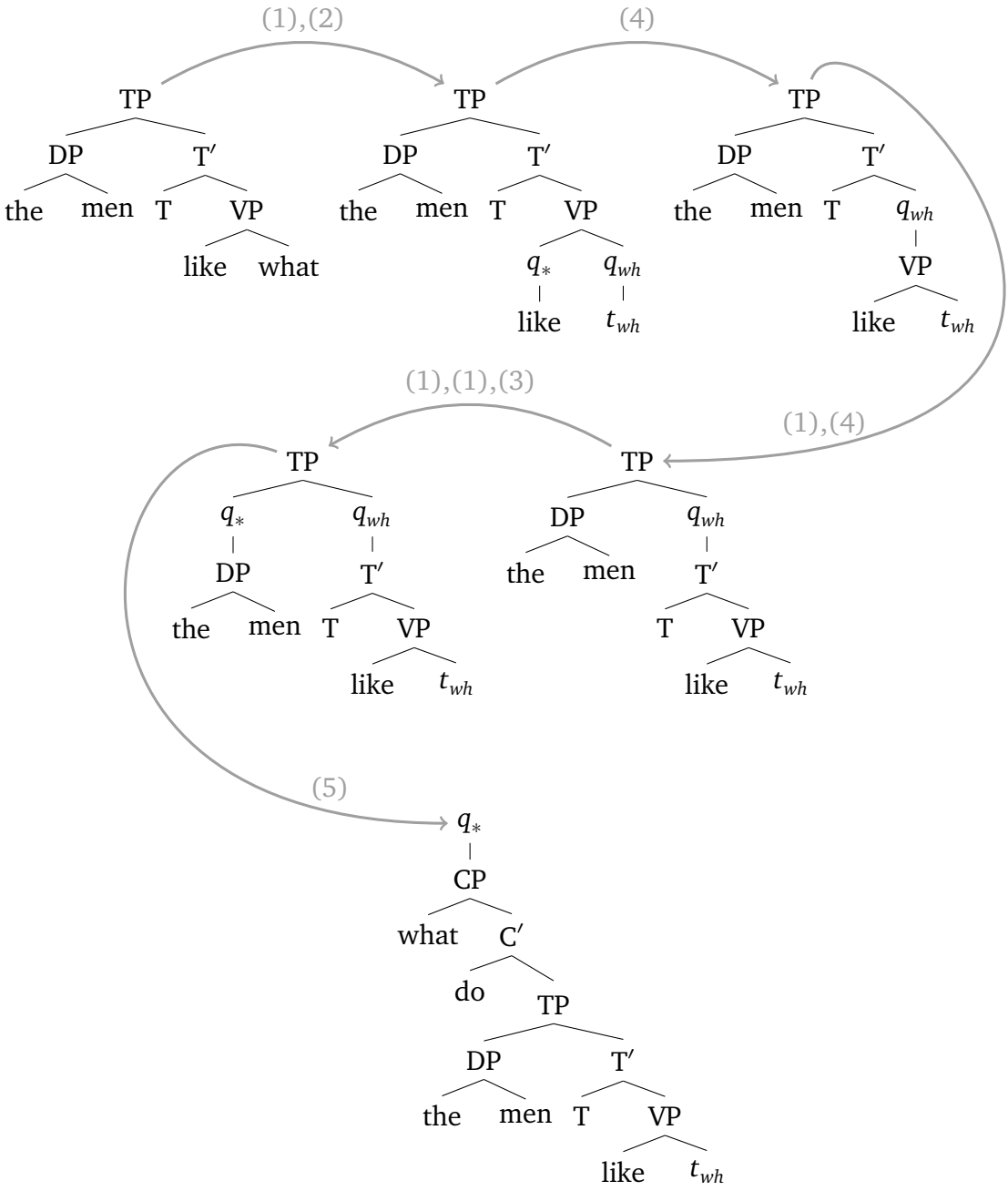


Figure 4.1: Example of transduction for simple wh-movement

Definition 4.3. A *top-down tree transducer* is 5-tuple $\mathscr{A} := \langle \Sigma, \Omega, Q, F, \Delta \rangle$, where Σ, Ω and Q are as before, $F \subseteq Q$ is the set of initial states, and all productions in Δ are of the form $q(f(x_1, \ldots, x_n)) \to t$, where $f \in \Sigma$ is of rank n, $q \in Q$, and t is a tree with the node labels drawn from $\Omega \cup \{q(x) \mid q \in Q, x \in \{x_1, \ldots, x_n\}\}$.

For the sake of succinctness (but to the detriment of readability), I adopt the following notational conventions for tree transducer productions:

- $\alpha_{\{x,y\}}$ is to be read as "α_x or α_y".
- $\alpha_{a\ldots z}(\beta_{a'\ldots z'}, \ldots, \zeta_{a'',\ldots,z''})$ is to be read as "$\alpha_a(\beta_{a'}, \ldots, \zeta_{a''})$ or ... or $\alpha_z(\beta_{z'}, \ldots, \zeta_{z''})$".

Example 4.1 Syntactic sugar for transductions

The production $\sigma(q_{ij\{a,b\}}(x), q_{jkc}(y)) \to q_{\{a,c\}}(\sigma(x,y))$ is a schema defining eight productions:

$$\sigma(q_i(x), q_j(y)) \to q_a(\sigma(x,y)) \qquad \sigma(q_i(x), q_j(y)) \to q_c(\sigma(x,y))$$
$$\sigma(q_j(x), q_k(y)) \to q_a(\sigma(x,y)) \qquad \sigma(q_j(x), q_k(y)) \to q_c(\sigma(x,y))$$
$$\sigma(q_a(x), q_c(y)) \to q_a(\sigma(x,y)) \qquad \sigma(q_a(x), q_c(y)) \to q_c(\sigma(x,y))$$
$$\sigma(q_b(x), q_c(y)) \to q_a(\sigma(x,y)) \qquad \sigma(q_b(x), q_c(y)) \to q_c(\sigma(x,y))$$

Only a proper subclass of bottom-up and top-down tree transducers will be used here, namely *linear* transducers. A production is linear if each variable in its left-hand side occurs at most once in its right-hand side. That is to say, copying of subtrees is forbidden. A transducer is linear if each production is linear. I denote a linear bottom-up/top-down tree transducer by lbutt/ltdtt. The class of ltdtts is properly contained in the class of lbutts, which in turn is closed under union and

composition (Engelfriet 1975). The domain and the range of an lbutt are both regular tree languages. The relation τ induced by a (linear) tree transducer is called a (linear) *tree transduction*. A bottom-up tree transducer rewrites s as t iff s and t are Σ- and Ω-labeled trees, respectively, and for some $q \in F$, $q(t)$ can be obtained from s by finitely many applications of productions $\delta \in \Delta$. The definition is almost unchanged for top-down tree transducers, except that we require that t can be obtained from $q(s)$ and $q \in F$.

4.1.3 Putting it All Together

So far we have seen MGs as a formalization of Minimalist syntax and transducers as a potential mathematical model of transderivationality, but it is still unclear how the two combine to reveal something fundamental about RCs.

The answer, albeit building on highly technical insights (Engelfriet 1975) is simple: if an RC can be be modeled as a linear transducer that rewrites suboptimal derivations of some MDTL M into optimal ones, then the set of these optimal derivations is a regular subset of M. The regularity of this subset is guaranteed by the fact that MDTLs are regular and regularity is preserved under linear transductions. The subset, in turn, can be viewed as a derivational constraint, which as we know can be expressed purely via Merge. RCs that can be modeled by transducers thus do not increase the power of the MG formalism.

A lot of the formal machinery can be readily adapted from previous work on the computational implementation of OT (Frank and Satta 1998; Wartena 2000; Jäger 2002; Kepser and Mönnich 2006). Frank and Satta (1998) propose to model OT as a system for describing transductions in a modular way. Rather than having one big transduction that immediately maps inputs to optimal outputs, OT starts out with the Generator as a very simple transduction from inputs to output candidates. Each constraint in turn defines a transduction from output candidates to optimal

candidates. In order to get the actual set of outputs, all the transductions have to be applied in sequence: the input is fed into the generator transduction, the output of said transduction into the transduction defined by the first constraint, the ouputs of this transduction into the transduction defined by the second constraint, and so on. Frank and Satta prove that given certain restrictions on the complexity of the individual constraints, the transductions can be composed into one big transduction that directly maps inputs to outputs and is no more powerful than the individual transductions by themselves. In particular, regularity of the input language is preserved.

Wartena (2000) and Kepser and Mönnich (2006) generalize Frank and Satta's ideas to tree languages, which are our primary concern here. If the input language belongs to the class of regular or linear context-free tree languages and the transduction realized by the OT grammar is a linear tree transduction, then the output language is also regular or linear context-free, respectively. But what kind of OT-style constraints can be captured by linear tree transductions? Frank and Satta's original proposal is very restrictive in that it only accommodates constraints that have an upper bound on the number of violations. That is to say, past some threshold n trees with n and $n+1$, $n+2$, ... violations are considered equally ungrammatical. This is clearly inadequate for OT, and it is also problematic for many RCs. Consider the Shortest Derivation Principle, which one might view as an OT-constraint that is violated once for every movement step in the derivation. As there is no upper bound on the length of derivations, even the most optimal output for some input might have more than n movement steps. But then all derivations with more than n movement nodes will be considered equally optimal, and no filtering takes place at all. In order to capture RCs, one needs a method to express these kinds of counting constraints by linear tree transductions without some artificial bound on the maximum number of registered violations.

Jäger (2002) uses a brilliant trick to get around the upper bound on constraint

violations. Rather than directly modeling a constraint's mapping from candidates to optimal outputs as a transduction, Jäger proposes to model the ranking induced by the constraint as a transduction, and to derive the correct mapping from the ranking.

Example 4.2 Rankings via transductions

A quick toy example might clarify the difference between the two types of transductions. Suppose that constraint C evaluates four inputs i, j, k and m such that m has more violations than k, k has more violations than i and j, which both incur the same number of violations. The transduction realized by C rewrites the four inputs as the optimal outputs, in this case i and j. We may informally denote this by $i, j, k, m \rightarrow i, j$. The ranking induced by C, on the other hand, is $i = j < k < m$. The transduction expressing this ranking rewrites i and j as k, and k as m.

This approach gets us around the upper bound because transductions are defined on structures of unbounded size, so for every candidate, no matter how deviant, it is in principle possible to rewrite it as an even less optimal one.

Example 4.3 A transduction for $^*a$

Suppose we have a constraint $^*a$ that punishes every occurrence of the symbol a in a string, and the inputs to the constraint are all of the form ba^+, i.e. a string that stars with a b, followed by one or more as. Then $^*a$ induces the ranking $ba < baa < baaa < \ldots$. This ranking can be expressed by a string transduction that rewrites a given string ω as ωa. This transduction is defined for every input and there is no greatest element that cannot be rewritten by the transduction, or must be rewritten by itself, so the transduction really defines the same infinite ranking as the constraint.

Deriving the input-output mapping from the ranking of candidates takes some ingenuity. The crucial insight is that the optimal output candidates are the least elements in the ranking. Hence an output candidate o is optimal only if there is no other output candidate o' such that o' is rewritten as o by the ranking transduction. Using several closure properties of regular (string or tree) languages, one can isolate the optimal candidates and build a transduction that discards all suboptimal candidates. First, one looks at the set of outputs produced by the ranking transduction. This is the set of suboptimal outputs. If one subtracts this set from the set of inputs to the constraint, one winds up with the set of optimal output candidates. One can then take the *diagonal* of this set — the transduction that rewrites every element as itself. This transduction is only defined for the optimal candidates in the input, and as a result all suboptimal outputs are discarded. As the diagonal of a regular tree language is guaranteed to be a linear transduction, this method yields the desired type of transduction for constraints.

Example 4.4 From rankings to optimal outputs

Let us return to example 4.2 for a moment. There we had a constraint C that applies to inputs i, j, k and m and induced a ranking $i = j < k < m$, so that only i and j aren't filtered out by the constraint. Now suppose that we also have a transduction τ that rewrites i and j as k, and k as m, mirroring the ranking by C. However, nothing is rewritten as i and j. We can write this more succinctly as $\tau(i) = k$, $\tau(j) = k$, $\tau(k) = m$, $\tau(m) =$ undefined. So applying τ to the set $\{i, j, k, m\}$ yields the set $\{k, m\}$. Subtracting the latter from the former, we get $\{i, j, k, m\} \setminus \{k, m\} = \{i, j\}$. The diagonal of $\{i, j\}$ is a transduction δ that rewrites i as i and j as j but is undefined for k and m. Hence if one applies δ to $\{i, j, k, m\}$, the output is $\{i, j\}$, which are exactly the optimal outputs.

Jäger's strategy makes the transducer approach sufficiently flexible to handle

the majority of OT-style constraints, and it is applicable to RCs as we will see later. There is an important restriction, though. As Jäger shows, representing a ranking as a transduction yields the correct result only if the OT grammar satisfies *global optimality*: if o is an optimal output candidate for input i, then it is an optimal candidate for every input that it is an output candidate for. This condition is clearly violated by faithfulness constraints, which evaluate how closely a candidate matches a given input. But even output markedness constraints, which pay no attention to the input, can be problematic.

Graf (2010b, 2012d) investigates this issue with respect to RCs. For RCs, the input plays no role beyond determining the reference set. The actual economy metric evaluates trees only with respect to their own shape and thus can be regarded as an output markedness constraint in OT terms. This entails that optimality is global with respect to reference sets: if two trees have the same reference-set, then the optimal trees in said reference set are identical for both of them. Moreover, RCs satisfy a property Graf calls *output join preservation*.[1] An RC preserves output joins iff it holds for all trees s and t whose reference sets overlap that there is another tree u whose reference set is a (not necessarily proper) superset of the union of the reference sets of s and t. Output join preservation holds of RCs because trees are always members of their own reference set. So if two reference sets overlap, then the trees in their intersection belong to both reference sets, wherefore their reference set must subsume the union of the two. Graf shows that output join preservation and global optimality with respect to reference sets jointly imply global optimality. Consequently, Jäger's strategy of defining transductions via rankings can be used without reservations for modeling RCs.

In order to demonstrate the viability of transducers as a model of RCs, I implement three popular transderivational constraints in the next few sections. In

[1] Actually, Graf refers to this as output join*t* preservation, but I know from a reliable source that this is not the term he had in mind.

the spirit of the OT approach, I present RCs in a modular fashion as a cascade of linear transducers that each realize a small part of the mapping from suboptimal derivations to optimal ones. Remember that this is a viable strategy because the composition of two linear transducers it itself a linear transducer (Engelfriet 1975). Defining RCs in a piece-wise manner we can break them into small, easily understood parts that can then be recombined into one big transducer representing the constraint as a whole. Note that this is just a matter of convenience, one could just as well directly define the complete transducer, but the definition would be a lot more complex. It is also worth keeping in mind that there are infinitely many ways to split a transducer into a sequence of smaller ones, so the transducers I describe are just one of many equivalent ways of defining the transductions induced by the respective RCs.

4.2 Example 1: Focus Economy

4.2.1 Focus Economy Explained

Our first example is Focus Economy (Szendrői 2001; Reinhart 2006), which was briefly discussed in the introduction. This constraint is an example of an RC operating at the interfaces, and as such it differs from syntactic RCs in that it does not filter out existing derivations but rather restricts the interface structures a derivation may be mapped to. Technically I will treat Focus Economy as a mapping from trees to interface structures annotated for stress and focus. Focus Economy makes for a good first example for two reasons. First, modeling it as a transducer is very natural considering that it actually involves a change in structure rather than just the removal of illicit trees. Second, the structural manipulations it carries out are very simple and involve only node relabelings.

Focus Economy has been invoked in order to account for the fact that sentences such as (12a), (12b) and (12c) below differ with respect to what is given and what

is new information. Once again main stress is marked by **boldface**.

(12) a. My friend Paul bought a new **car**.

b. My friend Paul **bought** a new car.

c. My friend Paul bought a **new** car.

That these utterances are associated to different information structures is witnessed by the following (in)felicity judgments. For the reader's convenience, the focus, i.e. the new discourse material introduced by each answer, is put in square brackets.

(13) What happened?

a. [$_F$My friend Paul bought a red **car**.]

b. # [$_F$My friend Paul **bought** a red car.]

c. # [$_f$My friend Paul bought a **red** car.]

(14) What did your friend Paul do?

a. He [$_F$ bought a red **car**].

b. # He [$_F$ **bought** a red car].

c. # He [$_F$ bought a **red** car].

(15) What did your friend Paul buy?

a. He bought [$_F$ a red **car**].

b. # He **bought** [$_F$ a red car].

c. # He bought [$_F$ a **red** car].

(16) Did your friend Paul sell a red car?

a. # No, he [$_F$ bought] a red **car**.

b. No, he [$_F$ **bought**] a red car.

c. # No, he [$_F$ bought] a **red** car.

(17) Did your friend Paul buy a green car?

a. # No, he bought a [$_F$ red] **car**.

b. # No, he **bought** a [$_F$ red] car.

c. He bought a [$_F$ **red**] car.

Restricting our attention to the a-sentences only, we might conclude that a constituent can be focused just in case one of its subconstituents carries sentential main stress. A short glimpse at the b- and c-utterances falsifies this conjecture, though. Perhaps, then, main stress has to fall on the subconstituent at the right edge of the focused constituent? This is easily shown to be wrong, too. In (18) below, the stressed constituent isn't located at either edge of the focused constituent.

(18) a. What happened to Mary?

b. [$_F$ John **killed** her.]

The full-blown Focus Economy system (rather than the simplified sketch given in the introduction) accounts for the data as follows. First, the *Main Stress Rule* demands that in every pair of sister nodes, the "syntactically more embedded" node (Reinhart 2006:p.133) is assigned strong stress, its sister weak stress (marked in the phrase structure tree by subscripted S and W, respectively). If a node has no sister, it is always assigned strong stress (over Minimalist phrase structure trees, this will be the case only for the root node, as all Minimalist trees are strictly binary branching). Main stress then falls on the unique leaf node that is connected to the root node by a path of nodes that have an S-subscript. See Fig. 4.2 for an example.

The notion of being syntactically more embedded isn't explicitly defined in the literature. It is stated in passing, though, that "...main stress falls on the most embedded constituent on the recursive side of the tree" (Reinhart 2006:p.133). While this is rather vague, it is presumably meant to convey that, at least for English, in which complements follow the heads they are introduced by, the right sister node is assigned strong stress as long as it isn't an adjunct. This interpretation seems to be in line with the empirical facts.

The second integral part of the proposal is the operation *Stress Shift*, which shifts

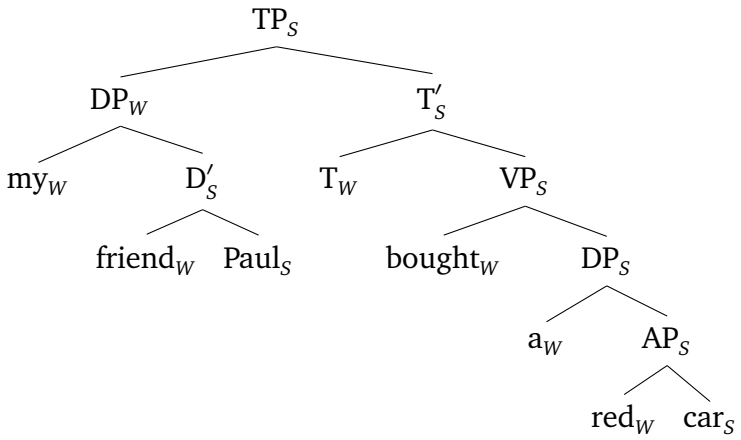


Figure 4.2: Stress-annotated phrase structure tree for 'My friend Paul bought a new **car**'.

the main stress to some leaf node n by assigning all nodes on the path from n to the root strong stress and demoting the sisters of these nodes to weakly stressed nodes. For instance, the tree for "My **friend** Paul bought a new red car" is obtained from the tree in Fig. 4.2 by changing friend$_W$ and Paul$_S$ to friend$_S$ and Paul$_W$, respectively, and DP$_W$ and T$'_S$ to DP$_S$ and T$'_W$, respectively.

While Stress Shift could be invoked to move stress from anaphoric elements to their left sister as in (18), this burden is put on a separate rule, for independent reasons. The rule in question is called *Anaphoric Destressing* and obligatorily assigns weak stress to anaphoric nodes, where a node is anaphoric iff it is "...D[iscourse]-linked to an accessible discourse entity" (Reinhart 2006:p.147). Thus Anaphoric Destressing not only accounts for the unstressed anaphor in (18), but also for the special behavior of stress in cases of parallelism.

(19) First Paul bought a red **car**.

a. Then **John** bought one.

b. * Then John bought **one**.

The overall system now works as follows. Given a phrase structure tree that has not been annotated for stress yet, one first applies Anaphoric Destressing to make

sure that all d-linked constituents are assigned weak stress and thus cannot carry main stress. Next the Main Stress Rule is invoked to assign every node in the tree either W or S. Note that the Main Stress Rule cannot overwrite previously assigned labels, so if some node n has been labeled W by Anaphoric Destressing, the Main Stress Rule has to assign S to the sister of n. Now that the tree is fully annotated, we compute its *focus set*, the set of constituents that may be focused.

(20) *Focus Projection*

Given some stress-annotated tree t, its focus set is the set of nodes reflexively dominating the node carrying main stress.

The focus set of "Paul bought a red **car**", for instance, contains [car], [$_{\text{AP}}$ red car], [$_{\text{DP}}$ a red car], [$_{\text{VP}}$ bought a red car] and [$_{\text{TP}}$ Paul bought a red car] (equivalently, we could associate every node in the tree with a unique address and simply use these addresses in the focus set). For "Then **John** bought one", on the other hand, it consists only of [John] and [$_{\text{TP}}$ Then John bought one].

At this point, Stress Shift may optionally take place. After the main stress has been shifted, however, the focus set has to be computed all over again, and this time the procedure involves reference-set computation.

(21) *Focus Projection Redux*

Given some stress-annotated tree t' that was obtained from tree t by Stress Shift, the focus set of t' contains all the nodes reflexively dominating the node carrying main stress which aren't already contained in the focus set of t.

So if "Then **John** bought one" had been obtained by Stress Shift from [$_{\text{TP}}$ Then John bought one] rather than Anaphoric Destressing, its focus set would have contained only [John], because [$_{\text{TP}}$ Then John bought one] already belongs to the focus set of "Then John bought **one**". As an easy exercise, the reader may want to draw annotated trees for the examples in (12) and compute their focus sets.

4.2.2 A Model of Focus Economy

In order to precisely model Focus Economy, I have to make some simplifying assumptions. First, I stipulate that every adjunct (but not the extended projection of the phrase it adjoins to) is explicitly marked as such by a subscript A on its label. This is simply a matter of convenience, as it reduces the complexity of the transducers and makes my model independent from the theoretical status of adjuncts in MGs. No matter how adjunction is implemented, if it preserves the regularity of MDTLs then tree transducers can correctly distinguish adjuncts from arguments. The subscript A simply abstracts away from this extraneous factor.

Second, I only consider MGs without Move for now. This is not due to some technical limitations — keep in mind that MDTLs are regular with or without Move. Rather, the interaction of focus and movement is not touched upon in Reinhart (2006), so there is no original material for me to formalize. Incidentally, movement seems to introduce several interesting complications, as illustrated by sentences involving topicalization, where no other assignment of focus and main stress is grammatical.

(22) a. $[_F$ **John**$_i]$ Paul likes t$_i$.

b. * John$_i$ $[_F$**Paul**] likes t$_i$.

c. * John$_i$ $[_F$ Paul **likes** t$_i]$.

At the end of the section I explain how the model can be extended to capture theories involving movement.

The last simplification concerns Anaphoric Destressing itself. While the core of Anaphoric Destressing, the destressing of pronominal (and possibly covert) elements, is easy to accommodate, the more general notion of d-linking is impossible to capture in any model that operates on isolated syntactic trees. This is a consequence of the limitation to purely structural reasoning mentioned in Sec. 4.1.1, which makes

it impossible to accommodate discourse factors. However, the role of d-linking in anaphoric destressing is of little importance here, as our focus is firmly on the reference-set computational aspects of Focus Economy. This aspect is completely independent of anaphoric destressing. That is to say, Focus Economy need not decide which elements may be destressed, it only needs to know which ones are, and this information can be supplied by some independent mechanism. Hence my implementation will allow almost any constituent to be anaphorically distressed and leave the task of matching trees to appropriate discourse contexts to an external theory of d-linking that remains to be specified.

With these provisions made explicit, we can finally turn to the formalization of Focus Economy in terms of transducers. The input language I is a set of trees generated by some movement-free MG $\mathscr{E}$ for English. Remember that for movement-free MGs, derivations and phrase structure trees differ only in their labels, so it does not matter what kind of trees are fed to the transducer. This also entails that no matter whether I contains derivations or phrase structure trees, it is a regular tree language.

The reference-set for a given input tree is computed by the composition of four linear transducers corresponding to Anaphoric Distressing, the Main Stress Rule, Stress Shift, and Focus Projection, respectively. Given a tree t derived by $\mathscr{E}$, the transducer cascade computes all logically possible variants of t with respect to stress assignment and then computes the focus in a local way. Note that Focus Economy does not apply during this stage, so no reference-set comparison is carried out yet. Consequently, the reference set of an input tree contains all trees that are correctly annotated for stress and involve the focusing of a constituent that contains the carrier of main stress. This leads to overgeneration with respect to focus, of course, a problem that is later taken care of by another transduction that enforces the Focus Economy rule.

Anaphoric Distressing is modeled by a non-deterministic ltdtt that may randomly

add a subscript D to a node's label in order to mark it as anaphoric. The only condition is that if a node is labeled as anaphoric, all the nodes it properly dominates must be marked as such, too. In a more elaborate model, the D subscripts would be supplied by a theory of discourse.

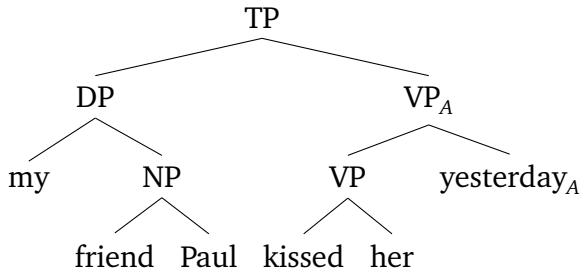
Definition 4.4. Let $\Sigma := \Sigma_L \cup \Sigma_A$ be the vocabulary of the MG $\mathscr{E}$ that generated the input language I, where Σ_L contains all node labels and Σ_A their counterparts explicitly labeled as adjuncts. *Anaphoric Destressing* is the ltdtt $\mathscr{D}$ where $\Sigma_{\mathscr{D}} := \Sigma$, $\Omega_{\mathscr{D}}$ is the union of Σ and $\Sigma_D := \{\sigma_D \mid \sigma \in \Sigma\}$, $Q := \{q_i, q_d\}$, $F := \{q_i\}$, and $\Delta_{\mathscr{D}}$ contains the rules below, with $\sigma \in \Sigma$ and $\sigma_D \in \Sigma_D$:

$$q_i(\sigma(x,y)) \to \sigma(q_i(x), q_i(y)) \qquad q_i(\sigma) \to \sigma$$
$$q_{\{i,d\}}(\sigma(x,y)) \to \sigma_D(q_d(x), q_d(y)) \qquad q_{\{i,d\}}(\sigma) \to \sigma_D$$

Although this definition looks intimidating, it describes a very simple transducer that either leaves a node unchanged or adds a subscript D. The state q_i is the default state, indicating that no dominating nodes have been altered so far. Once the transducer decides to add a subscript, though, it changes into the state q_d and must add the same subscript to all nodes it encounters until it reaches a leaf.

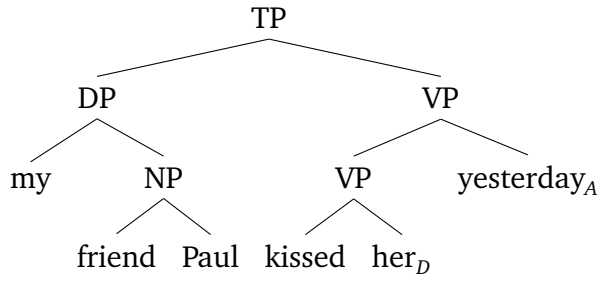
Example 4.5 A transduction for anaphoric destressing

Suppose the following phrase structure tree is fed as input to *Anaphoric Destressing*.



Among the many output trees created by the transducer, we also find the linguistically

plausible one in which the anaphor *her* is marked as destressed.



The transducer for the Main Stress Rule is non-deterministic, too, but it proceeds in a bottom-up manner. It does not alter nodes subscripted by A or D, but if it encounters a leaf node without a subscript, it randomly adds the subscript S or W to its label. However, W is allowed to occur only on left sisters, whereas S is mostly restricted to right sisters and may surface on a left sister just in case the right sister is already marked by A or D. Note that we could easily define a different stress pattern, maybe even parametrized with respect to category features, to incorporate stress assignment rules from other languages.

Definition 4.5. *Main Stress* is the lbutt $\mathscr{M}$ where $\Sigma_{\mathscr{M}} := \Omega_{\mathscr{D}}$, $\Omega_{\mathscr{M}}$ is the union of Σ, Σ_D and $\Sigma_* := \{\sigma_S, \sigma_W \mid \sigma \in \Sigma\}$, $Q := \{q_s, q_u, q_w\}$, $F := \{q_s\}$ and $\Delta_{\mathscr{M}}$ contains the following rules, with $\sigma \in \Sigma$, $\sigma_A \in \Sigma_A$, $\sigma_x \in \{\sigma_x \mid \sigma \in \Sigma\}$ for $x \in \{D, S, W\}$:

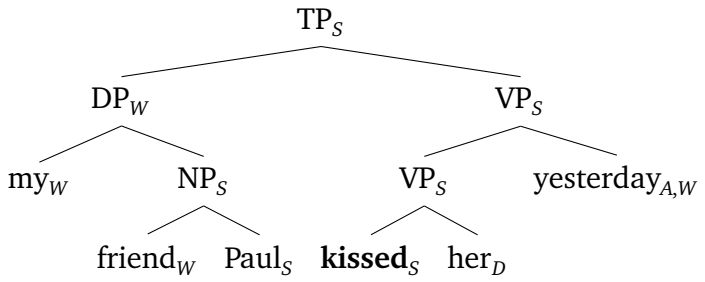
$$\sigma_A \to q_u(\sigma_A) \qquad \sigma_A(q_u(x), q_u(y)) \to q_u(\sigma_A(x, y))$$
$$\sigma_D \to q_u(\sigma_D) \qquad \sigma_D(q_u(x), q_u(y)) \to q_u(\sigma_D(x, y))$$
$$\sigma \to q_{sw}(\sigma_{SW}) \qquad \sigma(q_{\{u,w\}}(x), q_s(y)) \to q_{sw}(\sigma_{SW}(x, y))$$
$$\sigma(q_s(x), q_u(y)) \to q_{sw}(\sigma_{SW}(x, y))$$

It is crucial in the definition of *Main Stress* that the transition rules do not cover

all logically possible combinations of node labels and states. For example, there is not transition if both states on the left-hand side are q_s. This ensures that no stress patterns are generated where two sister nodes are both marked with S. Similarly, there is no rule where the right daughter of a node is q_w, so that no right daughter can ever be subscripted with W, and Should such configurations arise, the transducer aborts the construction of the output tree and starts with a new one. Due to this, the transducer generates exactly one output tree for every input, even though it is non-deterministic. This is the desired behavior, as for every tree there should be exactly one default stress.

Example 4.6 A transduction for the main stress rule

The phrase structure tree from the previous example is rewritten by *Main Stress* to yield the default prosody with main stress on *kissed*.



Stress Shift, in turn, is implemented as a non-deterministic ltdtt that may randomly switch the subscripts of two S/W-annotated sisters.

Definition 4.6. *Stress Shift* is the ltdtt $\mathscr{S}$ where $\Sigma_{\mathscr{S}} = \Omega_{\mathscr{S}} = \Omega_{\mathscr{M}}$, $Q := \{q_i, q_s, q_w\}$, $F := \{q_s\}$, and $\Delta_{\mathscr{S}}$ contains the rules below, with $\sigma \in \Sigma_{\mathscr{S}}$ and $\sigma_* \in \Sigma_* :=$

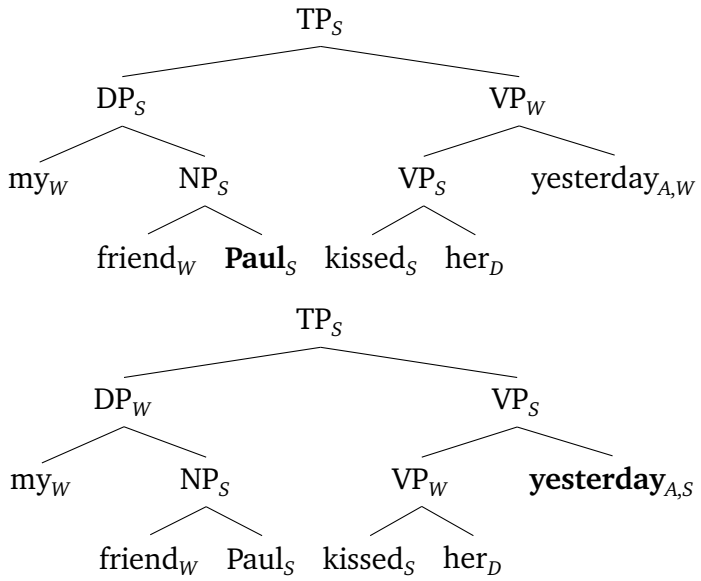
$\{\sigma_S, \sigma_W \mid \sigma \in \Sigma\}$:

$$q_s(\sigma_*(x,y)) \to \sigma_{SSS}(q_{isw}(x), q_{iws}(y)) \qquad q_s(\sigma_*) \to \sigma_S$$
$$q_w(\sigma_*(x,y)) \to \sigma_W(q_i(x), q_i(y)) \qquad q_w(\sigma_*) \to \sigma_W$$
$$q_i(\sigma(x,y)) \to \sigma(q_i(x), q_i(y)) \qquad q_i(\sigma) \to \sigma$$

This transducer non-deterministically assigns each daughter one of three states. If the state is q_i, the label of the daughter is not altered in the next rewrite step. If it is q_S or q_W, the subscript is changed to S or W, respectively. Notice that only three combinations of states can be assigned to the daughters: q_i and q_i (nothing changes), q_S and q_W, or q_W and q_S. This ensures that stress shift only creates well-formed stress patterns where no two daughters have the same stress subscript.

Example 4.7 A transduction for the stress shift rule

Among the many outputs produced by Stress Shift for our example tree, we find the ones for "My friend **Paul** kissed her yesterday" and "My friend Paul kissed her **yesterday**", as well as the example tree itself (Stress Shift need not change any subscripts at all).



The last component of the transduction computing the reference set for Focus Economy is Focus Projection. Focus Projection is formalized as a non-deterministic ltdtt with two states, q_f and q_g. The transducer starts at the root in q_f. Whenever a node n is subscripted by W, the transducer switches into q_g at this node and stays in the state for all nodes dominated by n. As long as the transducer is in q_f, it may randomly add a superscript F to a label to indicate that it is focused. Right afterward, it changes into q_g and never leaves this state again. Rather than associating a stress-annotated tree with a set of constituents that can be focused, Focus Projection now generates multiple trees that differ only with respect to which constituent along the path of S-labeled nodes is focus-marked.

Definition 4.7. *Focus Projection* is the ltdtt $\mathscr{F}$, where $\Sigma_{\mathscr{F}} = \Omega_{\mathscr{S}}$, $\Omega_{\mathscr{F}}$ is the union of $\Omega_{\mathscr{S}}$ and $\Omega^F_{\mathscr{S}} := \{\omega^F \mid \omega \in \Omega_{\mathscr{S}}\}$, $Q := \{q_f, q_g\}$, $F := \{q_f\}$, and $\Delta_{\mathscr{F}}$ contains the rules below, with $\sigma \in \Sigma_{\mathscr{F}}$ and $\sigma_{\overline{S}} \in \Sigma_{\mathscr{F}} \setminus \{\sigma_S \mid \sigma \in \Sigma\}$:

$$
\begin{aligned}
q_f(\sigma_S(x,y)) &\to \sigma_S(q_f(x), q_f(y)) & & \\
q_f(\sigma_S(x,y)) &\to \sigma^F_S(q_g(x), q_g(x)) & q_f(\sigma_S) &\to \sigma^F_S \\
q_f(\sigma_{\overline{S}}(x,y)) &\to \sigma_{\overline{S}}(q_g(x), q_g(x)) & q_f(\sigma_{\overline{S}}) &\to \sigma_{\overline{S}} \\
q_g(\sigma(x,y)) &\to \sigma(q_g(x), q_g(y)) & q_g(\sigma) &\to \sigma
\end{aligned}
$$

Example 4.8 A transduction for focus projection

Four nodes are viable hosts for the F-subscript in the tree for "My friend **Paul** kissed her yesterday": TP, DP, NP, and *Paul*. No other node is available because the subscript may be added only as long as the transducer is in state q_f, which it leaves as soon as

it encounters a node with subscript W. For the same reason only TP, the higher VP projection, or *yesterday* may carry the subscript in the tree for "My friend Paul kissed her **yesterday**".

All four transducers are clearly linear since they only relabel nodes and do not manipulate any structure, let alone copy subtrees. The ltdtt can be automatically converted into lbutts, so that one gets four lbutts that can then be composed into one big lbutt *RefSet* that rewrites input trees into stress-annotated and focus-marked output trees. More precisely, the reference set computed by this transducer contains all variants of a tree t such that I) some subtrees may be marked as adjuncts or anaphorical material (or both) and thus do not carry stress information, II) there is exactly one path from the root to some leaf such that every node in the path is labeled by S, and III), exactly one node belonging to this path is marked as focused.

Now it only remains for us to implement *Focus Projection Redux*. In the original account, Focus Projection Redux applied directly to the output of Stress Shift, i.e. trees without focus information, and the task at hand was to assign the correct focus. In my system, on the other hand, every tree is fed into Focus Projection and marked accordingly for focus. This leads to overgeneration for trees in which Stress Shift has taken place — a node may carry focus even if it could also do so in the tree without shifted main stress. Consequently, the focus set of "My friend **Paul** kissed her yesterday", for instance, contains $[_{\text{PN}}$ Paul], $[_{\text{NP}}$ friend Paul], $[_{\text{DP}}$ my friend Paul], and the entire DP. Only the first three are licit foci, though, because the TP can already be focused with the default stress assignment. Hence the transduction system devised in this section confronts Focus Projection Redux with the burden of filtering out focus information instead of assigning it. In other words, Focus Projection Redux is a constraint.

The constraint Focus Projection Redux can be attached to the reference-set

transduction through a mathematical trick. As we will see in a moment, Focus Projection Redux can be specified as an MSO formula (see Sec. 3.1.3), wherefore it defines a regular tree language. Every regular tree language L can be turned into a transduction τ by taking its diagonal: for every tree t, its image under τ is t if $t \in L$ and undefined otherwise. So the diagonal of a tree language corresponds to a transducer that rewrites every tree in the language by itself and aborts on all trees that are not contained in the language. This implies that composing *RefSet* with the diagonal of the language defined by Focus Projection Redux filters out all output trees created by the former that do not obey the latter. Concretely, a tree with shifted stress is filtered out if the F subscript is attached to a node that could also be focused under default stress.

The MSO definition of Focus Projection Redux relies on two predicates, *StressPath* and *FocusPath*. The former picks out the path from the root to the leaf carrying main stress, whereas the latter refers to the path from the root to the leaf that would carry main stress in the absence of stress shift. Hence *FocusPath* replicates some of the information that is already encoded in the Main Stress transducer, but there is no need to worry about this redundancy here.

In the formulas below, $A(x)$, $D(x)$ and $S(x)$ are predicates picking out all nodes with subscript A, D, S, respectively. These are simply convenient shorthands for disjunctions; for example, $A(x)$ might stand for $\mathrm{TP}_A(x) \vee \mathrm{VP}_A(x) \vee \mathrm{DP}_A(x) \vee \mathrm{NP}_A(x)$. Moreover, $x \triangleleft y$ holds iff x is the parent of y, $x \prec y$ iff x is the left sibling of y, and $x \triangleleft^* y$ iff x dominates y or the two are the same node. The predicate $\mathrm{Path}(X)$ holds of a set X of nodes iff it contains the root of the tree, exactly one leaf, and "has no holes", that is to say, if x dominates y and both belong to X, then so does every

node between the two.

$$\text{Path}(X) \iff \exists r\, \exists!l\, \forall z \Big[X(r) \wedge X(l) \wedge \neg \exists x \big[u \lhd r \vee l \lhd u \big] \wedge \big(r \lhd^* z \wedge z \lhd^* l \leftrightarrow X(z) \big) \Big]$$

The Stress Path is simply the unique path in which every node is subscripted with S. The leaf of this path is the LI carrying primary stress.

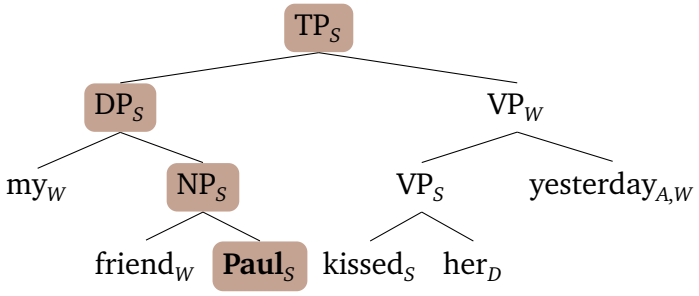
$$\text{StressPath}(X) \iff \text{Path}(X) \wedge \forall x [X(x) \rightarrow S(x)]$$

The Focus Path, on the other hand, extends from the root to the leaf that would carry default stress. Suppose x is in the path and has daughters y and z. Only one of the two can be in the path. If y is an adjunct or destressed — indicated by the subscripts A and D, respectively — then z belongs to the Focus Path irrespective of the linear order of y and z. In all other cases, the right daughter belongs to the path.

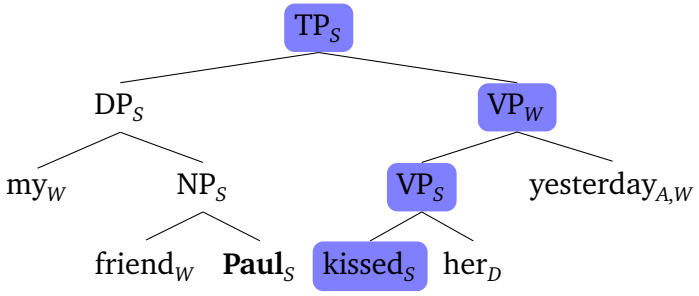
$$\text{FocusPath}(X) \iff \text{Path}(X) \wedge \forall x\, \forall y\, \forall z \Big[X(x) \wedge x \lhd y \wedge x \lhd z \rightarrow \big(A(y) \vee D(y) \rightarrow X(z) \big) \wedge \big(\neg A(y) \wedge \neg D(y) \wedge y \prec z \rightarrow X(z) \big) \Big]$$

Example 4.9 Stress Paths and Focus Paths

The Stress Path in "My friend **Paul** kissed her yesterday" extends from the root towards *Paul*.



Its Focus Path, on the other hand, is identical to the Stress Path of "My friend Paul **kissed** her", the default prosody. Thus it extends from the root to *kissed*. The two paths overlap only at the root node.



In a tree where no stress shift has taken place, StressPath and FocusPath are true of the same subsets and any node contained by them may be focused. After an application of the Stress Shift rule, however, the two paths are no longer identical, although their intersection is never empty (it has to contain at least the root node). In this case, then, the only valid targets for focus are those nodes of the Stress Path that are not contained in the Focus Path. Or the other way round, a node belonging to both the Stress Path and the Focus path may be focused only if the two are identical. This is formally expressed by the MSO sentence ϕ below. Just like $A(x)$, $D(x)$ and $S(x)$ before, $F(x)$ is a predicate defining a particular set of nodes, this time the set of nodes with subscript F. I furthermore use $X \approx Y$ as a shorthand

for $\forall x[X(x) \leftrightarrow Y(x)]$.

$$\phi := \forall x\, \forall X\, \forall Y\, [F(x) \wedge X(x) \wedge Y(x) \wedge \text{StressPath}(X) \wedge \text{FocusPath}(Y) \rightarrow X \approx Y]$$

Note that ϕ by itself does not properly restrict the distribution of focus. First of all, there is no requirement that exactly one node must be focused. Second, nodes outside StressPath may carry focus, in which case no restrictions apply to them at all. Finally, both the Stress Path and the Focus Path may be empty, because we have not made any assumptions about the distribution of labels. Crucially, though, ϕ behaves as expected over the trees produced by the transducer *RefSet*. Thus taking the diagonal of the language licensed by ϕ and composing it with *RefSet* filters out all trees with illicit foci, and only those. Since the diagonal over a regular language is a linear transduction, the transduction obtained by the composition is too. This establishes the computational feasibility of Focus Economy when the input language is a regular tree language.

So far I have left open the question, though, how movement fits into the picture. First of all, it cannot be ruled out *a priori* that the interaction of movement and focus are so intricate on a linguistic level that significant modifications have to be made to the original version of Focus Economy. On a formal level, this would mean that the transduction itself would have to be changed. In this case, it makes little sense to speculate how my model could be extended to accommodate movement, so let us instead assume that Focus Economy can remain virtually unaltered and it is only the input language that has to be modified. If we want the full expressive power of MGs, then the best strategy is to express Focus Economy as a constraint over derivation trees, since every MDTL is regular. As the differences between derivations and their derived trees are very minor, few formal alterations are needed. Of course empirical factors regarding movement might have to be taken into consideration, but it is unlikely that linear tree transducers are too weak to accommodate them.

4.3 Example 2: Merge-over-Move

Another well-known reference-set constraint is Chomsky's Merge-over-Move condition (MOM; Chomsky 1995c, 2000), which is the subject of inquiry in this section. After a short discussion of the mechanics of the constraint and its empirical motivation, I turn to the formal aspects of implementing MOM. In contrast to Focus Economy, where the transduction produces new output trees, MOM is a genuine filter in the sense that it maps a set of trees to a subset of itself. Nonetheless the procedure for devising a MOM transducer exhibits many parallels to Focus Economy.

4.3.1 Merge-over-Move Explained

In comparison to Focus Economy, modeling MOM is slightly more intricate, because there are multiple versions of the constraint, which are seldom carefully teased apart in the literature. Naturally they all share the core idea of MOM: if at some point in a derivation we are allowed to choose between Merge and Move as the next step of the derivation, Merge is preferable to Move. This idea can be used to account for some puzzling contrasts involving expletives (if not indicated otherwise, all examples are taken from Castillo et al. 2009).

(23) a. There seems to be a man in the garden.
b. * There seems a man to be in the garden.
c. A man seems to be in the garden.

In the original formulation of Minimalist syntax, every derivation starts out with a multiset of LIs — the *numeration* — that are enriched with interpretable and uninterpretable features, the latter of which have to be erased by feature checking. Under such a conception, (23a) and (23c) are easy to derive. Let us look at (23c) first. It starts out with the numeration {seems, to, be, a, man, in, the, garden}. Multiple applications of Merge yield the phrase $[_{\text{TP}}$ to be a man in the garden]. At this point,

the Extended Projection Principle (EPP) demands that the specifier of the infinitival TP be filled by some phrase. The only item left in the numeration is *seems*, which cannot be merged in SpecTP. Hence we are stuck with moving the DP *a man* into SpecTP, yielding $[_{TP}$ a man to be t_{DP} in the garden]. Afterwards, the TP is merged with *seems* and the DP is once again moved, this time into the specifier of *seems* to check the case feature of the DP and satisfy the EPP.

For (23a), however, things are slightly different. Here the numeration initially consists of {there, seems, to, be, a, man, in, the, garden}. Once again we start out by merging items from the numeration until we arrive at $[_{TP}$ to be $[_{DP}$ a man in the garden]]. But now we have two options: Merger of *there*, which is later followed by moving *there* into the specifier of *seems*, thus yielding the grammatical (23a), or first moving *a man* into the specifier of *to be* and subsequently merging *there* with *seems a man to be in the garden*, which incorrectly produces the ungrammatical (23b). MOM rectifies this overgeneration problem by barring movement of *a man* into the specifier of *to be*, as the more economical route of merging *there* in this position is available to us. At the same time, MOM does not block (23c) because we aren't given a choice between Merge and Move at any point of its derivation.

Different versions of MOM emerge depending on the setting of two binary parameters:

P1 Reference set algorithm: *indiscriminate*/*cautious*

Indiscriminate versions of MOM (iMOM) pick the most economical derivation even if it derives an ungrammatical phrase structure tree — such derivations are said to *crash*. Cautious versions of MOM (cMOM), on the other hand, choose the most economical derivation that yields a well-formed tree.

P2 Mode of application: *sequential*/*output*

Sequential versions of MOM (sMOM) check for MOM violations after every step of the derivation. Thus early violations of MOM carry a significantly

greater penalty than later ones.MOM applied to the output (oMOM), however, is sensitive only to the total number of violations, not their timing. So if derivation d incurs only one violation of MOM, which occurs at step 4 in the derivation, while derivation d' incurs seven, starting at step 5, then d will win against d' under an output filter interpretation of MOM and lose under a sequential one.

Combining the parameters in all logically possible ways (*modulo* underspecification) yields the four variants isMOM, csMOM, ioMOM and coMOM. All four of them supposedly use the *Identity of Numerations Condition* (INC) for computing reference sets, according to which the reference set of a derivation d contains all the derivations that can be built from the same numeration as d."Supposedly", because only the sMOM variants have been discussed at length in the literature. The original proposal by Chomsky (1995c) is what I call csMOM. But the global flavor of csMOM with its reliance on derivational look-ahead prompted the creation of isMOM as a strictly local alternative. Indeed isMOM can be argued to contain not even a modicum of reference-set computation, as it simply states that if there is a choice between Merge and Move, the former is to be preferred. Whether such a choice exists can always be checked tree-locally.

For simple cases like (23), where we only have to choose once between Merge and Move, all MOM variants produce the same results. But as soon as one encounters examples involving embedded clauses, the predictions diverge.

(24) a. There was [a rumor [that a man was t_{DP} in the room]] in the air.

b. [A rumor [that there was a man in the room]] was t_{DP} in the air.

Both oMOM-versions get the right result: Each sentence prefers Move over Merge exactly once, so assuming that there are no (grammatical) competing derivations that start from the same numeration and incur fewer violations, (24a) and (24b) should both be grammatical. The sMOM variants, on the other hand, struggle with

this data. The sentences are built up from the same numeration, so (24b) should block (24a), since the former violates MOM at a later derivational step than the latter. In order to account for such cases, Chomsky (2000) stratifies numerations into subnumeration such that each CP has its own numeration. In the case at hand, (24a) is built from the numeration {{there, was, a, rumor, in, the, air}, {that, was, a, man, in, the, room}}, and (24b) from the minimally different {{was, a, rumor, in, the, air}, {that, there, was, a, man, in, the, room}}. By the INC, then, derivations built from the former do not belong to the same reference set as derivations built from the latter.

So now we have a third parameter to take into account. Even though it isn't directly related to the makeup of MOM, I will indicate it as a prefix as before.

P3 Application domain: *restricted*/*unbounded*

Restricted versions of MOM (rMOM) are parts of a grammar where every CP has its own numeration. Unbounded versions (uMOM) belong to grammars with one big numeration.

Taking stock, we have csuMOM as the version of MOM introduced in (Chomsky 1995c), isuMOM as its local counterpart, and csrMOM as the modification put forward in (Chomsky 2000). Somewhat surprisingly, no oMOM variants are entertained in the Minimalist literature, despite the small empirical advantage they have displayed so far (but comparable proposals have been put forward by advocates of OT-syntax). For this reason, I shall mostly restrict myself to sMOM variants in the following.

4.3.2 Properties of Merge-over-Move

The defining property of sMOM is that the timing of violations is more important than the number of total violations. The earlier a violation occurs, the less optimal a derivation is with respect to sMOM. Thus the evaluation has to proceed in a bottom

up fashion, weeding out candidates with unnecessary instances of Move after every derivational step. But how exactly should this be accomplished? In the case of Focus Economy, comparing distinct trees was made easy by the fact that the competing trees only differ in their node labels, so the tree with default stress could be inferred from the tree with shifted stress. This made it possible to represent the competing trees within a single tree (using the FocusPath and StressPath predicates) and thus emulate the comparative procedures by well-formedness constraints on this one underlying tree. If we could find a similar way of representing competing derivations within one derivation tree, a major hurdle in the formalization of MOM would be out of the way.

Unfortunately there is yet another problem, and this one pertains to sMOM as well as oMOM, namely the INC. It is impossible for a linear tree transducer to compute the reference-sets defined by the INC. The major culprit here is the restriction to finite memory, which entails that we can only distinguish between a bounded number of occurrences of LIs. For some suitably large n, the multiset M' obtained from $M := \{\text{John}_n, \text{thinks}_n, \text{that}_n, \text{Mary died}\}$ by adding one more occurrence of *John*, *thinks*, and *that* will be indistinguishable from M for the transducer. But the undefinablity of the INC does not necessarily entail the undefinability of MOM.

4.3.3 A Model of MOM

The INC is both too powerful and too weak. Consider (23) again, repeated here for the reader's convenience.

(25) a. There seems to be a man in the garden.

b. * There seems a man to be in the garden.

c. A man seems to be in the garden.

MOM's objective is to explain why (25b) is ungrammatical, and it does so by using a metric that makes it lose out against (25a). The grammaticality of (25c), on

the other hand, follows from the fact that it isn't a member of the same reference set, due to the INC. But identity of numerations is a rather indirect encoding of the relationship that holds between (25a) and (25b). A formally simpler condition emerges upon inspection of their derivation trees (cf. Fig. 4.3). Ignoring the feature specifications of the LIs, we see that the only difference between the respective derivation trees is the timing of move. Rather than a transducer modeling the INC, then, all we need is a transducer that will produce (at least) the derivation trees for (25a) and (25b) when given either as an input. This involves merely changing the position of the unary branch, which is an easy task for a linear transducer. But

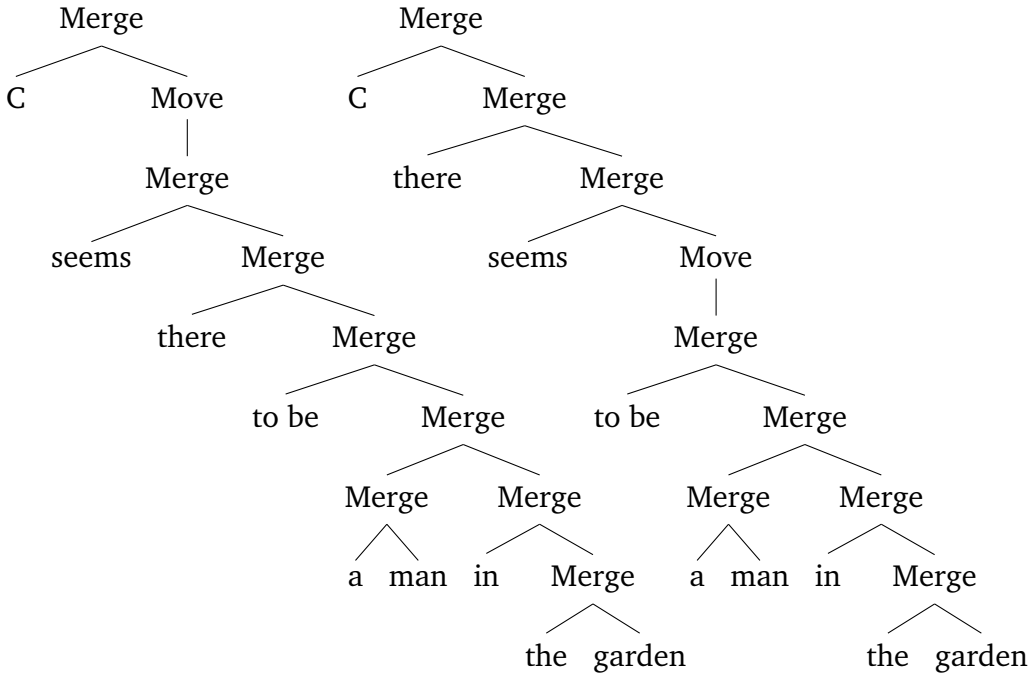


Figure 4.3: Modulo the feature components of their LIs, the derivation trees of (25a) and (25b) differ only in the position of the unary branch

now compare these derivations to the one for (25c) in Fig. 4.4 on the following page. The derivation trees of (25a) and (25b) are essentially the result of non-deterministically replacing one instance of move in the derivation tree of (25c) by merger with expletive *there*. Strikingly, though, rewriting the lower occurrence of Move yields the grammatical (25a), whereas rewriting the structurally higher

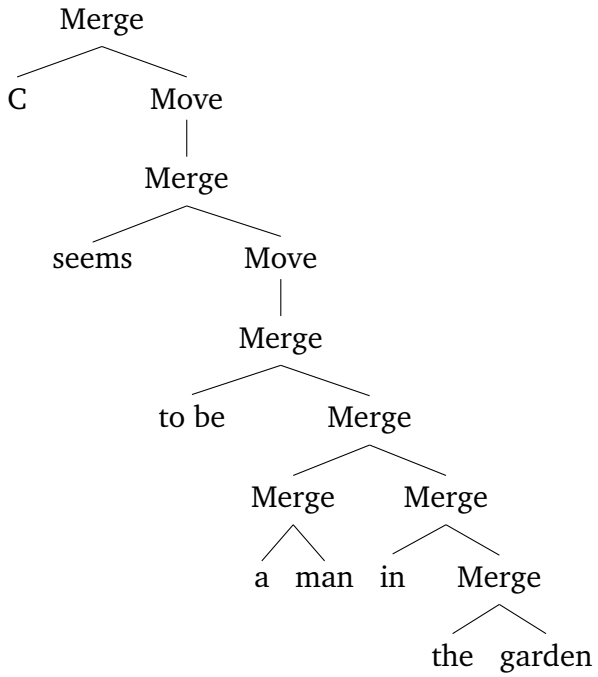


Figure 4.4: The derivation tree of (25c) can be taken as the basis for the previous two

occurrence gives rise to the ungrammatical (25b). Now if we design the transducer such that it won't rewrite Move as a *there*-merger after it has already passed up on an opportunity to do so earlier in the derivation, (25b) cannot be generated from the derivation tree of (25c). In linguistic parlance, this is tantamount to treating MOM as a well-formedness condition on derivation trees (note the similarity to the Focus Economy strategy).

The idea just outlined is captured as follows: First, we take as our input language I some MDTL. Then we use a transducer α to map this language to a set U of underspecified derivation trees. The transducer strips away all features from all LIs. In the TP-domain, it deletes expletive *there* and rewrites Move as the placeholder $\square$. The underspecified representation of (25a)–(25c), for instance, is almost identical to the derivation tree of (25c) except that the two Move nodes are now labeled $\square$ (and their LIs are devoid of any features). These underspecified representations are then turned into fully specified representations again by the transducer β. It

reinstantiates the features on the LIs and non-deterministically rewrites $\square$ as Move or Merger of *there*, but with the added condition that once a $\square$ has been replaced by Move, all remaining instance of $\square$ in the same CP have to be rewritten as Move.

I use J to denote the output language of the transduction realized by β. The name is meant to be a shorthand for *junk*, as J will contain a lot thereof, for two independent reasons. First, the non-deterministic rewriting of $\square$ allows for two occurrences of $\square$ to be rewritten as *there*, which yields the (derivation tree of the) ungrammatical *there seems there to be a man in the garden*. Second, the reinstantiation of the features is a one-to-many map that will produce a plethora of illicit derivation trees as some LIs may not be able to get all their features checked. This overgeneration problem is taken care of by intersecting J with I. All outputs that weren't already part of the input language are thereby removed, as are the inputs that violate MOM.

After these important remarks, let us get started on the low-level implementation of MOM. As mentioned before, I assume that the input language I is an MDTL of some MG $\mathscr{E}$ with lexicon *Lex*. The transduction α is obtained from composing the two transducers *Remove Features* and *Underspecify*.

Definition 4.8. *Remove Features* is the deterministic (one-state) relabeling that maps each $l := \langle \sigma :: f_1, \ldots, f_{base}, \ldots, f_n \rangle \in Lex$ to $l' := \sigma_{f_{base}}$, where f_{base} is the base feature of l. The set of these simplified LIs is denoted by Λ.

As every Minimalist LI has exactly one category feature, *Remove Features* is well-defined. The map defined by *Remove Features* is many-to-one, so Λ is finite by virtue of the finiteness of *Lex*.

Definition 4.9. *Underspecify* is the lbutt $\mathscr{U}$, where $\Sigma_{\mathscr{U}} := \Lambda \cup \{\text{Merge}, \text{Move}\}$, $\Omega_{\mathscr{U}} := \Sigma_{\mathscr{U}} \cup \{\square\}$, $Q := \{q_*, q_c, q_i, q_t\}$, $F := \{q_*\}$, and $\Delta_{\mathscr{U}}$ consists of the rules below, where I use the following notational conventions:

- σ_I (σ_C) denotes any LI $l \in \Lambda$ whose base feature is I (C),
- the symbol "there" refers to any expletive $l \in \Lambda$ involved in MOM (usually just *there*, but possibly also *it*),
- σ_l denotes any LI which does not fall into (at least) one of the categories described above,
- rules for binary branching nodes are stated under the assumption that slices are right-branching (cf. Sec. 1.2.1).

$$\begin{aligned} \sigma_l &\to q_*(\sigma_l) & \text{Merge}(q_{i*}(y), q_{c*}(x)) &\to q_*(\text{Merge}(x,y)) \\ \sigma_I &\to q_i(\sigma_I) & \text{Merge}(q_{\{i,*\}}(y), q_i(x)) &\to q_i(\text{Merge}(x,y)) \\ \text{there} &\to q_t(\text{there}) & \text{Merge}(q_{\{i,*\}}(y), q_t(x)) &\to q_i(\square(y)) \\ \sigma_C &\to q_c(\sigma_C) & \text{Move}(q_*(x)) &\to q_*(\text{Move}(x)) \\ & & \text{Move}(q_i(x)) &\to q_i(\square(x)) \end{aligned}$$

The underspecified derivations have to be turned back into fully specified ones by the transduction β, which is the composition of *Path Condition* and the inverse of *Remove Features*.

Definition 4.10. *Path Condition* is the lbutt $\mathscr{P}$, where $\Sigma_{\mathscr{P}} := \Omega_{\mathscr{U}}$, $\Omega_{\mathscr{P}} := \Sigma_{\mathscr{U}}$, $Q := \{q_*, q_c, q_o\}$, $F := \{q_*\}$, and $\Delta_{\mathscr{P}}$ contains the rules below (the same notational conventions apply):

$$\begin{aligned} \sigma_l &\to q_*(\sigma_l) & \text{Merge}(q_{c\{c,*\}}(x), q_{o\{c,*\}}(y)) &\to q_*(\text{Merge}(x,y)) \\ \sigma_I &\to q_*(\sigma_I) & \text{Merge}(q_{\{*,o\}}(x), q_o(y)) &\to q_o(\text{Merge}(x,y)) \\ \sigma_C &\to q_c(\sigma_C) & \text{Move}(q_*(x)) &\to q_*(\text{Move}(x)) \\ & & \square(q_*(x)) &\to q_*(\text{Merge}(\text{there}, x)) \\ & & \square(q_{\{o,*\}}(x)) &\to q_o(\text{Move}(x)) \end{aligned}$$

The crucial step toward capturing MOM is the last rule of *Path Condition*, which tells the transducer that after it has rewritten one instance of □ as Move, it has to switch into state q_o, which tells it to always rewrite □ as Move. Only if it encounters a node of category C may the transducer switch back into its normal state q_* again. Figures 4.5 and 4.6 show how *Path Condition* turns the underspecified derivation back into the derivations for *There seems to be man in the garden* and *A man seems to be in the garden* (before the features are added back in).

4.3.4 Empirical Evaluation

As discussed above, the transducer model of MOM accounts for simple expletive/non-expletive alternations as in (25). Instead of going through another iteration of the same basic argument, let us look at a more complex example that we have encountered before, repeated here as (26).

(26) a. There was [a rumor [that a man was t_{DP} in the room]] in the air.

b. [A rumor [that there was a man in the room]] was t_{DP} in the air.

Recall that this was a problematic case for pre-Chomsky (2000) versions of MOM (i.e. csuMOM and isuMOM), because in the absence of stratified numerations the INC puts (26a) and (26b) in the same reference set, where they have to compete against each other. Under a sequential construal of MOM, then, (26a) will block (26b) as it opts for Merge rather than Move at the first opportunity.

Under the transducer conception of MOM (tMOM), on the other hand, (26) is a straightforward generalization of the pattern in (25). The underspecified derivation tree of both sentences is shown in Fig.4.7. When the underspecified derivation is expanded to full derivations again, all four logical possibilities are available: *there*-insertion in both CPs, Move in both CPs, *there*-insertion in the lower CP and Move in the higher one, and Move in the lower CP and *there*-insertion in the higher

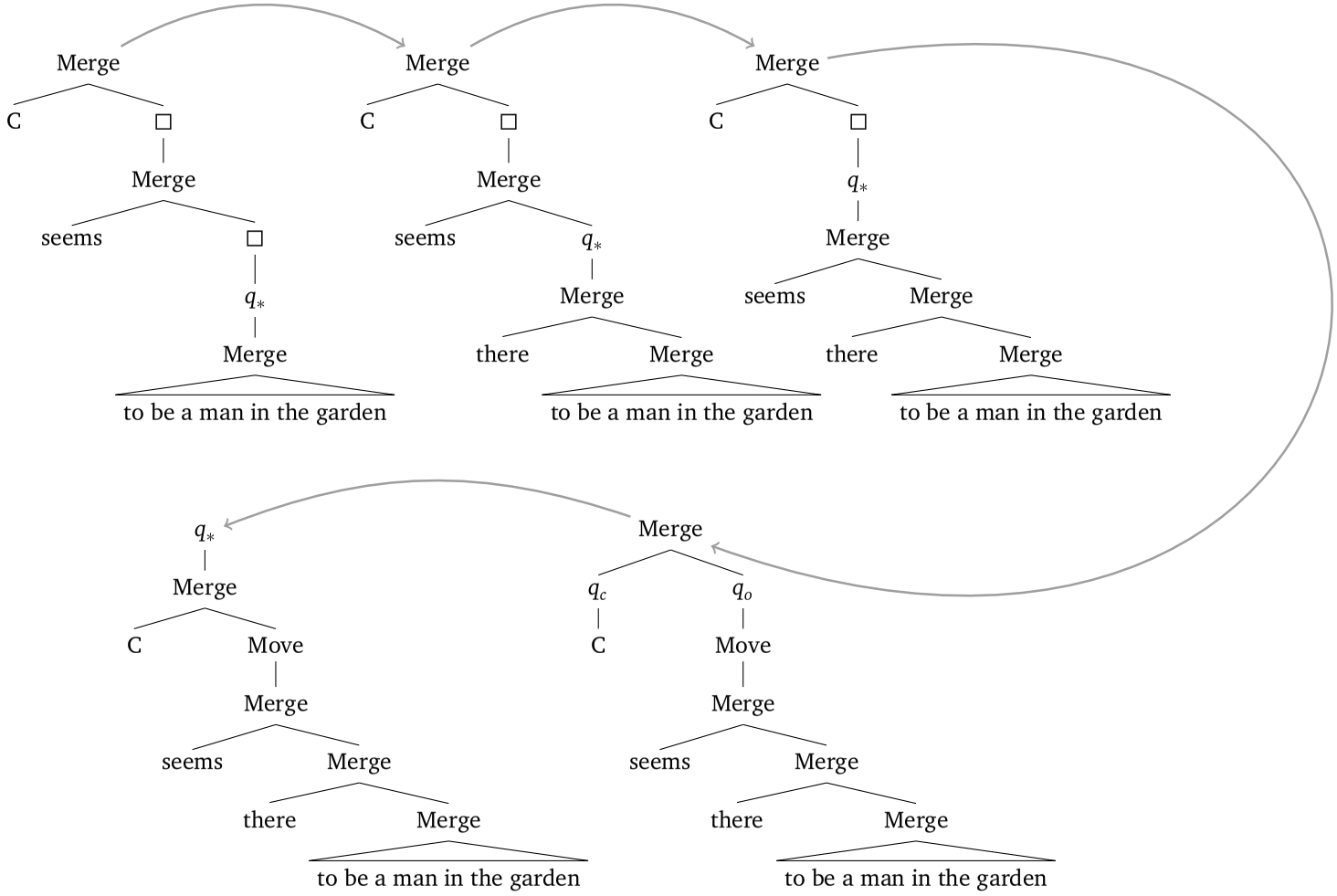


Figure 4.5: Rewriting the underspecified derivation tree into the derivation tree of *There seems to be a man in the garden*

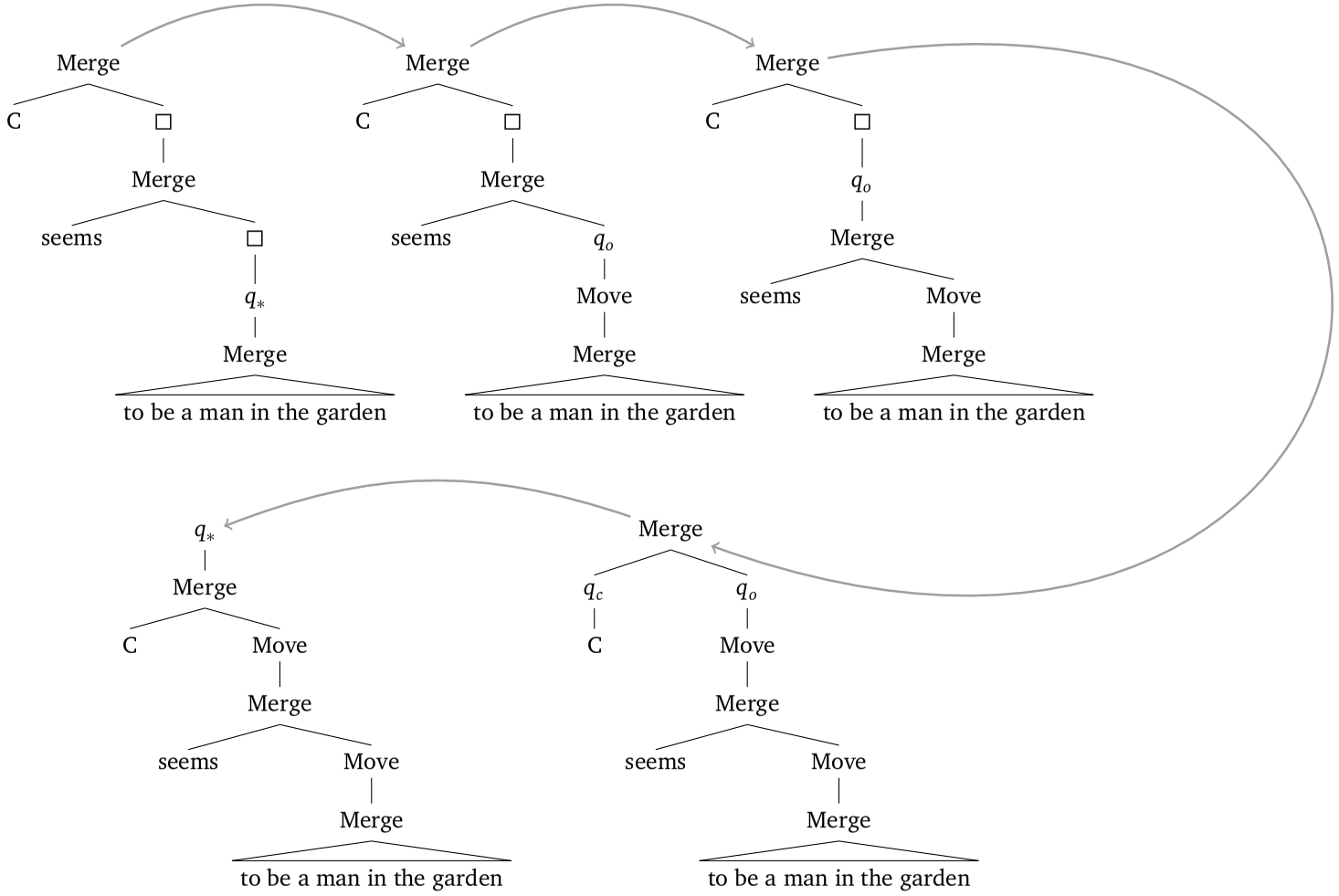


Figure 4.6: Rewriting the underspecified derivation tree into the derivation tree of *A man seems to be in the garden*

one. The last option is available because the transducer, which is in the "rewrite all instances of □ as Move"-state q_o after rewriting □ as Move, switches back into the neutral state q_* after encountering the CP headed by *that*. Thus when it encounters the second □ node in the higher CP, it can once again choose freely how to rewrite it. Provided the four derivation trees obtained from the underspecified derivation

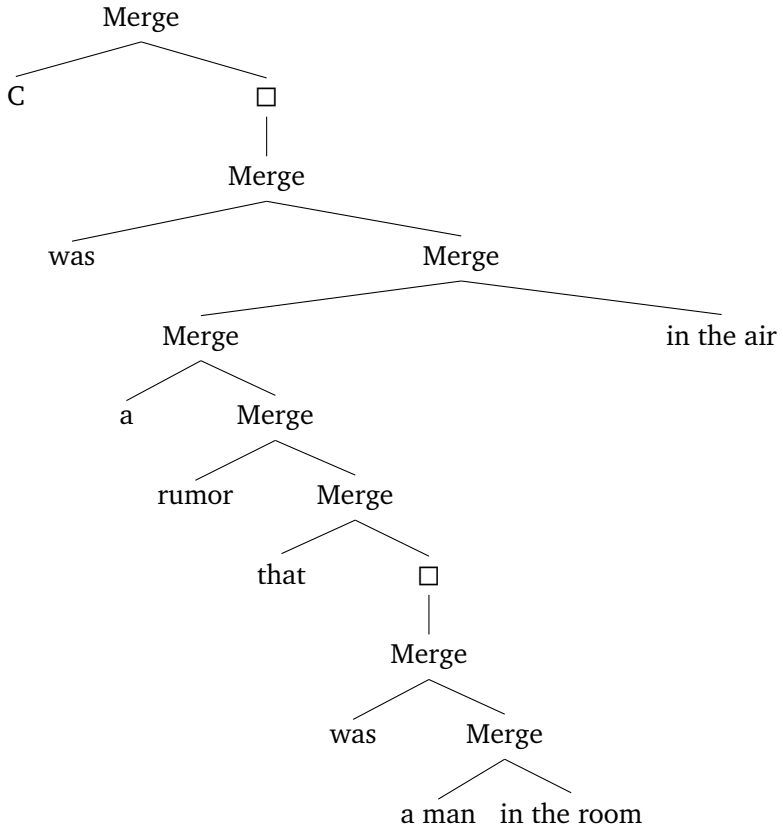


Figure 4.7: Underspecified derivation tree of (26a) and (26b).

aren't filtered out by the MG, they are in turn transformed into the following derived structures, all of which are grammatical:

(27) a. There was a rumor that there was a man in the room in the air.

b. There was a rumor that [a man]$_i$ was t_i in the room in the air.

c. [A rumor that there was a man in the room]$_i$ was t_i in the air.

d. [A rumor that [a man]$_i$ was t_i in the room]$_j$ was t_j in the air.

The identity of numerations condition of the original version of MOM entails that these four sentences belong to three distinct equivalence classes, one containing (27a), one containing (27b) and (27c), and one containing (27d). MOM enriched with stratified numerations, on the other hand, puts each sentence into its own equivalence class. Only tMOM lumps them all together into one equivalence class, which is the more plausible route to take, at least intuitively.

The very fact that Merge variants as well as Move variants can be obtained from the same underspecified derivation indicates that the transducer version is less of a relativized ban against Merge and more of a description of the set of possible continuations of a derivation once a choice pro-Merge or pro-Move has been made. Empirically, this has the welcome effect that we do not run into the undergeneration problems that plague isMOM, and to a lesser degree csMOM. Consider the following utterances.

(28) a. It seems that John was in the room.

b. * John seems it was in the room.

The derivation for either sentence starts out by assembling the small clause [John [in the room]], which is subsequently merged with a T head (phonetically realized by *was*). Now isMOM would enforce base-merger of *it* into the specifier of the TP, rather than movement of *John* into said position. From there on, only ungrammatical structures can be generated. Either *John* remains in situ and the derivation crashes because of the unchecked case feature of *John*, or *John* moves over the expletive into SpecTP of the matrix clause, in violation of the Shortest Move Condition. The only grammatical alternative, (28a), cannot be generated because it is blocked by isMOM. With tMOM one does not run into this problem, as it will generate both sentences, but the second one will probably be filtered out by the MG itself because of the illicit movement step. The csMOM variant alternates between the two options: If (28b) is ungrammatical for independent reasons, (28a) does not have to compete

against it and will emerge as the winner, just as with the transducer model. If (28b) is grammatical, it will block (28a), in line with isMOM.

This general theme is repeated in various configurations where other versions of MOM undergenerate. Shima (2000) lists a number of cases where Merge-over-Move makes false predictions and, in fact, something along the lines of a Move-over-Merge principle seems to be required.

(29) a. It is asked [how likely t_{John} to win]$_i$ John is t_i.

b. * John is asked [how likely t_{John} to win]$_i$ it is t_i.

The assembly of [is [how likely John to win]] proceeds as usual. At this point, a decision has to be made as to whether we want to move *John* into SpecTP or base-merge the expletive instead. The isMOM variant once again picks the base-merger route, so we end up with [it [is [how likely John to win]]. After this phrase is merged with *asked* and *is*, *John* moves into the specifier of the matrix TP to get its case feature checked. Unless moving *John* is barred for independent reasons, (29b) will be grammatical, so that (29a) will be blocked under both indiscriminate and cautious construals of MOM. Thus we get the following contrast between different versions of MOM. The variant isMOM always blocks (29a), csMOM blocks it only if (29b) is grammatical, and tMOM never blocks it. So for both csMOM and tMOM we have to make sure that our MG $\mathscr{G}$ contains some locality condition that rules out (29b). A natural candidate would of course be the islandhood of [how likely John to win].

We also have to make further assumptions about $\mathscr{G}$ to rule out cases of super-raising like (30a) and multiple occurrences of *there* as in (30b). On a conceptual level, this is a defensible move as the deviancy of those examples does not seem to be directly related to MOM, and they are hardly ever discussed with respect to MOM in the literature. However, if we really wanted to incorporate those restrictions into MOM, at least the ban against double *there* can easily be accommodated

by changing from a "once you go Move, you never go back" version of *Path Condition* to "once you have chosen, it's always Move". This is easily accomplished by replacing the rule $\Box(q_*(x)) \rightarrow q_*(\text{Merge}(\text{there}, x))$ by the minimally different $\Box(q_*(x)) \rightarrow q_o(\text{Merge}(\text{there}, x))$.

(30) a. * A man seems there to be in the room.

b. * There seems there to be a man in the room.

Interestingly, at least German allows for multiple expletives to occur in a single clause, even within the *mittelfeld*, which is usually considered a part of the TP. Examples are given in (31) (my own judgments). As multiple expletives can be hosted by German TPs, the contrast between German and English can't be reduced to the fact that German mandatorily requires SpecCP to be filled and thus has two specifiers that may host expletives.

(31) a. Es/Da scheint da ein Mann im Garten zu sein.
it/there seems there a man in.the garden to be

b. Es/?Da scheint da ein Mann da im Garten zu sein.
it/there seems there a man there in.the garden to be
'There seems to be a man in the garden.'

c. Es/?Da scheint da ein Mann im Garten da zu sein.
it/there seems there a man in.the garden there to be
'There seems to be a man in the garden.'

If we assume that economy principles are universal, then any cross-linguistic variation has to arise from other grammar-internal factors. From a transducer perspective, though, there are no good reasons for such a stipulation. As long as language-specific variants of a constraint all belong to the same transducer class, they are all equally economic in a mathematical sense. In the case of *Path Condition*, the slight modification proposed above has absolutely no effect on the runtime-behavior of the transducer, nor is it in any tangible way less intuitive or less "Minimalist". Reference-set constraints must not be artificially kept away from matters of crosslin-

guistic variation, because this is an empirical domain where they are in principle superior to standard well-formedness conditions. This has not been noticed in the syntactic literature yet — e.g. for Müller and Sternefeld (1996:491) "it is [...] not clear how a [reference-set; TG] constraint like Economy can be rendered subject to parametrization" — but in contrast to well-formedness conditions these constraints offer multiple loci of parametrization: the transductions α and β, and the definition of the filter as well as at which point of the transduction it is applied. Now that our formal understanding of reference-set constraints has finally reached a level where at least such basic questions can be given satisfactory answers, the initial empirical questions can be reapproached from a new angle that challenges the received wisdom on when, where and how reference-set constraints should be employed.

4.4 Example 3: Shortest Derivation Principle

In the last section, I left open how to formalize oMOM, the variant of MOM which doesn't weigh violations depending on how early they happen in the derivation. In other words, oMOM simply counts the number of violations and picks the candidate(s) that incurred the least number of violations. This is very close in spirit to the *Shortest Derivation Principle* (SDP) of Chomsky (1991, 1995b), which I (and many authors before me) have also referred to as *Fewest Steps*.[2]

The SDP states that if two convergent (i.e. grammatically well-formed) derivations are built from the same LIs, the one with the fewest operations is to be preferred. Usually, the set of operations considered by the economy metric is assumed to comprise only Move, the reason being that Merge is indispensable if all the LIs are to be combined into a single phrase marker. Naively, then, oMOM is but a variant of the SDP that does not penalize every instance of Move but only those

[2]Technically, Fewest Steps is the original formulation and the SDP its more "minimalist" reformulation that does away with representational machinery such as Form-Chain. This makes it a better fit for MGs, and for this reason I prefer the name SDP over the better-known Fewest Steps.

where Merge would have been a feasible alternative. Even though I will refrain from discussing oMOM any further in this section and focus on the SDP instead, their close relation means that after reading this and the previous section, the reader will be in possession of all the tools required to formalize oMOM. In fact, I explicitly encourage the reader to draw at least a sketch of the implementation to test their own understanding of the material.

Returning to the SDP, I explore two variants of this principle, one being the original proposal and the other one the result of extending the set of operations that enter the economy metric to Merger of phonologically unrealized material such as (certain) functional heads in the left periphery. The underlying intuition of this extension is that covert material should be merged only if it is required for convergence. Curiously, this *prima facie* innocent modification has the potential to push the SDP out of the realm of linear transductions: the SDP restricted to Move can be defined by a linear transducer, whereas the SPD applied to Move and Merge of covert material is not, unless restrictions on the distribution of silent heads are put into place.

4.4.1 The Shortest Derivation Principle Explained

To give the reader a better feeling for the constraint I once more present a simple example first. It is a well-known fact that A-movement in English exhibits freezing effects. While arguments may be extracted from a DP in complement position, as soon as the DP A-moves to a higher position, usually Spec,TP, extraction is illicit — the DP's arguments are frozen in place. This contrast is illustrated in (32).

(32) a. Who$_i$ did John take [$_{\text{DP}}$ a picture of t_i]?

b. * Who$_i$ was [$_{\text{DP}_j}$ a picture of t_i] taken t_j by John?

At first (32b) seems to be a mere instance of a CED-effect (Huang 1982) as in (33), so whatever rules out the latter should also take care of (32b).

(33) Who$_i$ is [$_{DP}$ a picture of t_i] on sale?

The corresponding derivation for this analysis of the ungrammaticality of (32b) would be (34).

(34) a. [$_{VP}$ taken [$_{DP_j}$ a picture of who$_i$] by John]

b. [$_{TP}$ [$_{DP_j}$ a picture of who$_i$] T [$_{VP}$ taken t_j by John]]

c. [$_{CP}$ who$_i$ was [$_{TP}$ [$_{DP_j}$ a picture of t_i] T [$_{VP}$ taken t_j by John]]]

Notably, though, the DP in (32b) is not base-generated in subject position but in object position, so in theory it should be possible to extract the wh-word from the DP before it moves into subject position and thus becomes a barrier for movement in the sense of Chomsky (1986). There are two distinct derivations that make use of this loophole, the relevant stages of which are depicted in (35) and (36) below.

(35) a. [$_{VP}$ taken [$_{DP_j}$ a picture of who$_i$] by John]

b. [$_{CP}$ who$_i$ was [$_{TP}$ T [$_{VP}$ taken [$_{DP_j}$ a picture of t_i] by John]]]

c. [$_{CP}$ who$_i$ was [$_{TP}$ [$_{DP_j}$ a picture of t_i] T [$_{VP}$ taken t_j by John]]]

(36) a. [$_{VP}$ taken [$_{DP_j}$ a picture of who$_i$] by John]

b. [$_{VP}$ who$_i$ taken [$_{DP_j}$ a picture of t_i] by John]

c. [$_{TP}$ [$_{DP_j}$ a picture of t_i] T [$_{VP}$ who$_i$ taken t_j by John]]

d. [$_{CP}$ who$_i$ was [$_{TP}$ [$_{DP_j}$ a picture of t_i] T [$_{VP}$ taken t_j by John]]]

The first derivation can be ruled out on grounds of the extension condition, which bans countercyclic movement. The second, however, seems to be well-formed, provided that extraction of the wh-phrase is licensed by feature checking (in a phase-based approach, this could be handled by an EPP/OCC-feature, for instance). So we erroneously predict that (32b) should be grammatical.

Collins (1994) solves this puzzle by recourse to the SDP. Note that (34) and (35) involve one movement step less than (36). So if (36) has to compete against

at least one of the two, it will be filtered out by the SDP. The filtering of (34) and (35), respectively, is then left to the subject island constraint and the ban against countercyclic movement (whatever their technical implementation might be in our grammar).

4.4.2 A Model of the Shortest Derivation Principle

The SDP features interesting extensions of the constraints seen so far. As it punishes every single instance of Move, it needs some limited counting mechanism, in contrast to Focus Economy and MOM, which relied on surprisingly simple well-formedness conditions on somewhat peculiar paths. This means that the SDP is the first time that we have to fall back to the strategy of Jäger and define a transduction for the ranking of candidates. This ranking will be computed by a transducer that non-deterministically adds Move nodes to derivations. That way, if a derivation d is rewritten as d', we know that d is more optimal than d'. The derivations that cannot be obtained from other derivations by adding Move nodes are optimal with respect to the SDP.

As with MOM, we start out with the set of derivation trees of some MG $\mathscr{E}$. And as we did before, we immediately strip away all the features except the category feature, which is preserved so that distinct trees with identical string components won't be put in the same reference set later on.

Definition 4.11. *Remove Features* is the deterministic (one-state) relabeling that maps each $l := \langle \sigma :: f_1, \ldots, f_{base}, \ldots, f_n \rangle \in Lex_{\mathscr{E}}$ to $l' := \sigma_{f_{base}}$, where f_{base} is the base feature of l. The set of these simplified LIs is denoted by Λ.

In the next step, we have to ensure that two derivations wind up in the same reference set if and only if they differ merely in their number of movement steps. To this end, we first define a transducer that deletes all unary branches (i.e. branches representing Move) from the derivation, and then another one which arbitrarily

reinserts unary branches. This will generate derivations that were not present at the stage immediately before we removed all instances of Move, but as the reader might have guessed, this can easily be fixed by following our own example set in the previous section and use the input language as a filter — only this time the "input language" isn't the derivation language of $\mathscr{E}$ but the language serving as the input to the transducer that removed all movement nodes, i.e. the output language of *Remove Features*. The result of these three transductions and the filtration is a transduction that relates only those (feature-free) derivations that are identical *modulo* movement.

Definition 4.12. *Remove Move* is the deterministic ltdtt $\mathscr{R}$, where $\Sigma_{\mathscr{R}} := \Lambda \cup \{\text{Merge}, \text{Move}\}, \Omega := \Sigma_{\mathscr{R}} \setminus \{\text{Move}\}, Q = F := \{q\}$, and $\Delta_{\mathscr{R}}$ consists of the rules below:

$$\sigma \to q(\sigma) \qquad \text{Merge}(q(x), q(y)) \to q(\text{Merge}(x, y))$$
$$\text{Move}(q(x)) \to q(x)$$

Definition 4.13. *Insert Move* is the non-deterministic ltdtt $\mathscr{I}$, where $\Sigma_{\mathscr{I}} := \Omega_{\mathscr{R}}$, $\Omega_{\mathscr{I}} := \Sigma_{\mathscr{R}}$, $Q = F := \{q\}$, and $\Delta_{\mathscr{I}}$ contains the rules below, with $O^{\leq n}$ denoting n-many unary O-labeled branches or less for some fixed, non-negative n:

$$\sigma \to q(\sigma) \qquad \text{Merge}(q(x), q(y)) \to \text{Move}^{\leq n}(\text{Merge}(x, y))$$

One strong restriction of *Insert Move* is that at any node in the derivation tree it can only insert a finite number of movement steps. This is so because a transducer may only have finitely many rules and after every step in the transduction the transducer has to move one step up in the input tree, so it cannot remain stationery at one point and keep inserting one unary branch after another until it finally decides

to move on. A transducer with such capabilities is said to have ϵ-moves, and such transducers do not share the neat properties of their standard brethren. However, the restriction to only finitely many unary branches per rewrite-step is immaterial for MGs. This follows from the simple observation that since the number of features per LI is finite, and so is the lexicon itself, there is a longest string of features for each grammar. The length of this string dictates how many movement steps may be licensed by a single LI, and thus there is an upper bound on the number of movement steps between any two instances of Merge. For MGs, the limits of the transducer are inconsequential because they are also limits of the grammar.

The composition of *Remove Features*, *Remove Move*, *Insert Move*, and the diagonal of the output language of *Remove Features* will be our reference-set algorithm. It maps every derivation to the derivations that differ from it only in the number of Move nodes. The next step, then, is the definition of the economy metric. But for this not much more work is needed, because the metric is already given by *Insert Move*. The transducer all by itself already defines the ranking of all output candidates relativized to those candidates that compete against each other, so all we have to do is follow Jäger's procedure. Recall the basic intuition: the transducer defines a relation $<$ on the output candidates such that $o < o'$ iff o' is the result of applying the transducer to o. Given this relation, a few nifty regular operations are enough to filter out all elements that are not minimal with respect to $<$, i.e. the suboptimal candidates. The result will be a transduction mapping, as desired, inputs to the derivation tree(s) over the same LIs that contain(s) the fewest instances of Move — it only remains for us to reinstantiate the features, which is taken care of by the inverse of *Remove Features*, and to remove any output trees that weren't part of the original MDTL.

The astute reader may point out that my implementation of the SDP, while technically correct, leads to both underapplication and overapplication of the intended principle. Overapplication is caused by the indiscriminate removal of features, in

particular movement-licensors that are involved in topicalization, wh-movement and (possibly) scrambling. As a consequence, these instances of movement will appear redundant to the SDP and cause the derivation to lose out to the one that involves only standard A-movement. This is easily fixed by "blacklisting" these features such that they have to be preserved by *Remove Features*.

Underapplication, on the other hand, is due to the lack of a transduction that would remove covert material whose only purpose is to host a movement-licensor feature. So if, say, a topicalization feature is always introduced by the category *Topic* (cf. Rizzi 1997, 2004), a derivation hosting this functional element will never compete against a derivation without it. For topicalization, this is actually a welcome result and presents an alternative for avoiding overapplication. In general, though, this must be regarded as a loophole in the SDP that needs to be fixed lest the principle can be deprived of any content by assigning every movement feature its own functional category. A solution is readily at hand: Extend *Remove Move* and *Insert Move* such that they may also remove or insert certain functional elements, just like MOM's *Underspecify* may remove instances of expletive *there* that can later be reinserted by *Path Condition*.

While the parametrization of *Remove Features* poses no further problems irrespective of the MG involved, extending *Remove Move* and *Insert Move* to functional categories will produce the correct results only if our initial grammar does not allow for recursion in the set of categories that the transducer should remove. In other words, there has to be an upper limit on the number of removable categories that can be merged subsequently before non-removable material has to be merged again. This is because of the previously mentioned inability of linear transducers to insert material of unbounded size. On a linguistic level, the ban against recursion in the functional domain is fairly innocent as even highly articulated cartographic approaches give rise only to a finite hierarchy of projections.

4.4.3 Scope Economy: A Semantic SDP?

At first glance, the SDP seems very similar to Scope Economy (Fox 1995, 2000), which was briefly mentioned in example 3.6 on page 149 during the discussion of the limits of MSO with respect to semantics. Scope Economy limits the cases where a quantifier may be covertly displaced via QR. In the original formulation, QR is allowed only if movement of the quantifier brings about a change in meaning. Later on, this condition was weakened such that QR is blocked only if it is semantically vacuous in all configurations. That is to say, if permuting two quantifiers never changes meaning, then any QR step inducing such a permutation is illicit. For instance, QR of a universal is blocked across another universal but fine across an existential because there is a possibility that a new meaning might arise in the latter case. As Scope Economy is a ban against extraneous movement, one might be inclined to treat it as a variant of the SDP. But this view is problematic.

First, linear transducers have the same limitations as MSO when it comes to semantics. They can only read and manipulate structure, all semantic inferences that be expressed in purely structural terms are beyond their capabilities. This limitation immediately rules out the strong version of Scope Economy which evaluates the impact of QR on the specific meaning of the tree. But even the weak interpretation of Scope Economy exceeds the limits of linear transductions. Recall that the SDP transduction contains a substep in which Move nodes are inserted in the derivation, and that a linear transducer can only insert a finitely bounded number of these nodes at any given position. If Scope Economy were to be implemented in an SDP-style fashion, these Move nodes would have to inserted in the C-domain, which hosts the landing sites for QR. But since the size of CPs is unbounded, a single CP may contain an unbounded number of quantifiers. If all these quantifiers are eligible to undergo QR, then there is no upper limit of Move nodes that might have to be inserted above the C-head. Hence no linear transduction could correctly model Scope Economy.

But the problem goes even deeper. The limitations of linear transductions could be overcome by a more powerful transducer model, e.g. extended linear top-down tree transducers (Graehl et al. 2008; Maletti 2008:cf.). These transducers allow for ϵ-transition, which means that rules can be applied arbitrarily often to the same node. They also preserve regularity, so an RC that can be described by such an extended linear transduction can still be expressed via Merge (they are not closed under composition, though, so a cascade of these transducer might not itself be an extended linear top-down transduction). Hence it might be possible to implement Scope Economy as an extended linear top-down tree transduction. But the derivation trees created this way would not be well-formed Minimalist derivations because only a finite number of LIs may move at any given point in the derivation. Moving an unbounded number of quantifiers to the same CP simply isn't possible with canonical MGs.

I emphasized several times throughout this thesis that MGs provide a very malleable formalism that can easily be adapted to meet various demands. This becomes particularly easy once one views MGs as the pairing of MDTLs with a specific mapping from derivations to multi-dominance trees. The crucial property of MDTLs is their regularity, and said regularity would be preserved even if an unbounded number of Move nodes could occur at any position (just like the string language ab^*a in which an arbitrary number of bs may occur between the two as is regular). The derivational limits of MGs, then, could be fixed to reign in Scope Economy, but even this would be insufficient, because the real source of the problem is the MSO transduction. If there is no upper bound on the number of movement nodes, then we can no longer reliably pick out the Move nodes that check an LI's licensee features. For every licensee feature it is still possible to find the block of Move nodes that contains the one associated to a matching licensor feature, but it cannot be determined which node within this block is the right one. This means that if an unbounded number of quantifiers undergo QR, one can tell that they all will

occupy specifiers of the containing CP, but their relative order cannot be uniquely determined. An existential might end up higher than a universal, or maybe lower, there is no way to enforce a unique order. But the relative scope of the quantifiers is exactly what matters for Scope Economy, so if allowing for an unbounded number of QR steps comes at the price of losing the means to determine their relative order, there simply is no feasible way of accommodating Scope Economy as a variant of the SDP.

In conclusion, an SDP-style implementation of Scope Economy is only feasible for the weakened version that foregoes all semantic interpretation, and even then the number of quantifiers per CP that may undergo QR must be finitely bounded. I do not know whether this restriction is empirically tenable. If it turns out that no such upper bound exists, a viable strategy might be to adopt Scope Economy to different types of movement. For example, the clustering movement presented in Gärtner and Michaelis (2010) can combine an unbounded number of constituents into a single cluster than could then undergo movement to Spec,CP. Provided all possible orders of quantifiers can be obtained by clustering, Scope Economy could be stated as a restriction on clustering rather than QR.

4.5 The Chapter in Bullet Points

- Transderivational constraints use a relative notion of grammaticality. A tree is grammatical only if there is no competing tree that is more optimal according to the relevant metric.
- Transderivational constraints are modeled by tree transducers. Rather than acting as a filter that removes suboptimal derivations, they are rewrite rules that turn suboptimal derivations into optimal ones.
- A tree transducer is linear if isn't allowed to copy subtrees, which is never

needed for transderivational constraints.

- As MDTLs are regular and the image of a regular tree language under a linear tree transduction is a regular tree language, a transderivational constraint can be expressed by Merge if it is a linear tree transduction from MDTLs into themselves.
- Case studies of three constraints from the literature indicate that most transderivational constraints satisfy this property.

CONCLUSION

A lot of ground was covered in this thesis, on a technical as well as a conceptual level. The initial research question concerned the status of constraints in Minimalist syntax, in particular whether a syntax with constraints is more powerful than one without. Rather than detailed case studies or philosophical arguments, mathematical and computational techniques were the essential tools in this inquiry. Minimalist grammars served as a model of Minimalist syntax, and monadic second-order logic as a powerful yet computationally well-behaved description language for constraints.

My central result is that all constraints that can be defined in monadic second-order logic are expressible via Merge. This finding hinges on just two facts: I) Merge is symmetric in the sense that it involves the checking of a category feature on the argument and a selector feature on the head, and II) constraints definable in monadic second-order logic can be converted into finite-state tree automata. In order to enforce a constraint via Merge, it suffices to refine the lexical entries such that category and selector features are suffixed with the states of the automaton computing said constraint. Hence any version of Minimalist syntax that uses subcategorization restrictions of some kind furnishes all the means to express a wide variety of constraints.

The expressive equivalence of Merge and monadic second-order logic has interesting implications. We saw that constraints over representations are exactly as powerful as constraints over derivations. Moreover, locality restrictions cannot limit the power of these constraints. Even transderivational constraints can be expressed via Merge as long as they do not rely on semantic notions such as identity of meaning.

Crucially, though, all these equivalences hold only with respect to the expressive power of constraints. Constraints can still differ in succinctness so that some generalizations are easier to state over representations than derivations, or the other

way round. Most importantly, the feature refinement strategy that makes it possible to express constraints via Merge can cause enormous blow-ups in the size of the lexicon and as a consequence, it obscures generalizations that could be captured in a succinct manner by constraints.

To a certain extent, the expressive equivalence of Merge and monadic second-order logic constitutes cause for worry because the latter can define configurations that are highly unnatural from a linguistic perspective. Why, then, don't we find them in language even though Merge already provides syntax with the necessary means? One option might be to alter the feature checking mechanism involved in Merge, but it is unclear how it could be sufficiently weakened and still capture all subcategorization restrictions. A more intriguing option is that syntax is indeed capable of generating unnatural languages, but that none of the grammars that do so are learnable and/or parsable. This would be in the spirit of Chomsky's (2005) idea that the class of natural language is restricted by factors external to UG. It will be interesting to see whether this line of research can limit the class of syntactic constraints in a meaningful way.

Part III

Appendices

APPENDIX A

Basic Mathematics

Contents

This appendix is supposed to bring readers up to speed with basic mathematical vocabulary and notation. Most of the material here should already be familiar to readers who had the pleasure of reading or attending a decent graduate-level intro to semantics. Among the covered topics are sets and tuples as well as basic operations on them (union, intersection, complement, concatenation), followed by relations, functions, and orders. Propositional logic and first-order logic are also discussed very briefly. The appendix closes with a discussion of pairs and their representation in terms of sets of the form $\{\{a\},\{a,b\}\}$. The set-theoretic definition of pairs has attracted some interest from syntacticians due to its resemblance to bare phrase structure sets like $\{a,\{a,b\}\}$. I prove that this simplified format is already powerful enough to represent pairs, despite claims to the contrary in the syntactic literature. Thus the last section might be of interest even to readers with a well-rounded mathematical background.

A.1 Textbooks and References

There are several beginner-friendly textbooks that are suitable for linguists and do a much better job at covering the material than I can hope to accomplish in a few pages. None of them are required or even recommended reading for this thesis, but they are great sources for the curious reader who would like to learn more. Partee et al. (1990) is a solid introduction to set theory, algebra and logic with an eye towards semantic applications. It also contains a closing chapter on formal language theory. The second edition fixes many typos (farewell, my beloved lambada calculus) but is not an indispensable upgrade over the first one. Keenan and Moss (2008) is a more succinct introduction with a greater emphasis on algebra, at the expense of logic and formal language theory (it also has the minor disadvantage of not being published yet). Gamut (1990a,b) is a very rigorous and mathematically sophisticated introduction to logic and algebra for semanticists.

Readers that carefully work through the exercises in those books should be at a level where they can pick up some undergraduate-level mathematics textbooks. Enderton (2001) is an excellent introduction to first-order logic. Ebbinghaus et al. (1996), which covers a slightly different selection of topics, is a highly recommended supplement, but its very European style of presentation makes for a dense read. The fifth edition features solutions to selected exercises but has not been translated into English yet to the best of my knowledge. Davey and Priestley (2002) is a slim but informative textbook on order theory and lattices. However, anything past chapter 4 is of little relevance to linguistics.

I do not cover any formal language theory in this chapter as all relevant concepts are slowly introduced throughout the main body of the thesis. For further background information, readers should consult Hopcroft and Ullman (1979), which is still the best reference on formal string languages (the revised version by Hopcroft, Motwani and Ullman has some unfortunate omissions). If a gentler learning curve is

preferred, Sipser (2005) is a better choice. Kozen (1997) lies between the two regarding difficulty and coverage. Tree languages and tree automata aren't commonly taught topics even in computer science graduate programs, and as such there are no textbooks or surveys that do not already presuppose a significant amount of mathematical maturity and familiarity with formal language theory. The best starting point is Gécseg and Steinby (1984, 1997), while Comon et al. (2008) features a more contemporary selection of topics, including a chapter on the connection between tree automata and monadic second-order logic.

A.2 Sets and Tuples

Sets naively The notion of a set has been the subject of extensive research in philosophy and mathematics. Fortunately, a naive approach that ignores those foundational issues is sufficient for our purposes. Hence a *set* will simply be viewed as a collection of zero or more objects that are all distinct from each other. We do not make any specific assumptions about the ontological status of these objects, nor what their properties are. In particular, they may be sets, too, i.e. a set can be a collection of other sets. A set of sets is also called a *family*.

By convention sets are denoted by curly braces, so that the set of a, b, and c is written $\{a, b, c\}$. More generally, $\{a_1, \ldots, a_n\}$ is the set containing all a_i for $1 \leq i \leq n$, and only those. Suppose we call this set A. Then each a_i is an *element* — or equivalently, a *member* — of A. In mathematical notation this is indicated by $a_i \in A$ or $A \ni a_i$. On the other hand, $a_i \notin A$ and $A \not\ni a_i$ both state that a is not a member of A. The *empty set* $\{\}$ is the set with no members, and is often denoted by the special symbol $\emptyset$. Given a set A, its *size* or *cardinality*, written $|A|$, is identical to the number of its members (so $|\emptyset| = 0$). A set is *singleton* iff its cardinality is 1.

Example A.1 Some sets and their members

The set $\{a, \{a, b\}\}$ has two members: a and $\{a, b\}$. Consequently, it has cardinality 2. Crucially, $b \notin \{a, \{a, b\}\}$ even though $b \in \{a, b\}$. This holds because set containment differs from the intuitive understanding of containment in that $b \in B$ and $B \in A$ does not jointly imply $b \in A$ (as we will see in Sec. A.3, this means that set containment is not a transitive relation). The minimally different set $\{a, a, b\}$, on the other hand, has b among its members. However, its cardinality is 2 rather than 3 since $\{a, a, b\}$ is equivalent to $\{a, b\}$ (remember that we only consider distinct elements). If furthermore $a = b$, then $\{a, a, b\} = \{a, b\} = \{a\} = \{b\}$ and the set is a singleton.

Some sets commonly encountered in high-school mathematics are the *natural numbers* $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$, the *positive natural numbers* $\mathbb{N}^+ := \{1, 2, 3, \ldots\}$, the *integers* $\mathbb{Z} := \{0, 1, -1, 2, -2, \ldots\}$, the *rationals* $\mathbb{Q} := \left\{0, 1, -1, \frac{1}{2}, -\frac{1}{2}, \ldots\right\}$, and the *reals* $\mathbb{R}$, which consists of all numbers with an infinite number of digits, e.g. $0.0000\ldots$, $1.1546666\ldots$, and $3.14159\ldots$. A proper definition of the latter would take us too far here, and they do not play an important role in this thesis anyways, for reasons to be discussed in a second.

Infinite sets It should be clear intuitively that $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ have infinitely many members and thus infinite cardinality. At the same time, though, one might expect that some of these sets are bigger than others. After all, the set $\mathbb{Z}$ of integers contains both a positive and a negative variant for every member of $\mathbb{N}^+$, so shouldn't it be bigger than the latter? The answer is "yes and no". Infinity indeed comes in different sizes. In fact, there are infinitely many sizes of infinity. They are, in increasing order of size, denoted $\aleph_0, \aleph_1, \aleph_2, \ldots$, and so on (where $\aleph_i$ is pronounced "aleph i"). The cardinality of $\mathbb{N}$ is $\aleph_0$. No infinite set can be smaller than this. This immediately entails that $|\mathbb{N}| = |\mathbb{N}^+|$, even though $\mathbb{N}$ contains every element of $\mathbb{N}^+$ while the opposite does not hold — 0 is not a member of $\mathbb{N}^+$. To add further to the confusion,

it also holds that $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| = \aleph_0$, but $|\mathbb{R}| = \aleph_1 > \aleph_0 = |\mathbb{N}|$. So out of all the sets listed above, only $\mathbb{R}$ contains more elements than $\mathbb{N}$. We do not have the tools yet to appreciate the reasoning behind these equivalences, in particular why $\mathbb{R}$ is bigger than $\mathbb{N}$. The important point is that sets of cardinality $\aleph_0$ — which are called *countable*, *recursively enumerable*, *denumerable*, or *denumerably infinite* — are computationally well-behaved in that all their members can be enumerated in a purely mechanical fashion (given unlimited time and resources).

As a corollary, every element of a countable set can be retrieved after a finite amount of time — when asked to count from 0 to some natural number n, an immortal being existing inside an ever-expanding universe will eventually reach the number n, even though the being is still incapable of naming all natural numbers in a finite amount of time. More generally, the elements of sets with cardinality $\aleph_0$ can be enumerated with a procedure p such that every element is retrievable in a finite amount of time. The procedure p varies between sets, but it must exist for every set of cardinality $\aleph_0$. Sets of cardinality $\aleph_1$ and higher, are *non-denumerably infinite*, or simply *non-denumerable*. A non-denumerable set contains so many elements that it is impossible to enumerate all of them in a purely algorithmic fashion.

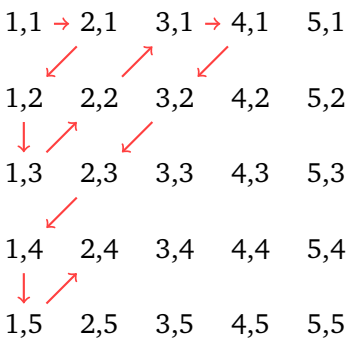
Example A.2 Counting denumerable sets

Enumerating the set of all natural numbers is easy, as one simply has to count up from 0. For the positive natural numbers, one starts at 1 instead. In order to enumerate all integers, one must alternate between positive and negative numbers. One successful strategy would proceed as follows: $0, 1, -1, 2, -2, \ldots$.

Counting through all the rationals without skipping any numbers takes some ingenuity. Every rational is represented as a fraction of two integers, the numerator above the line and the denominator below it. At first one may try enumerating the rationals by keeping the numerator fixed while counting through all denominators.

Once that step is completed, the numerator is incremented (or decremented for negative numbers) and one cycles through all denominators again. But this strategy must fail, of course, because there are infinitely many integers and thus infinitely many denominators, so that the first step would never terminate. For instance, if the numerator is fixed to be 1 the first step would only serve to list the rationals between 0 and 1 and one would never reach, say, 2.

An easy procedure for enumerating the rationals relies on representing them as a diagram that can easily be traversed from one corner to the other. Suppose we have a table that extends infinitely far to the right and to the bottom. This table has as many rows as there are positive natural numbers, and exactly as many columns. Every field in this table can be given by two coordinates x and y such that x denotes the column number and y the row number. The positive rationals can also be described in terms of two positive natural numbers, so we can identify each cell with coordinates x and y with the positive rational with numerator x and denominator y (some cells will correspond to the same rational, for instance the cells $(2,1)$ and $(4,2)$, but this isn't important here). In order to show that the positive rationals are denumerable, we have to show that this infinite table can be traversed in a fashion such that each cell is encountered at some point of the traversal. The safest way of doing so is to start at the cell $(1,1)$ and then zig-zag diagonally towards the right bottom corner. So the cells would be reached in the order $(1,1)$, $(2,1)$, $(1,2)$, $(1,3)$, $(2,2)$, $(3,1)$, $(4,1)$, $(3,2)$, $(2,3)$, $(1,4)$, $(1,5)$, $(2,4)$, and so on. The picture below illustrates the first few steps.



In order to extend this procedure to all rationals, we slightly alter our interpretation of the cells such that each coordinate represents the value $(x+1)/2$ if it is odd and $-x/2$ otherwise. For instance, the cell $(4,5)$ then corresponds to the fraction $\frac{-4/2}{(5+1)/2} = -\frac{2}{3}$.

Example A.3 (A failed attempt at) counting non-denumerable sets

Let us go back to the immortal being we tasked with enumerating the natural numbers, and let us increase the stakes by challenging it to list all real numbers. Suppose furthermore that the being has spent some part of eternity on doing mathematics and has figured out how to enumerate the rationals. Would this strategy also work for the reals?

The answer is no. The reals include the irrational numbers, which cannot be represented by finite means. So one cannot use the cells of a table to represent all reals. Even using tables with more than 2 dimensions does not change this fact. The immortal being would have to use a table with infinitely many dimensions, but then the diagonal traversal would no longer be able to reach every cell in the table. Intuitively this is the same problem we encountered during our first attempt

at enumerating the rationals by keeping the numerator fixed and counting through all denominators.

Obviously the intuitive argument above does not qualify as a formal proof that there is absolutely no procedure that could enumerate all reals. The important point is that infinity comes in different sizes, and that a denumerably infinite set is small enough that any given member can still be found in a finite amount of time. Non-denumerable sets pose all kinds of computational challenges — for instance, computers cannot calculate with real numbers and must use finite approximations via floating point representations. Fortunately, all the infinite sets covered in this thesis are denumerable. In particular, the class of Minimalist grammars is denumerable, as is every Minimalist derivation tree language. So all the infinities we have to deal with are well-behaved in the sense that they are in principle computable, although it is possible that some problems might still demand more memory or take more time than is practically feasible.

Set-builder notation Let us return from the lofty realm of infinities to more practical issues. As we just saw a set may contain an infinite number of elements (and even more than that), so it isn't feasible to define every set merely by listing all its members, as we have done so far. Even when a list is sufficient it is often an inelegant solution. Instead one may resort to *set-builder notation*. This notation makes it possible to define a set A via a property that is satisfied by all elements of A, and only those. The general format is $A := \{a \mid \text{property } P \text{ holds of } a\}$. It states that the set A contains of all elements a that satisfy property P, and only those.

The expression can be decomposed as follows. First the set is optionally assigned a name A. This name is then followed by the special symbol $:=$ to signal the start of the definition. Inside the set braces we then find two statements that are separated

by the vertical bar |. In linguistic parlance, the element to the left of the bar is a variable, which must also occur in the statement on the right-hand side. The right-hand side then establishes a condition on the elements of the set. The set-builder notation is often expanded such that several variables can occur to the left of |, operations are applied to the variables, and conditions may also be enforced on the left-hand side.

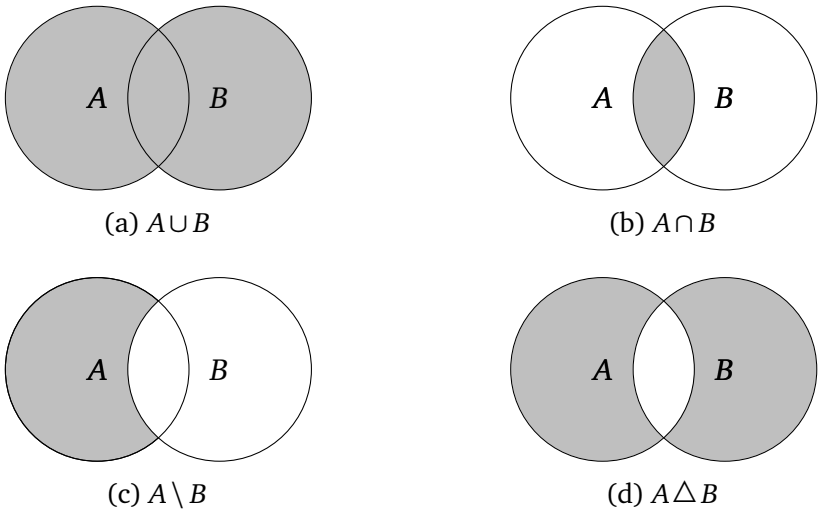
Example A.4 Some set-builder expressions

The set of all odd numbers is given by $\textsc{Odd} := \{n \mid \text{the remainder of } n+1 \text{ divided by 2 is } 0\}$. Note that this includes both positive and negative numbers. For example, $(5+1)/2 = 6/2$ has remainder 0, but so does $(-5+1)/2 = -4/2$. If we want the set of positive odd numbers, we can either use intersection such that $\textsc{Odd}^+ := \textsc{Odd} \cap \mathbb{N}$, or amend our set-builder expression: $\textsc{Odd}^+ := \{n \in \mathbb{N} \mid \text{the remainder of } n+1 \text{ divided by 2 is } 0\}$. We could also define $\textsc{Odd}$ in terms of $\textsc{Odd}^+$: $\textsc{Odd} := \{n, -n \mid n \in \textsc{Odd}^+\}$. And given $\textsc{Odd}$, it is of course very easy to define the analogous set of even numbers: $\textsc{Even} := \{n-1 \mid n \in \textsc{Odd}\}$. Note that contrary to what one might expect, this definition does not yield the set of positive even numbers if $\textsc{Odd}$ is replaced by $\textsc{Odd}^+$. Why not?

Difference between $=$ and $:=$ We just encountered the special symbol $:=$ as a notational device in set-builder expressions. Some authors use the equality sign $=$ instead of $:=$, but this is slightly inaccurate. The difference between $=$ and $:=$ is one of the subtleties of mathematical notation that are often confusing to the uninitiated. Intuitively, $a := b$ assigns b the name a, whereas $a = b$ says that a and b are the same. So $:=$ merely names an object, whereas $=$ is a relation that holds between two objects only if they are in fact the same object. This brings us to our next topic.

Set relations and operations Of course we do not want to content ourselves with merely defining sets, we would also like to compare and combine them in various ways. Given two sets A and B, A is a *subset* of B iff every member of A is also a member of B. In this case we write $A \subseteq B$. If furthermore some element of B is not contained by A, A is a *proper subset* of B, denoted by $A \subset B$. Alternatively, one may also say that B is a *(proper) superset* of A. Two sets are identical iff both $A \subseteq B$ and $B \subseteq A$. Notationally this is expressed by $A = B$.

The *intersection* $A \cap B$ of A and B is $\{a \mid a \in A \text{ and } a \in B\}$. Similarly, $A \cup B := \{a \mid a \in A \text{ or } a \in B\}$ is the *union* of A and B. Note that $A \cap B = A$ iff $A \subseteq B$ iff $A \cup B = B$. If $A \cap B = \emptyset$, A and B are *disjoint*. If A and B are not disjoint and neither is a subset of the other, they are *incomparable*. The *relative complement of A in B* is $B \setminus A := \{b \in B \mid b \notin A\}$. If B is some contextually fixed superset of A, one may simply speak of the *complement* of A, written $\overline{A}$. Note that the complement of the complement of A is A itself, i.e. $\overline{\overline{A}} = A$. The *symmetric difference* of A and B is $A \triangle B := A \setminus B \cup B \setminus A$ (which is equivalent to $(A \cup B) \setminus (A \cap B)$; the reader might want to take a quick break to verify this for himself). See Fig. A.1 for depictions of these operations.



(a) $A \cup B$ (b) $A \cap B$

(c) $A \setminus B$ (d) $A \triangle B$

Figure A.1: Venn diagrams of set-theoretic union, intersection, relative complement, and symmetric difference

Example A.5 Comparing some finite sets

Suppose $A := \{0, 1\}$ and $B := \{0, 1, 2\}$. Since 0 and 1 are both members of A and B, but $B \ni 2 \notin A$ (i.e. 2 belongs only to B), it holds that $A \subset B$ and $A \cap B = A = \{0, 1\}$, while $A \cup B = B = \{0, 1, 2\}$. Moreover, $B \setminus A = \{2\}$, whereas $A \setminus B = \emptyset$. Hence their symmetric difference is $\emptyset \cup \{2\} = \{2\}$.

Now let $red := \{\heartsuit, \diamondsuit\}$ and $black := \{\clubsuit, \spadesuit\}$. Clearly they have no elements in common and are thus disjoint. The relative complement of *red* in *black* is identical to *black*, and the same holds the other way round: $black \setminus red = black$ and $red \setminus black = red$. Consequently their symmetric difference is $\{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\}$, which is also their union.

Finally, the minimally different sets $red' := red \cup \{\natural\}$ and $black' := black \cup \{\natural\}$ are incomparable, since they both contain $\natural$. Nonetheless their relative complements and symmetric difference are equivalent to those of *red* and *black*. However, their union is $\{\heartsuit, \diamondsuit, \clubsuit, \spadesuit, \natural\}$. From this it follows immediately that $(red' \cup black') \setminus (red' \triangle black') = \{\natural\} = red' \cap black'$.

Power sets A particularly important constructor on sets is the power set operation. Given some set A, the *power set of* A is the set of all subsets of A. Formally, $\wp(A) := \{B \mid B \subseteq A\}$. Some authors also write 2^A instead of $\wp(A)$. This notation is motivated by the fact that for every finite set A, $|A| = n$ iff $|\wp(A)| = 2^n$. If instead A is infinite and has cardinality $\aleph_i$, $|\wp(A)| = \aleph_{i+1}$. So the power set of a countable set is non-denumerably infinite.

Example A.6 Power sets

If $A := \{a, b, c\}$, then $\wp(A) := \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. Note that $|A| = 3$ and $|\wp(A)| = 8 = 2^3$. The power set of the empty set $\emptyset$ is $\wp(\emptyset) := \{\emptyset\}$

(and we have $|\emptyset| = 0$ and $|\wp(\emptyset)| = 2^0 = 1$). Hence $\wp(\wp(\emptyset)) = \{\emptyset, \{\emptyset\}\}$. The reader is invited to calculate $\wp(A \cup \emptyset)$, $\wp(A \cup \{\emptyset\})$, and possibly $\wp(A \cup \{\emptyset, \{\emptyset\}\})$ (which will be rather big).

"Big" union and intersection Set-theoretic union and intersection also provide us with yet another way of defining sets. Let F be a (possibly infinite) set of sets A_1, $\ldots$, A_n. Recall that such a set is also called a family (of sets). Sometimes we might want to use a specific set I of indices for the sets contained in a family, in which case we call it an *I-indexed family*. Given such a family F, then, $\bigcap F := A_1 \cap \cdots \cap A_n$ is the largest set that contains all elements that occur in every A_i that belongs to F. In the same vein $\bigcup F := A_1 \cup \cdots \cup A_n$ is the smallest set that contains every element that occurs in some A_i that belongs to F. Alternative notations are $\bigcap_1^n A_i$ or $\bigcap_{1 \leq i \leq n} A_i$, and similarly for $\bigcup$. Irrespective of whether F is indexed one can always fall back to the notational conventions $\bigcup_{A \in F} A := \bigcup F$ and $\bigcap_{A \in F} A := \bigcap F$. This can be used to define sets recursively.

Example A.7 Constructing sets via union and intersection

Recall that $\wp(\mathbb{N})$ is the power set of $\mathbb{N}$, i.e. the smallest set that contains every subset of $\mathbb{N}$. It is easy to see that $\bigcup \wp(\mathbb{N}) = \mathbb{N}$. Obviously $\bigcup \wp(\mathbb{N}) \subseteq \mathbb{N}$, since it is a union of subsets of $\mathbb{N}$, and those in turn can only contain members of $\mathbb{N}$. At the same time it cannot be the case that $\bigcup \wp(\mathbb{N}) \subset \mathbb{N}$, for then there would be some number $n \in \mathbb{N}$ that is not contained in any subset of $\mathbb{N}$, which is a contradiction. It should also be clear that $\bigcap \wp(\mathbb{N}) = \emptyset$ since no element of $\mathbb{N}$ is contained by every subset of $\mathbb{N}$.

A more interesting application is furnished by DP coordination in any natural language of your choice. Let DP_1 be the set of non-recursive DPs (containing, among

others, *Mary* and *these boys*, but not *the destruction of a city* or *a picture of Mary*). Furthermore, $DP_i := \{d \textit{ and } d' \mid d, d' \in DP_{i-1}\}$ for all $i \geq 2$. That is to say, DP_i is obtained by picking elements of DP_{i-1} and coordinating them with *and*. Then $\bigcup_{i>1} DP_i$ is the set of all non-recursive DPs and coordinations involving such DPs. It includes both *Mary* and *Mary and Mary and the boy*, as well as infinitely many others. The reader is invited to give a similar definition for the set of center-embedding sentences as in *That that John surprised me surprised me surprised me*.

Pairs and Tuples The definition of set-theoretic equivalence implies that sets are unordered: $\{a, b\} = \{b, a\}$ because both $\{a, b\} \subseteq \{b, a\}$ and $\{b, a\} \subseteq \{a, b\}$. A set $\{a, b\}$ with an order defined on its elements is an *ordered pair* such that $\langle a, b\rangle = \langle c, d\rangle$ iff $a = c$ and $b = d$. Consequently, $\langle a, b\rangle$ and $\langle b, a\rangle$ are distinct unless $a = b$. The generalization of an ordered pair to n many elements is called an *n-tuple*, or equivalently, a *sequence of length n*. For $n = 1$, one obtains 1-tuples such as $\langle a\rangle$, while there is only one tuple of length 0, the *empty tuple* $\langle\rangle$. The elements of tuple are also called *components*. A tuple's number of components is identical to its length.

Tuples can be formalized in various ways. They may be defined recursively in terms of pairs such that $\langle a_1, \ldots, a_n\rangle = \langle a_1, \langle a_2, \ldots, a_n\rangle\rangle$. A slightly more general definition uses *concatentation* of sequences. For any two tuples $\langle a_1, \ldots, a_i\rangle$ and $\langle a_{i+1}, \ldots, a_n\rangle$, their concatenation is given by $\langle a_1, \ldots, a_i\rangle \cdot \langle a_{i+1}, \ldots, a_n\rangle := \langle a_1, \ldots, a_i, a_{i+1}, \ldots, a_n\rangle$. Given some fixed set A, then, t is a tuple over A iff $t \in \bigcup_{i\geq 1} T_i^A$, where $T_1^A := \{\langle\rangle\} \cup \{\langle a\rangle \mid a \in A\}$ and $T_i^A := \left\{u \cdot v \mid u, v \in T_{i-1}^A\right\}$. That is to say, the empty tuple is a tuple, every 1-tuple is a tuple, and so is every tuple that can be built by concatenating tuples. This definition slightly differs from the previous one in two respects. First, it allows for the empty tuple $\langle\rangle$, which may be concatenated with any tuple t without changing it: $t \cdot \langle\rangle = \langle\rangle \cdot t = t$. Second,

it furthermore allows for tuples of length 1. However, since every singleton set is trivially ordered, whether one explicitly allows 1-tuples or not has no interesting consequences.

Example A.8 Pairs and tuples in linguistics

Pairs and tuples are commonly encountered in semantics, where they are used to indicate the semantic type of a constituent. For example, sentential negation could be subscripted by $\langle t, t\rangle$ to indicate that it maps truth values to truth values. Sentential coordination, on the other hand, would have the type $\langle t, t, t\rangle$ as it takes two truth values as arguments and returns another truth value. Sometimes the triple is instead decomposed into a pair $\langle\langle t, t,\rangle, t\rangle$.

Non-privative features, i.e. features that can be valued, can also be analyzed as pairs $\langle f, v\rangle$, where f is the name of the feature and v its value. As discussed in Sec. 1.2.1, the features of Minimalist grammars are triples rather than pairs as they also keep track of whether a feature must be checked by Merge or Move. So rather than, say, $\langle \text{case}, +\rangle$ one would have $\langle \text{case}, \textit{move}, +\rangle$.

The string yield of a phrase structure tree can also be viewed as a tuple so that *John kissed Mary* would correspond to the triple $\langle \textit{John}, \textit{kissed}, \textit{Mary}\rangle$. Each word, in turn, could be treated as a tuple of phonemes or graphemes, e.g. $\langle J, o, h, n\rangle$. The verb could also be morphologically decomposed into $\langle \textit{kiss}, \textit{ed}\rangle$ to indicate that it was obtained by a morphological operation that concatenated the 1-tuple $\langle \textit{kiss}\rangle$ with the 1-tuple $\langle \textit{ed}\rangle$. Quite generally, if T is a tree $[_T\ A\ B\ ]$ and A and B have the string yields $\langle a_1, a_2, a_3\rangle$ and $\langle b_1, b_2\rangle$, respectively, then the string yield of t is $\langle a_1, a_2, a_3\rangle \cdot \langle b_1, b_2\rangle = \langle a_1, a_2, a_3, b_1, b_1\rangle$.

Cartesian product Tuples are related to the *Cartesian product* of sets. For arbitrary sets A and B, $A \times B := \{\langle a, b\rangle \mid a \in A, b \in B\}$. Hence $\langle a, b\rangle = \{a\} \times \{b\}$. In the

special case where $A = B$, one often writes A^2 instead of $A \times A$. This notion is easily generalized such that $A^n := A \times A^{n-1}$ and $A^0 := \langle\rangle$. So an n-tuple can also be viewed as an element of A^n for some suitably chosen A.

Example A.9 Cartesian products for linguists

Cartesian products are usually used when one wants to define a general template for pairs without worrying about the specifics of how components may be combined. Above we treated syntactic features as pairs of feature names f and values v. Probably not all values can combine with all feature names, as, say, person might take a numerical value and case a binary-valued one. Still all features will be members of the Cartesian product of feature names and values: $Feat \subseteq Name \times Value$. Suppose $Name := \{\text{number}, \text{case}\}$ and $Value := \{\text{sg}, \text{pl}, +, -\}$. Then $Name \times Value$ yields the following set of pairs:

$$\left\{ \begin{array}{ll} \langle \text{number}, \text{sg} \rangle, & \langle \text{case}, \text{sg} \rangle, \\ \langle \text{number}, \text{pl} \rangle, & \langle \text{case}, \text{pl} \rangle, \\ \langle \text{number}, + \rangle, & \langle \text{case}, + \rangle, \\ \langle \text{number}, \text{-} \rangle, & \langle \text{case}, \text{-} \rangle \end{array} \right\}$$

The set *Feat* of features arguably only contains 4 of those elements.

We may also naively think of sentences as a pair consisting of a surface string π and a logical form λ. Clearly there are dependencies on which surface strings can be matched up with which logical form, so the set of linguistically possible combinations is a proper subset of the Cartesian product of linguistically possible surface strings and linguistically possible logical forms.

Pairs as sets (Kuratowski) As the reader can see, there are many different ways of formalizing the intuitive notion of pairs and tuples. In fact, he or she may actually

be familiar with a purely set-theoretic interpretation of pairs, according to which $\langle a, b\rangle := \{\{a\}, \{a, b\}\}$. This definition (note that it is not an equivalence!) has attracted considerable interest from syntacticians due to its similarities to the bare phrase structure encoding of phrase markers in Chomsky (1995a). At the same time there seems to be some confusion about why pairs can be represented as such sets and what the implications are for linguistics. Addressing these issues here would take us too far (and requires some mathematical maturity that cannot be presupposed at this point), but the interested reader is referred to Sec. A.5.

A.3 Relations and Functions

Binary relations Tuples make it easy to talk about relations that hold between members of a set. A *binary relation* R over sets A and B is a triple $R := \langle A, B, G\rangle$, where $G \subseteq A \times B$ is the *graph* of R. It is important to realize that defining relations as triples is merely a succinct way of stating that they consist of three easily identified components. Under no circumstances should one actually think of them as ordered sets of cardinality 3, which is hardly insightful and distracts from the truly relevant aspects. For example, it would be nonsensical to concatenate two relations $R := \langle A, B, G\rangle$ and $S := \langle C, D, H\rangle$ into a 6-tuple $\langle A, B, G, C, D, H\rangle$. The connection between tuples and relations stems from the fact that the graph of a binary relation is a set of pairs, not that they are defined as triples.

Let us consider these components, then. The sets A and B are the *domain* and *co-domain* of R, respectively, also written $\mathrm{dom}(R)$ and $\mathrm{codom}(R)$. We do not put any restrictions on these sets; they need not be finite, nor do they have to be distinct. The third component is the graph of the relation. It is simply a set-theoretic encoding of which elements are related to each other by R. Relations are often equated with their graph so that they can be treated notationally like sets rather than triples. Hence it suffices to write $\langle a, b\rangle \in R$ to express that a is R-related to b. This is an innocent

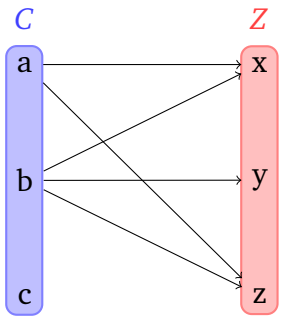


Figure A.2: Graphical representation of the relation S with domain $C := \{a, b, c\}$, co-domain $Z := \{x, y, z\}$ and graph $G := \{\langle a, x\rangle, \langle a, z\rangle, \langle b, x\rangle, \langle b, y\rangle, \langle b, z\rangle\}$

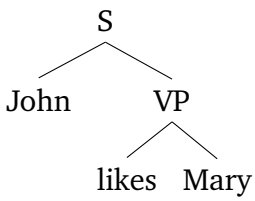
simplification without negative repercussions in practice.

Example A.10 Relations and their graphs

Let $A := \{a, b\}$, $Z := \{x, y, z\}$ and $C := A \cup \{c\}$. Now consider the relations $R := \langle A, Z, G\rangle$ and $S := \langle C, Z, G\rangle$, where $G := \{\langle a, x\rangle, \langle a, z\rangle, \langle b, x\rangle, \langle b, y\rangle, \langle b, z\rangle\}$. Strictly speaking R and S are distinct relations since they differ on their domain, but since they both have the same graph G they are usually regarded as the same relation. They both relate a to x and z but not to y, whereas b is related to all elements of Z. A graphical depiction of S (and thus R, too, implicitly) is provided in Fig. A.2.

Example A.11 Some linguistic relations

Dominance and precedence are commonly encountered relations in syntax. Consider the simplified tree below.



Since dominance relates nodes of the tree to other nodes of the tree, its domain and co-domain are identical and contain all the nodes of the tree. Its graph consists of the pairs $\langle \mathrm{S}, \mathrm{John} \rangle$, $\langle \mathrm{S}, \mathrm{VP} \rangle$, $\langle \mathrm{S}, \mathrm{likes} \rangle$, $\langle \mathrm{S}, \mathrm{Mary} \rangle$, $\langle \mathrm{VP}, \mathrm{likes} \rangle$, and $\langle \mathrm{VP}, \mathrm{Mary} \rangle$. The precedence relation also holds only between nodes of the same tree, so once more domain and co-domain are identical and cover the whole tree. Its graph consists at least of $\langle \mathrm{John}, \mathrm{VP} \rangle$ and $\langle \mathrm{likes}, \mathrm{Mary} \rangle$. If one takes precedence to extend beyond siblings, then the graph also contains $\langle \mathrm{John}, \mathrm{likes} \rangle$ and $\langle \mathrm{John}, \mathrm{Mary} \rangle$. Usually, though, these pairs are considered redundant as they can easily be inferred from the dominance graph and the precedence relations between siblings.

Syntactic sugar for relations Given a binary relation R, it is often preferable to use infix notation and write $a\ R\ b$ instead of $\langle a, b \rangle \in R$ (which strictly speaking is already a shorthand for saying that $\langle a, b \rangle$ is contained in the graph of R). Instead of $\langle a, b \rangle \notin R$ one may similarly write $a \not R\ b$. Furthermore, $aR := \{b \mid a\ R\ b\}$, i.e. the set of elements that a is R-related to, and $Rb := \{a \mid a\ R\ b\}$, i.e. the set of elements that are R-related to b. Of particular interest is the *range* of a relation, given by $\mathrm{ran}(R) := \bigcup_{a \in A} aR$. Note that a relation's range is not necessarily identical to its co-domain, although it is always a subset of it.

Example A.12 Domain, co-domain and range

Suppose that for $S := \langle C, Z, G \rangle$ as in example A.10, the co-domain Z is widened from $\{x, y, z\}$ to $\{u, x, y, z\}$. In this case $\bigcup_{a \in C} aS = \{x, y, z\} \subset \{u, x, y, z\}$.

In example A.11 we saw that the co-domain of the dominance relation contains all nodes in the tree. Its range, however, is a proper subset because no node dominates the root node S. However, if one assumes that every node dominates

itself (also called reflexive dominance), then the range of the dominance relation is identical to its co-domain.

Operations on relations Just as with sets, there are several operations to manipulate and combine relations. Given two relations R and S, their *union* is $R \cup S := \{\langle a, b\rangle \mid a\ R\ b \text{ or } a\ S\ b\}$, whereas their *intersection* $R \cap S := \{\langle a, b\rangle \mid a\ R\ b \text{ and } a\ S\ b\}$. Hence the union and intersection of two relations are equivalent to the set-theoretic union and intersection of their graphs, respectively. If one wants to be perfectly precise and also compute the new domain and co-domain, these are obtained by taking the union/intersection of the domains and co-domains of R and S. The *composition of R with S* is $R \circ S := \{\langle a, c\rangle \mid a\ R\ b \text{ and } b\ S\ c\}$. The domain and co-domain of $R \circ S$ are identical to the domain of R and co-domain of S, respectively. The *inverse of* R is $R^{-1} := \{\langle b, a\rangle \mid a\ R\ b\}$. Here domain and co-domain of the new relation are equivalent to, respectively, the original co-domain and domain. If R is a relation over domain A and co-domain B, its *complement* is $\overline{R} := \{\langle a, b\rangle \in A \times B \mid a \not R\ b\}$. Note that $\overline{R}$ is well-defined only if the domain and co-domain of R are known. This is one of the few cases where relations with identical graphs may behave differently.

Example A.13 Combining the relations R and S

Consider once more the relations $R := \langle A, Z, G\rangle$ and $S := \langle C, Z, G\rangle$, where $A := \{a, b\}$, $C := A \cup \{c\}$, $Z := \{x, y, z\}$, and $G := \{\langle a, x\rangle, \langle a, z\rangle, \langle b, x\rangle, \langle b, y\rangle, \langle b, z\rangle\}$. Then $R \circ S = R \circ R = \emptyset$, since the domain and range of the relations are disjoint. However, composing R with $R^{-1} := \{\langle x, a\rangle, \langle x, b\rangle, \langle y, b\rangle, \langle z, a\rangle, \langle z, b\rangle\}$ yields $R \circ R^{-1} := \{\langle a, a\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle b, b\rangle\}$. As we can see $R \circ R^{-1}$ has A as its domain, co-domain, and range, so $\overline{R \circ R^{-1}} := \emptyset$. But $S \circ S^{-1}$, which has the same graph as $R \circ R^{-1}$,

has C as its domain and co-domain and A as its range. Since the domain and co-domain are different from before, we expect the complement to be different, too, and indeed $\overline{S \circ S^{-1}} := \{\langle a,c\rangle, \langle b,c\rangle, \langle c,a\rangle, \langle c,b\rangle, \langle c,c\rangle\}$. Keep in mind that in general the complement of the composition of two relations is different from the composition of their complements. This is witnessed by $\overline{R} \circ \overline{R^{-1}} := \{\langle a,y\rangle\} \circ \{\langle y,a\rangle\} = \{\langle a,a\rangle\}$.

Properties of binary relations There are several properties that are commonly used to classify binary relations. Many of them have also found their way into the linguistic literature, in particular with respect to the Linear Correspondence Axiom (Kayne 1994).

reflexive For all x, $x\ R\ x$ holds.

irreflexive For all x, $x \not R\ x$ holds.

symmetric For all x and y, $x\ R\ y$ implies $y\ R\ x$.

asymmetric For all x and y, $x\ R\ y$ implies $y \not R\ x$.

antisymmetric For all x and y, $x\ R\ y$ and $y\ R\ x$ jointly imply $x = y$.

transitive For all x and y and z, $x\ R\ y$ and $y\ R\ z$ jointly imply $x\ R\ z$.

total For all x and y, $x\ R\ y$ or $y\ R\ x$.

Example A.14 Properties of various relations over trees

The dominance relation is usually not taken to be reflexive. This by itself does not entail that it is irreflexive since it might be the case that some but not all nodes

dominate themselves. However, this option isn't entertained by linguists either, so dominance is indeed irreflexive. It is also asymmetric and transitive.

Reflexive dominance, i.e. the variant of dominance where every node dominates itself, is reflexive, as the name indicates. It is also transitive, but antisymmetric rather than asymmetric. This means that dominance is still asymmetric for distinct nodes. So antisymmetry is a weakened version of asymmetry that allows for cases of reflexivity.

The mother-of relation (also called immediate dominance) is irreflexive, asymmetric, but not transitive — while multi-dominance trees allow for the mother of the mother of node x to be a mother of x, this is usually not the case for all nodes in a multi-dominance tree.

C-command is irreflexive (by definition, no node c-commands itself), and transitive. Under its canonical definition, which does not incorporate the distinction between segments and categories of Chomsky (1986), it is neither symmetric, nor asymmetric or antisymmetric. C-command is symmetric between siblings, but asymmetric otherwise. As it fails all three variants of symmetry, it is non-symmetric.

None of the relations above are total. Dominance, for instance, does not hold between *John* and *Mary* in the tree in example A.11. An example of a total relation over trees is reachability such that x is reachable from y iff one can reach y from x by moving up or down along dominance branches. Since all nodes of a tree are connected to the root via dominance, it follows that every node can be reached from everywhere in the tree. On the other hand, a stricter notion of reachability that forbids moving upwards along dominance branches would once again fail totality.

The properties listed above are mostly independent of each other, except that being asymmetric is equivalent to being both irreflexive and antisymmetric. Ir-

reflexivity follows from asymmetry since x and y in the definition of asymmetry may be identical and $x\ R\ x$ implying $x \not R\ x$ is a contradiction, so there cannot be any x for which $x\ R\ x$ holds. Antisymmetry, on the other hand, follows trivially because asymmetry does not allow for both $x\ R\ y$ and $y\ R\ x$ to hold at the same time, so whether $x = y$ is irrelevant. Now consider the case where a relation is both irreflexive and antisymmetric. Antisymmetry tells us that asymmetry may be violated only if x and y are identical. But this case is already blocked by irreflexivity, so asymmetry holds after all.

In addition, every total relation must be reflexive. This is due to the fact that x and y in the definition of totality are not stipulated to be distinct. For $x = y$ totality hence reduces to "for all x, $x\ R\ x$", which is exactly the definition of reflexivity.

Contrary to common intuition, no further interdependencies hold. Reflexivity and irreflexivity are strictly speaking not mutually exclusive because there are infinitely many relations that are both reflexive and irreflexive. As a matter of fact, these relations satisfy all the properties above. The definition of this class of relations is left as an exercise to the reader. For relations that do not belong to this slightly pathological class, reflexivity and irreflexivity are incompatible, as are symmetry and asymmetry.

Orders Orders are binary relations with identical domain and co-domain that satisfy specific properties. For example, a weak partial order is a binary relation that is reflexive, antisymmetric and transitive. Table A.1 gives an overview of the most common types of binary relations and which of the properties above they must satisfy. The most commonly encountered orders are equivalence relations, weak partial orders and total orders, but that is not to say that the other types of relations are irrelevant to linguistics. Preorders, for example, play a crucial role in the formalization of hyperintensional semantics (Fox and Lappin 2005; Plummer and Pollard 2012). Note that for strict total orders totality is weakened to trichotomy:

	refl.	irrefl.	symm.	asymm.	antisymm.	trans.	total
preorder	✓					✓	
equivalence relation	✓		✓			✓	
(weak) partial order	✓				✓	✓	
strict partial order		(✓)		✓	(✓)	✓	
total/linear order	(✓)				✓	✓	✓
strict total order		(✓)		✓	(✓)	✓	∼

Table A.1: Important types of binary relations by their properties; implied properties are put in brackets

for all x and y, it holds that $x = y$ or $x\ R\ y$ or $y\ R\ x$.

A relation may satisfy more properties than required in order to be counted among a specific type, so that, say, an equivalence relation is also a preorder because all properties that hold of the latter also hold of the former; in linguistic parlance, there are no scalar implicatures when classifying relations.

Example A.15 Tree relations as orders

Going back to example A.14, we see that dominance is a strict partial order, whereas reflexive dominance is a weak partial order. The mother-of relation does not qualify as any of the standard orders because of its lack of transitivity. C-command does not constitute an order either due to its non-symmetry. *Modulo* the cases of symmetric c-command between siblings, however, c-command is a strict partial order, just like dominance. Reachability does not fit into any type of order because it is total and symmetric. But if one discards all instances of reflexivity, reachability turns out to be a strict total order. Precedence in the sense of Kayne (1994) is also a strict total order, the domain and co-domain of which are the leafs of a given tree.

***n*-ary relations** So far we have only considered binary relations. Recall that the graph of a binary relation is a set of ordered pairs. Generalizing from ordered pairs to n-tuples, we obtain *n-ary relations*.

Example A.16 Addition as a ternary relation

Addition of natural numbers can be regarded as a ternary relation *Add* such that $\langle a, b, c\rangle \in Add$ iff $a + b = c$. Note that only the last element of the tuple belongs to the co-domain of the relation. The alternative interpretation in which a is related to both b and c by *Add* is not available and would instead be represented by $\langle a, b\rangle, \langle a, c\rangle \in Add$.

Just like pairs could also be generalized "downwards", to 1-tuples and the empty sequence, there are also unary and nullary relations. Both are slightly impoverished, though, because they merely define sets. The intuitive reasoning is as follows: since order is irrelevant for 1-tuples, one may equate $\langle a\rangle$ with a for any choice of a, wherefore the graph $\{\langle a\rangle \mid a \in A\}$ of a (total) relation R over some set A can be treated like $\{a \mid a \in A\} = A$. Since we also conflate relations with their graphs when convenient, unary relations define sets. Following this line of reasoning we can also conclude that all nullary relations define the empty set. If we do not distinguish relations with identical graphs but distinct domains or co-domains, there is only one nullary relation (since there is only one empty set).

Example A.17 Labels as unary relations

Let R_{VP} be the unary relation with graph $\{\langle n\rangle \mid \text{node } n \text{ is labeled VP}\}$. This relation can be equated with the set of nodes that are labeled VP. Hence given a tree t with labels drawn from some fixed set Σ, $\bigcup_{\sigma\in\Sigma} R_\sigma$ is the set of all nodes in t.

In sum, an n-ary relation is a triple $\langle A_1 \times \cdots \times A_{n-1}, A_n, G\rangle$, where $G \subseteq A_1 \times \cdots \times A_n$, and as before relations may be equated with their graph, so domain and co-domain need not be specified.

Functions Functions are a special subtype of relations. A relation is a *function* (also called *map* or *mapping*) iff it is right-unique: for all x and y and z, $x\ R\ y$ and $x\ R\ z$ jointly imply $y = z$. Hence a function is a relation that associates every element in its domain to at most one element in its co-domain.

Example A.18 Functions for trees and strings

One might think of a tree as a set of nodes that is paired with a labeling function. The function maps every node to some label. Crucially, every node is assigned no more than one label.

The same strategy works for strings. So rather than treating strings as tuples, we can view them as a totally ordered set of nodes with a labeling function. Take the string $abbca$. It can be formally represented as a triple $\langle D, \leq, \ell \rangle$ where $D := \{1, 2, 3, 4, 5\}$, $a \leq b$ iff a is not greater than b, and ℓ maps 1 to a, 2 to b, 3 to b, 4 to c, and 5 to a.

Just like relations, functions may have arbitrary arity. But since the last element of an n-tuple in the graph of a function can always be inferred from the other elements thanks to right-uniqueness, it is by convention not included in the arity count. So an n-ary function is actually an $(n+1)$-ary relation. This is also in line with the procedural intuition that an n-ary function f takes n arguments $a_1, \ldots, a_n$ and returns an output o. One also says that f *maps* $a_1, \ldots, a_n$ to o.

Notationally, functions differ only slightly from relations. They are often given in the form $f : A_1 \times \cdots \times A_n \to B$, where f is the name of the n-ary function, and $A_1 \times \cdots \times A_n$ and B its domain and co-domain, respectively. In this case one also says that f is a function *from* $A_1 \times \cdots \times A_n$ *into* B. One writes $f(x) = y$ or $x \mapsto y$ to indicate that all x are mapped to y, where y is some specification. For instance, $f : \mathbb{N} \to \mathbb{N}$ with $x \mapsto x + 1$ maps each natural number to its successor. Given a function $f : A \to B$, the range of f is also called the *image of (A under)* f. The

preimage of f is the image of its inverse f^{-1}.

Example A.19 A formalized labeling function

The labeling function from the previous example can now be represented more formally as $\ell : D \to \{a, b, c\}$, with $\ell(1) = a$, $\ell(2) = b$, $\ell(3) = b$, $\ell(4) = c$, and $\ell(5) = a$. Its inverse ℓ^{-1} relates elements of $\{a, b, c\}$ to members of D. Note that ℓ^{-1} is not a function because a is related to both 1 and 5, and b to 2 and 3, so ℓ^{-1} is not right-unique. However, it can be turned into a function by changing its co-domain from D to $\wp(D)$. That is to say, symbols are mapped to sets of nodes. Formally, $\ell' : \{a, b, c\} \to \wp(D)$ with $\ell'(a) = \{1, 5\}$, $\ell'(b) = \{2, 3\}$, and $\ell'(c) = \{4\}$.

All the usual operations such as union, complementation and composition that apply to relations can also be used on functions. One must be careful, though, because there is an important notational difference between functions and relations regarding composition. While $R \circ S$ is the result of "feeding the output of R into S" and thus can be viewed as applying S after R, the order of interpretation is reversed for functions: $(f \circ g) = f(g(x))$. Thus $f \circ g$ means "feeding the output of g into f", which is also why it is often read as f *after* g. Changing the order of the arguments in composition makes sense if one considers functions in isolation, since $f \circ g$ then corresponds to $f(g(x))$ rather than $g(f(x))$, but it creates unfortunate confusion as soon as both relations and functions are discussed; the reader has to pay extra attention to the type of objects being composed.

Properties of functions A function is *partial* if there is some element in its domain that is not related to any element of its co-domain. Functions that aren't partial are *total*. This notion of totality should not be confused with the identically named property of orders. A function f is *one-to-one* (or *injective*) if $x \neq y$ implies $f(x) \neq f(y)$. This is equivalent to requiring that f^{-1} is a function. If f isn't one-to-one it

is *many-to-one*. It is *onto* (or *surjective*) if its co-domain and its range are identical. We call f *bijective* iff it is both one-to-one and onto. A function is *constant* iff the cardinality of its range is at most 1.

Example A.20 Analyzing the labeling and yield functions

We already encountered the labeling function $\ell : D \to \Sigma$, where D is the set of nodes of a tree and Σ some fixed set of labels. As every node in a tree is assigned a label, this function is always total. Whether the labeling function is injective, surjective, both, or neither varies between trees. For the tree in the previous example, it is surjective because every element of Σ is a label of some node. It is also many-to-one as both a and b are labels for multiple nodes. Its inverse $\ell^{-1} : \Sigma \to \wp(D)$ is total and injective.

The yield function maps a tree to its string yield (i.e. the string obtained by reading the leafs of the tree from left to right). Formally, $\text{yd} : T_\Sigma \to \Sigma^*$, where T_Σ is the set of all (finite) trees with labels from Σ, and Σ^* is the set of all strings that can be built by concatenating zero or more symbols from Σ. The first question is whether this function is total or partial. In order for it to be total, there has to be some tree whose string yield is not contained in Σ^*. But Σ^* contains all strings over Σ, and T_Σ only contains trees with labels drawn from Σ, so every tree's string yield must be contained in Σ^*. Hence yd is total. Is yd surjective? The answer is yes, because every string can easily be turned into a tree by adding a finite number of non-terminal nodes, and T_Σ contains every tree, including those obtained from strings by adding nodes in this fashion. This entails that for every string s there is some tree t such that $\text{yd}(t) = s$. But yd is not injective because many trees may have the same string yield.

Since partial functions are not as well-behaved as total functions, it is often important to know whether a given function is allowed to be partial. Unfortunately,

this is not always mentioned explicitly. In the mathematical literature, that usually means that the function is allowed to be partial. In computer science, on the other hand, functions are assumed to be total unless they are explicitly allowed to be partial.

Some important functions A *closure operator* is a function from sets to sets. Closing a set S under p means to map it to its smallest superset that satisfies p. Formally, closure$(S,p) := \bigcap \{S' \supseteq S \mid \text{S' satisfies } p\}$. S is *closed under* p iff closure$(S,p) = S$.

Example A.21 Taking closures

The transitive closure of a relation R is obtained by adding pairs to R until it satisfies transitivity. More precisely, if $x\ R\ y$ and $y\ R\ z$, then the pair $\langle x,z\rangle$ is added to R. The mother-of relation, for instance, has the dominance relation as its transitive closure. The sibling permutation closure S of a set of trees adds new trees to the set such that if t is a member of S, then so is every tree that only differs from t with respect to the order of some siblings. So $[_A\ B\ C] \in S$ iff $[_A\ C\ B] \in S$.

Often one is less interested in taking the closure of some set than expressing that the set is (not) closed under some operation. The natural numbers are closed under addition since for any choice of x and y $\mathbb{N}$, $x+y$ is also in $\mathbb{N}$. But they are not closed under subtraction because $x-y$ is not necessarily a natural number, e.g. for $5-8$. The set of English strings is not closed under concatenation, as it contains both *I ran* and *John likes Mary* but not *I ran John likes Mary*. Nor is it closed under string reversal, since *ran I* isn't part of English either.

In a sense, closure operators are the opposite of a particular type of functions called restrictions. Given a set S and an n-ary relation R with graph G, the *restriction of R to S* is a function that maps G to $R \restriction S := G \cap S^n$.

Example A.22 Restrictions of relations

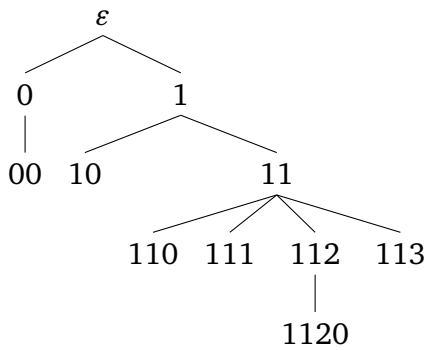
Let N be the set of non-terminal node labels, e.g. S, VP, NP, and so on. Furthermore, let D be the dominance relation for the tree encountered in example A.11 on page 280. Then $D \restriction N$ is the singleton set $\{\langle \mathrm{S}, \mathrm{VP}\rangle\}$. All other pairs in the graph of D contain terminal node labels and thus are not contained in $N \times N$. Hence they cannot be in the restriction of D to N, either.

The *i-th projection* π_i is a (partial) function that maps an n-tuple to its i-th component, provided $i \leq n$. So $\pi_2(\mathrm{S}, \mathrm{VP}) = \mathrm{VP}$. In this case one might also call VP a projection of $\langle \mathrm{S}, \mathrm{VP}\rangle$. The inverse of a projection is called *cylindrification*.

Defining trees Each node in a tree can be associated with a unique numeric address, represented as a tuple. The root node has address $\langle\rangle$ (also written ε), and the daughter of a node with address α and i-many left siblings is assigned $\alpha \cdot \langle i \rangle$.

Example A.23 Tree node addresses

Each node in the tree below is labeled by its address.



The node addressing system allows for a very elegant definition of trees. First, a

(Gorn) tree domain (Gorn 1967) is a *hereditarily prefix closed* set D of node addresses, that is to say, it obeys the following two conditions:

dominance prefix closure For every $d \in D$ with $d := \alpha i$, it holds that $\alpha \in D$.

precedence prefix closure For every $d \in D$ with $d := \alpha i \neq \alpha 0$, $\alpha(i-1) \in D$.

Example A.24 Gorn tree domains

The set of node addresses used in the previous example is a tree domain. The set $\{0\}$ is not, because it contains 0 but not ε, which violates dominance prefix closure. Similarly, $\{\varepsilon, 1\}$ is not a Gorn tree domain because it has 1 as a member but not 0, so precedence prefix closure is not satisfied. But $\{\varepsilon, 0\}$ is a Gorn tree domain, as are $\{\varepsilon\}$ and $\emptyset$.

A Σ-*labeled tree* is a pair $T_\Sigma := \langle D, \ell \rangle$ such that D is a tree domain and $\ell : D \to \Sigma$ labels every node with some symbol drawn from Σ. Note that in contrast to other definitions, we do not need to say anything about dominance and precedence as they are already implicit in the fact that D is a tree domain. Hence x is the mother of y ($x \triangleleft y$) iff $y = xi$ for some digit i, and x is the left sibling of y iff $x = \alpha i$ and $y = \alpha(i+1)$. The reader is invited to compare this simple definition with the standard graph-theoretic one (cf. Partee et al. 1990), which requires additional stipulations such as an explicit ban against tangling branches.

A.4 Formal Logic

Propositional logic A particular kind of formal logic — called *monadic second-order logic* — features prominently in this thesis. A formal logic consists of two parts: a procedure for generating the set of well-formed expressions, and a function that provides the interpretation of said expressions.

Propositional logic (also called sentential logic) is one of the simplest and most commonly encountered logics. Let $P := p_1, \ldots, p_n, \ldots$ be a (possibly infinite) set of *propositional variables*. The set L of *well-formed formulas* is given by

- $p \in P$ implies $p \in L$,
- for all $p \in L$, $(\neg p) \in L$,
- for all $p, q \in L$, $(p \vee q) \in L$,
- nothing else belongs to L.

Example A.25 Some propositional formulas

Suppose P contains at least p, q, and r. Then L includes, among others, $(p \vee (q \vee r))$, $((\neg p) \vee (\neg q))$, and $(\neg(p \vee (\neg q)))$. It does not contain pq, $(\vee p)$, $\neg(p)$, or $p \vee q$.

The symbols $\neg$ and $\vee$ are *logical connectives* and usually called *negation* or *not* and *disjunction* or *logical or*.

So far well-formed formulas are just meaningless sequences that have been obtained by combining symbols in a particular way. In the next step we define a very rudimentary semantics that assigns every well-formed formula a truth value — 1 for true, 0 for false (alternative notations include T and F as well as $\top$ and $\bot$). The truth value of the formula is computed compositionally from the truth values of the propositional variables. An *assignment* is a total map $g : P \to \{0, 1\}$ from propositional variables to truth values. We extend g to a function $\hat{g}$ from L into $\{0, 1\}$ such that

- for all $p \in P$, $\hat{g}(p) = g(p)$,
- for all $\phi \in L$, $\hat{g}((\neg \phi)) = 1 - \hat{g}(\phi)$,

- for all $\phi, \psi \in L$, $\hat{g}((\phi \vee \psi)) = \max(\hat{g}(\phi), \hat{g}(\psi))$

Here the function max takes two arguments and returns the bigger one of the two. That is to say, $\max(i, j) = i$ iff $i \geq j$. As can be seen, $\neg\phi$ has the opposite truth value of ϕ, whereas $\phi \vee \psi$ is true as long as at least one of the two, ϕ or ψ, is true. So it makes sense that these symbols are called negation and disjunction, respectively.

Example A.26 Computing truth values in propositional logic

Assume $g(p) = 1$ and $g(q) = g(r) = 0$. The truth values of the formulas from the

previous example then are computed as follows:

$$\begin{aligned}
\hat{g}((p \vee (q \vee r))) &= \max(\hat{g}(p), \hat{g}((q \vee r))) \\
&= \max(g(p), \max(\hat{g}(q), \hat{g}(r))) \\
&= \max(g(p), \max(g(q), g(r))) \\
&= \max(1, \max(0, 0)) \\
&= \max(1, 0) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\hat{g}(((\neg p) \vee (\neg q))) &= \max(\hat{g}((\neg p)), \hat{g}((\neg q))) \\
&= \max(1 - \hat{g}(p), 1 - \hat{g}(q)) \\
&= \max(1 - g(p), 1 - g(q)) \\
&= \max(1 - 1, 1 - 0) \\
&= \max(0, 1) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\hat{g}((\neg(p \vee (\neg q)))) &= 1 - \hat{g}((p \vee (\neg q))) \\
&= 1 - \max(\hat{g}(p), \hat{g}((\neg q))) \\
&= 1 - \max(g(p), 1 - \hat{g}(q)) \\
&= 1 - \max(g(p), 1 - g(q)) \\
&= 1 - \max(1, 1) \\
&= 1 - 1 \\
&= 0
\end{aligned}$$

Several additional connectives are used as convenient shorthands.

- *conjunction/and*: $(p \wedge q)$ iff $\neg((\neg p) \vee (\neg q))$
- *implication/implies*: $(p \rightarrow q)$ iff $((\neg p) \vee q)$
- *bi-implication/equivalence*: $(p \leftrightarrow q)$ iff $((p \rightarrow q) \wedge (q \rightarrow p))$

The resulting truth values can be succinctly represented in terms of a truth table as shown in Tab. A.2.

ϕ	ψ	$\neg\phi$	$\phi \vee \psi$	$\phi \wedge \psi$	$\phi \rightarrow \psi$	$\phi \leftrightarrow \psi$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Table A.2: Truth table for the five standard propositional connectives

Seeing how an excess of brackets is detrimental to readability, we adopt certain scope conventions that allows us to drop brackets without rendering the formulas ambiguous. Without brackets, every connective takes the narrowest scope possible, with negation taking precedence over *and* and *or*, which in turn have narrower scope than *implication* and *bi-implication*.

Example A.27 Dropping brackets

The formula $((\neg p) \vee (\neg q))$ can be simplified to $\neg p \vee \neg q$. The outer brackets are irrelevant for scope, and by convention negation has a narrower scope than logical or, the other pairs of brackets are also redundant. On the other hand, $(\neg(p \vee (\neg g)))$ can only be simplified to $\neg(p \vee q)$, as the negation must scope over the entire disjunction. Similarly, $(((p \wedge (\neg q)) \rightarrow (q \vee r)) \leftrightarrow (p \wedge r))$ reduces to $(p \wedge \neg q \rightarrow q \vee r) \leftrightarrow p \wedge r$. The formula $p \wedge (q \vee r)$ cannot be simplified at all because neither connective is by default preferred over the other, so $p \wedge q \vee r$ is ambiguous between $p \wedge (q \vee r)$ and $(p \wedge q) \vee r$.

Propositional logic has some interesting applications in mathematical linguistics, foremost in phonology (see Rogers et al. 2010 and Rogers and Pullum 2011 for a technical perspective and Heinz 2010 for applications). In the context of this thesis, though, it merely serves as a stepping stone toward more powerful logics.

First-order logic First-order logic extends propositional logic in various ways. Besides variables, there is also the existential quantifier $\exists$, the equality symbol $\approx$, as well as symbols for constants, functions and relations. The latter are called non-logical symbols. The selection of non-logical symbols may vary and is called a *signature*. Relation symbols are also called *predicates*. Rather than discuss first-order logic in its full generality, I restrict myself to the signature that is used in this thesis for the description of trees, and I only give an intuitive sketch of the corresponding semantics.

Our signature contains the symbols $\triangleleft$ and $\triangleleft^+$ — intended to be mother-of and (proper) dominance, respectively — as well as $\prec$ for left-sibling. Note that all these symbols denote binary relations. We also add a dedicated predicate for each label we wish to use. Since labels can be viewed as unary relations, these predicates are unary. We do not need any functions or constants.

Given an infinite set X of variables $x_1, \ldots, x_n, \ldots$, the set L of well-formed formulas is defined inductively:

- If P is a unary predicate, then $P(x) \in L$,
- $x \triangleleft y \in L$,
- $x \triangleleft^+ y \in L$,
- $x \prec y \in L$
- $x_i \approx x_j \in L$,
- for all $\phi \in L$, $(\neg\phi) \in L$,

- for all $\phi, \psi \in L$, $(\phi \vee \psi) \in L$,
- for all $\phi \in L$ and $x \in X$, $\exists x[\phi] \in L$.

The universal quantifier $\forall$ is used as a shorthand such that $\forall x[\phi]$ ("ϕ holds of every x") is equivalent to $\neg\exists[\neg\phi]$ ("$\neg\phi$ holds of no x"). Brackets are still omitted according to the conventions for propositional logic, but I also drop the square brackets from $\exists x[\phi]$ if ϕ is itself a quantified formula. This shortens $\exists x[\exists y[\phi]]$ to $\exists x \exists y[\phi]$.

Example A.28 Some first-order formulas

Among the set of well-formed formulas built with our signature we find $x \triangleleft y$, $\neg\exists x \forall y[x \triangleleft y \vee \neg(y \prec x)]$, $\exists z[\mathrm{VP}(z)] \rightarrow x \approx y$, and even $\exists z[x \triangleleft y]$. Illicit formulas include $\triangleleft x$, $\mathrm{VP} \approx \mathrm{XP}$, and $\exists x \vee \neg\forall x[x \triangleleft y]$.

The semantics of first-order formulas is rather complex. Instead of simply mapping variables to truth values, the formulas are interpreted with respect to specific structures. A structure M is a *model of formula* ϕ iff ϕ is true with respect to M. So we need to clarify what it means to be a structure, and how a formula can be determined to be true in such a relative sense.

A structure is a set S with relations $R_1, \ldots, R_m$ defined over it. In the case of trees, for example, S is a tree domain and the relations are mother-of, dominance, left-sibling, plus one unary relation for each label. An *interpretation* is a function I that maps each element of the signature to a corresponding relation in S. For the tree signature defined above, we only need to match up the symbols with the respective relations. Now let T be a tree with domain D and $g : X \rightarrow D$ an assignment that maps every variable to some node of T. Whether formula ϕ holds with respect to T

under assignment g ($T, g \models \phi$) is computed as follows:

$T, g \models$	$P(x_1, \ldots, x_n)$	iff $\langle g(x_1), \ldots, g(x_n) \rangle \in I(P)$
$T, g \models$	$x_i = x_j$	iff $g(x_i) = g(x_j)$
$T, g \models$	$\neg\phi$	iff ϕ does not hold with respect to I under assignment g
$T, g \models$	$\phi \vee \psi$	iff $T, g \models \phi$ or $T, g \models \psi$
$T, g \models$	$\exists x[\phi]$	iff there is some u such that $T, g[x/u] \models \phi$

If there is some assignment g such that $M, g \models \phi$, then M *satisfies* ϕ, or equivalently, M is a *model* of ϕ. If $M, g \models \phi$ holds for all g, then ϕ is *valid in* M.

A.5 Excursus: Pairs as Sets and Bare Phrase Structure 

As mentioned at the end of Sec. A.2, a particular way of defining ordered pairs in purely set-theoretic terms has earned some notoriety among syntacticians in the wake of the introduction of Bare Phrase Structure in Chomsky (1995a). The definition is due to Kuratowski (1921) and states that $\langle a, b \rangle := \{\{a\}, \{a, b\}\}$. The fascination with the Kuratowski definition seems to stem from the close resemblance to the set-theoretic Bare Phrase Structure representation of a tree $[_a\ a\ b]$ as the set $\{a, \{a, b\}\}$.[1] This has been taken to imply that while Bare Phrase Structure trees are unordered sets, it takes only a minor modification of the Merge operation to linearize them.

This view, however, treats the relation between tuples and sets of this peculiar form as a mathematical theorem, rather than a definition. In contrast to theorems, definitions cannot be evaluated in terms of veridicality but only with respect to

[1] It is unclear whether the Bare Phrase Structure representation of $[_a c[_a a b]]$ should be $\{a, \{c, \{a, \{a, b\}\}\}\}$ or $\{\{a, \{a, b\}\}\{c, \{a, \{a, b\}\}\}\}$. The latter format is the one obeying the Kuratowski definition whereas the former is closer to the idea that the label of a tree is a projection of its head. If one adopts the projection-based version, the mathematical connections between sets and the Kuratowski definition are no longer relevant, so let us assume for the sake of argument that the second set expression is the correct one.

adequacy and usefulness. In the next few paragraphs, I first explain why these "Kuratowski sets" are an adequate encoding of ordered pairs. Right afterwards, though, I show that Bare Phrase Structure sets are just as adequate, i.e. $\langle a, b\rangle := \{a, \{a, b\}\}$. The upshot is that there is nothing profound or insightful about these correspondences from a linguistic perspective; loosely paraphrasing Quine (1995/1941:5), we should not make the mistake of putting notation over subject-matter.

In order to show that the Kuratowski definition is adequate, one has to demonstrate that it captures the relevant properties of pairs. Remember that we introduced pairs by contrasting them with sets—while $\{a, b\} = \{b, a\}$, $\langle a, b\rangle \neq \langle b, a\rangle$ unless $a = b$. More generally, the defining property of pairs is that $\langle a, b\rangle = \langle c, d\rangle$ iff $a = c$ and $b = d$. Hence this equivalence must be derivable using the Kuratowski definition.

Theorem A.1. For all $\langle a, b\rangle, \langle c, d\rangle \in A^2$, $\langle a, b\rangle = \langle c, d\rangle$ iff $a = c$ and $b = d$.

Proof. Since many readers might not be used to reading proofs, I fully spell out every step. As an equivalence can be decomposed into two implications, it suffices to show that the claim holds for both the *if* and *only if* direction.

RIGHT TO LEFT. If $a = c$ and $b = d$, then $\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\}$. By the Kuratowski definition, then, $\langle a, b\rangle = \langle c, d\rangle$.

LEFT TO RIGHT. If $\langle a, b\rangle = \langle c, d\rangle$ then $\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\}$ according to the Kuratowski definition. We now need to consider two distinct cases, $a = b$ and $a \neq b$.

> CASE 1 ($a = b$). In this case $\{\{a\}, \{a, b\}\} = \{\{a\}, \{a\}\} = \{\{a\}\}$. From our initial assumption that $\langle a, b\rangle = \langle c, d\rangle$, it follows that $\{\{a\}\} = \{\{c\}, \{c, d\}\}$. As sets with distinct cardinalities cannot be identical, $\{\{c\}, \{c, d\}\}$ must also be singleton. Consequently, $\{c\} = \{c, d\}$, whence $c = d$. We thus have $a = b = c = d$, which trivially implies $\langle a, b\rangle = \langle c, d\rangle$.

CASE 2 ($a \neq b$). Our goal is to show that $\langle a, b\rangle = \langle c, d\rangle$ and $a \neq b$ jointly imply $a = c$ and $b = d$. We must again consider two subcases, depending on whether $\{a\} = \{c\}$ or $\{a\} = \{c, d\}$.

> CASE 2.1 ($\{a\} = \{c, d\}$). If $\{a\} = \{c, d\}$ then $a = c = d$ (c and d cannot be distinct as $\{a\}$ and $\{c, d\}$ would then have different cardinalities). But then $\{\{c\}, \{c, d\}\} = \{\{c\}, \{c\}\} = \{\{c\}\} = \{\{a\}\} = \{\{a\}, \{a\}\} = \{\{a\}, \{a, a\}\}$. By our initial assumption that $\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\}$, then, $\{\{a\}, \{a, b\}\} = \{\{a\}, \{a, a\}\}$, so $a = b$. But we also assume that $a \neq b$. This yields a contradiction, so one of our initial assumptions is incorrect. Either $a = b$ after all, which was already covered in the first case, or $\{a\} = \{c\} \neq \{c, d\}$, which is covered in the next one.
>
> CASE 2.2 ($\{a\} = \{c\}$). In this case $a = c$ and $\{a, b\} = \{c, d\}$. It only remains to establish that $b = d$. Suppose towards a contradiction that $a = d$. Then $a = c = d$, whence $\{a, b\} = \{c\} = \{a\}$, which erroneously implies $a = b$. The only valid option, then, is $b = d$. We thus have $a = c$ and $b = d$.

We see that in all consistent cases $a = c$ and $b = d$. This proves the left to right direction. ■

The proof shows that the Kuratowski definition is an adequate encoding of tuples because it preserves their defining property that $\langle a, b\rangle = \langle c, d\rangle$ iff $a = c$ and $b = d$. This holds because the set-theoretic representations $\{\{a\}, \{b, c\}\}$ and $\{\{c\}, \{c, d\}\}$ furnish enough structure so that $\{a\}$ and $\{a, b\}$ must be identical to $\{c\}$ and $\{c, d\}$, respectively. From this it follows that $a = c$ and $b = d$. As one might expect, though, there are other ways of structuring sets so that they indirectly enforce this kind of correspondence — as a matter of fact, there are infinitely many (for instance, one can always put the Kuratowski sets inside another set, and those sets inside yet another set, and so on).

While it is hardly surprising that more elaborate set-theoretic constructions can do the same work as the Kuratowski sets, there is also a simpler one. Intriguingly, this option uses a format that is already familiar from Bare Phrase Structure sets: $\langle a, b\rangle := \{a, \{a, b\}\}$. As before we need to show that this definition is adequate to the extent that $\langle a, b\rangle = \langle c, d\rangle$ iff $a = c$ and $b = d$. Since the proof mostly follows the previous one I now proceed at a slightly brisker pace.

Proof. As before the right to left direction is trivial, so we only cover left to right: $\langle a, b\rangle = \langle c, d\rangle$ entails $a = c$ and $b = d$. By our definition of ordered pairs we have $\{a, \{a, b\}\} = \{c, \{c, d\}\}$. Therefore $a = c$ or $a = \{c, d\}$. It must also be the case that $\{a, b\} = c$ or $\{a, b\} = \{c, d\}$. This leaves us with four different valuations.

CASE 1. Suppose $a = \{c, d\}$ and $\{a, b\} = \{c, d\}$. Substituting for $\{c, d\}$ we have $a = \{a, b\}$, so a is a set containing itself, which is illicit.

CASE 2. Suppose $a = \{c, d\}$ and $\{a, b\} = c$. Then substituting for c we have $a = \{\{a, b\}, d\}$, so a is a set containing a set containing a itself, which is also illicit.

CASE 3. Suppose $a = c$ and $\{a, b\} = c$. Substituting for c we have once more $a = \{a, b\}$, so a is a set containing itself, which is illict.

CASE 4. Suppose $a = c$ and $\{a, b\} = \{c, d\}$. This immediately entails $b = d$.

Among the four possible valuations, only one is internally consistent, namely $a = c$ and $b = d$. This proves the left-to-right direction: $\langle a, b\rangle = \langle c, d\rangle$ entails $a = c$ and $b = d$ under the simpler definition of ordered pairs, too. ■

This proof seems less abstract than the previous one in that the first three cases are all instances of one and the same problem: unless $a = c$ and $b = d$, we run into an infinite regress and wind up constructing sets that contain themselves. So $a = c$ and $b = d$ is the only way of obtaining a well-formed set.

To a mathematician, however, the proof is actually less appealing than the first one because it implicitly requires the *Axiom of Regularity* (also called the *Axiom of Foundation*) in order to rules out the cases where a set contains itself. Without the Axiom of Regularity, nothing prevents the existence of such self-containing sets. Admittedly, the Axiom of Regularity is widely accepted among mathematicians. It is included in almost all axiomatizations of set theory, in particular the three most dominant ones: ZF (Zermelo Fraenkel), ZFC (Zermelo Fraenkel with the Axiom of Choice), and NBG (von Neumann Bernays Gödel). Nonetheless it is preferable to have a definition of ordered pairs that holds for as many variants of set theory as possible, and this is arguably why the Kuratowski definition has historically been preferred over its simpler Bare Phrase Structure-like brethren. There is also a minor issue in that the pair of empty sets $\langle \emptyset, \emptyset \rangle$ is represented as $\{\emptyset, \{\emptyset, \emptyset\}\} = \{\emptyset, \{\emptyset\}\}$, which is also the set-theoretic encoding of the natural number 2. For mathematicians those are good enough reasons to prefer the Kuratowski definition of ordered pairs.

Linguists, on the other hand, have no reason to care whether their definitions are incompatible with certain proposals at the fringes of axiomatic set theory or a particular representation of natural numbers. Both $\{\{a\}, \{a, b\}\}$ and $\{a, \{a, b\}\}$ are equally suitable definitions of $\langle a, b \rangle$, so the Bare Phrase Structure format has nothing to say about linear order in syntax. It can easily support it if so desired, but it does not have to — the connection between pairs and certain sets is a matter of definition, not theorems. It is also unsurprising that Bare Phrase Structure sets can be mapped to ordered pairs: an ordered pair establishes an asymmetry between two elements by having one precede the other, just like Bare Phrase Structure establishes an asymmetry between two siblings via projection of a label. Even if labels are completely dropped there is still the asymmetry between heads and arguments that is encoded through the feature calculus and can be used to define a mapping from sets to pairs.

APPENDIX B

Formal Definitions

Contents

B.1 Alphabets, Strings, Trees

A finite, non-empty set Σ is called an *(unranked) alphabet*. The *Kleene closure* of Σ (also called "Sigma Star") is $\Sigma^* := \bigcup_{i \geq 0} \Sigma^i$ (remember that for any given set A, $A^i := A \times A^{i-1}$). A Σ-*string* is some element of Σ^*. It is of *length* n, written $|s| = n$, iff it is a member of Σ^n. The special string s with $|s| = 0$ is called the *empty string* ε. When the choice of alphabet is clear from context, we simply speak of strings rather than Σ-strings. Since strings are essentially sequences over some fixed alphabet Σ (with

ε corresponding to the empty sequence $\langle\rangle$), the same operations can be applied to them. In particular, strings can be concatenated: given strings $s := \langle \sigma_1, \ldots, \sigma_i \rangle$ and $t := \langle \sigma_{i+1}, \ldots, \sigma_n \rangle$, $s \cdot t := \langle \sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n \rangle$ is their *string concatenation*. For the sake of brevity, one often writes st instead of $s \cdot t$. Note that $s \cdot \varepsilon = \varepsilon \cdot s = s$. Given two Σ-strings $s := \sigma_1 \cdots \sigma_n$ and t, s is a *subsequence* of t iff there are Σ-strings $u_0, \ldots, u_n$ such that $u_0 \cdot \sigma_1 \cdot u_1 \cdot \sigma_2 \cdots u_{n-1} \cdot \sigma_n \cdot u_n = t$. If $u_i = \varepsilon$ for all $1 \leq i < n$, s is a *substring* of t. If furthermore $u_0 = \varepsilon$, then s is a *prefix*, and if $u_n = \varepsilon$, then s is a *suffix*. However, s might be both a prefix and a suffix of t yet $s \neq t$ (consider the case where $s := a$ and $t := aa$).

The *rank function* $\text{rank} : \Sigma \to \mathbb{N}$ associates each $\sigma \in \Sigma$ with some natural number n. We write $\sigma^{(n)}$ iff $\text{rank}(\sigma) = n$. The symbol σ is said to be *of rank/arity* n. In the special case where $n = 0$, σ is called a *frontier label* or *leaf label*. The set of all $\sigma^{(n)} \in \Sigma$ is denoted by $\Sigma^{(n)}$. The set T_Σ of all Σ-*trees* is defined recursively as follows: $\Sigma^{(0)} \subseteq T_\Sigma$, and $\sigma(t_1, \ldots, t_n) \in T_\Sigma$ iff both $\sigma \in \Sigma^{(n)}$ and $t_i \in T_\Sigma$ for all $1 \leq i \leq n$.

B.2 Minimalist Derivation Tree Languages

This section offers a succinct recap of the definition of MDTLs given in Sec. 1.2. A fully model-theoretic definition of MDTLs and the transduction to derived structures is presented in Sec. B.4.

Definition B.1. Let BASE be a non-empty, finite set of *feature names*. Furthermore, OP := {*merge*, *move*} and POLARITY := {+, −} are the sets of *operations* and *polarities*, respectively. A *feature system* is a non-empty set *Feat* ⊆ BASE × OP × POLARITY.

For every feature $f \in \mathit{Feat}$, $\nu(f)$, $\omega(f)$ and $\pi(f)$ are the first, second, and third projection of f, respectively.

Definition B.2. Given a string alphabet Σ and feature system *Feat*, a (Σ, Feat)-*lexicon* is a finite subset of $\Sigma \times \mathit{Feat}^+$.

Definition B.3. Let *Lex* be a $(\Sigma, Feat)$-lexicon, $Lex_\star := \{\sigma :: f_1 \cdots f_n \star \mid \sigma :: f_1 \cdots f_n \in Lex\}$, and Ω the ranked alphabet $\{l^{(0)} \mid l \in Lex\} \cup \{\text{Move}^{(1)}, \text{Merge}^{(2)}\}$. Then the *slice lexicon of Lex* is $\text{slice}(Lex) := \{\zeta(l) \mid l \in Lex_\star\}$, where $\zeta : Lex_\star \to T_\Omega$ is given by

$$\zeta(\sigma :: f_1 \cdots f_i \star f_{i+1} \cdots f_n) := \begin{cases} \sigma :: f_1 \cdots f_n \\ \qquad \text{if } f_1 \cdots f_i = \varepsilon \\ \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n) \\ \qquad \text{if } \pi(f_i) = - \\ \text{Move}(\zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) \\ \qquad \text{if } \omega(f_i) = \textit{move} \text{ and } \pi(f_i) = + \\ \text{Merge}(\square_i, \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) \\ \qquad \text{if } \omega(f_i) = \textit{merge} \text{ and } \pi(f_i) = + \end{cases}$$

Definition B.4. The closure of slice(*Lex*) under tree concatenation is the *free slice language* FSL(*Lex*).

The *slice root* of a slice s is the unique node reflexively dominating every node in s. This extends straight-forwardly to LIs such that n is the slice root of LI l iff it is the slice root of the slice of l. For every $t \in \text{FSL}(Lex)$ and LI l in t with string $-f_1 \cdots - f_n$ of licensee features, the *occurrences* of l in t are defined as below:

- $occ_0(l)$ is the slice root of l in t.
- $occ_i(l)$ is the unique node m of t labeled *move* such that m matches $-f_i$, properly dominates occ_{i-1}, and there is no node n in t that matches $-f_i$, properly dominates occ_{i-1}, and is properly dominated by m.

For every $t \in \text{FSL}(Lex)$, node m of t, and LI l with licensee features $-f_1 \cdots -f_n$, $n \geq 0$:

Final If the slice root of l is the root of t, then the category feature of l is C.

Merge If m is associated to selector feature $=f$, then its left daughter is the slice root of an LI with category feature f.

Move There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i^{th} positive occurrence of l, $1 \leq i \leq n$.

SMC If m is labeled *move*, there is exactly one LI for which m is a positive occurrence.

Definition B.5. A set L of trees is a *Minimalist derivation tree language* iff there is some Minimalist lexicon *Lex* such that L is the largest subset of FSL(*Lex*) whose members all satisfy **Final**, **Merge**, **Move**, and **SMC**.

B.3 Tree Automaton for Derivation Trees

In Sec. 2.1.1 I showed how MDTLs can be recognized by tree automata, building on the work of Kobele et al. (2007). The full definition of this automaton is given in this section, following the construction in Kobele et al. (2007).

Definition B.6. A *bottom-up tree automaton* is a tuple $A := \langle \Sigma, Q, F, \Delta \rangle$ where Σ is a ranked alphabet, Q is a finite, non-empty set of states, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq \bigcup_{n \geq 0} Q^n \times \Sigma^{(n)} \times Q$ is the transition relation. The automaton is A deterministic iff Δ is a function (in which case δ is used to denote the transition function).

Definition B.7. The *state assignment induced by automaton A* is the total function $\rho_A : T_\Sigma \to \wp(Q)$ such that for every tree $t := f(t_1, \ldots, t_n)$, $\rho_A(t) = \Big\{ q \mid (\rho_A(t_1) \times \cdots \times \rho_A(t_n) \times \{f\} \times \{q\}) \cap \Delta \neq \emptyset \Big\}$. A Σ-tree t is *recognized* by A iff $\rho_A(t) \cap F \neq \emptyset$. The tree language L recognized by A is the smallest set that contains all trees recognized by A.

Let G be an MG with $(\Sigma, Feat)$-lexicon Lex. Then $|\gamma|$ is the smallest n such that no LI has more than n positive polarity features. Similarly, $|\delta|$ is the maximum number of licensee features on an LI. Then the MDTL of G is recognized by the deterministic automaton $A_G := \langle \Sigma, Q, F, \delta \rangle$ where

- $Q = \bigcup_{0 \leq n \leq |\gamma| + |\delta| + 1} (Feat^n)^{(|Feat|+1)}$, and
- $F = \Big\{ \langle c \rangle \cdot \langle \varepsilon \rangle^{|Feat|} \Big\}$.

The idea is that each state consists of a fixed number $k = |Feat| + 1$ of components, the first of which contains the string of unchecked features of the current head and all others the unchecked features of a mover. Thanks to the SMC there are at most as many movers as there are distinct feature names, so at most $|Feat| + 1$ components are ever needed. As no LI carries more than the maximum of positive polarity features, the maximum of licensee features and a single category feature, each component is also limited to feature strings of length less than $|\gamma| + |\delta| + 1$.

The transition rules of A mirror the definition of Merge and Move in the chain-based format of Stabler and Keenan (2003). Since we operate with k-tuples where all components but the first one are reserved for strings starting licensee features, it is helpful to reserve each slot for feature strings starting with a specific feature. Assume then w.l.o.g. that *Feat* is an family of feature names with index set $\{i \mid 1 \leq i \leq |Feat|\}$. Furthermore, $\oplus$ is an operator on strings such that $u \oplus v$ is undefined if neither $u = \varepsilon$ nor $v = \varepsilon$. In all other cases $u \oplus v = u \cdot v$. The operator is extended to a function $\oplus$ that takes a state $q := \langle \alpha, \beta_1, \ldots, \beta_k \rangle$ and a feature string α'. If $\alpha' = \varepsilon$, then $\oplus$

simply returns q. If $\alpha' = -f_i\alpha''$, then $\oplus$ returns the state $\langle \alpha, \beta_1, \ldots, \beta_i \oplus \alpha', \ldots, \beta_k \rangle$. In all other cases the function is undefined.

The function $\oplus$ serves two purposes. Since it is undefined if neither u nor v are empty, it ensures that the automaton aborts if we try to merge two states that both have a non-empty i-th component, which enforces the SMC in an indirect way. At the same time it provides an elegant way of pushing a feature string from one component to another after its first feature has been checked. The set of all transition rules can be now expressed by the scheme below. If desired, a second rule may be added for Merge in which the order of the states is switched.

$$\delta(\sigma :: \alpha) = \langle \alpha, \varepsilon, \ldots, \varepsilon \rangle,$$

$$\delta(\text{Merge}, \langle =f\,\alpha, \beta_1, \ldots, \beta_k \rangle, \langle f\,\alpha', \beta'_1, \ldots, \beta'_k \rangle) = \langle \alpha, \beta_1 \oplus \beta'_1, \ldots, \beta_k \oplus \beta'_k \rangle \oplus \alpha'$$

$$\delta(\text{Move}, \langle +f_i\alpha, \beta_1, \ldots, \beta_{i-1}, -f_i\beta_i, \beta_{i+1}, \ldots, \beta_k \rangle) = \langle \alpha, \beta_1, \ldots, \beta_{i-1}, \varepsilon, \beta_{i+1}, \ldots, \beta_k \rangle \oplus \beta_i$$

B.4 Monadic Second-Order Definition of MGs

B.4.1 Defining $L^2_{K,P}$ as a Variant of MSO

Rather than MSO itself, I use the expressively equivalent variant $L^2_{K,P}$ as developed in Rogers (1998). The syntax of $L^2_{K,P}$ uses the standard logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, quantifiers $\forall$ and $\exists$ over individuals and sets, and the standard grouping symbols (,), [, and]. Furthermore, $L^2_{K,P}$ employs

- the equality predicate $\approx$,
- the immediate dominance predicate $\triangleleft$,
- a fixed countably infinite set X_0 of node variables $x_1, \ldots, x_n$,
- a fixed countably infinite set X_1 of set variables $X_1, \ldots, X_n$,
- a set K of individual constant symbols,
- a set P of unary predicate symbols.

In our case K is empty and P contains a predicate λ_i for every LI l_i in the lexicon of the MG that is to be defined, as well as the predicates Merge and Move. Rogers also introduces binary predicates for the relations left-of and reflexive dominance, but the former serves no purpose here and the latter will be defined directly from immediate dominance. The set L of well-formed $L^2_{K,P}$ formulas is constructed as usual:

- for all $x, y \in X_0$ and $p \in P \cup X_1$, L contains $p(x)$, $x \triangleleft y$, and $x \approx y$,
- for all $\phi, \psi \in L$, $x \in X_0$ and $X \in X_1$, L contains $(\neg\phi)$, $(\phi \vee \psi)$, $\exists x[\phi]$, and $\exists X[\phi]$.

The remaining logical connectives and quantifiers are obtained from $\neg$, $\vee$, and $\exists$ in the standard fashion, and brackets may be dropped where convenient. Per convention $\neg(x \approx y)$ is conflated into $x \not\approx y$. I also extend $\approx$ to sets such that $X \approx Y$ iff $\forall z[X(z) \leftrightarrow Y(z)]$. Furthermore, $\exists! x[\phi]$ acts as a shorthand for $\exists x[\phi] \wedge \forall y[\phi[x/y] \rightarrow x \approx y]$, where either $x, y \in X_0$ or $x, y \in X_1$ and $\phi[x/y]$ is the result of substituting y for all free occurrences of x in ϕ.

I implicitly adopt the $L^2_{K,P}$-axiomatization of finite trees presented in Rogers (1998:15–17). A model for the language $L^2_{K,P}$ as defined here is a tuple $M := \left\langle D, R_\triangleleft, R_p \right\rangle_{p \in P}$, where

- D is a finite Gorn tree domain (see p. 292),
- $R_{\triangleleft} \subseteq D \times D$ interprets $\triangleleft$ such that $\langle x, y \rangle \in R_{\triangleleft}$ iff $x = u$ and $y = ui$ for some $u \in \mathbb{N}^*$ and $i \in \mathbb{N}$.
- for each $p \in P$, R_p is a (possibly empty) subset of D interpreting p.

An $L^2_{K,P}$ assignment s is a function that maps every $x \in X_0$ to some element of D and every $X \in X_1$ to some subset of D. Satisfaction of an $L^2_{K,P}$ formula with respect to a model M and assignment s is determined as follows:

$$
\begin{array}{lll}
M, g \models & X(x) & \text{iff } s(x) \in s(X) \\
M, g \models & p(x) & \text{iff } s(x) \in R_p \\
M, g \models & x_i \approx x_j & \text{iff } s(x_i) = s(x_j) \\
M, g \models & x_i \triangleleft x_j & \text{iff } \langle s(x_i), s(x_j) \rangle \in R_{\triangleleft} \\
M, g \models & \neg\phi & \text{iff } M, g \not\models \phi \\
M, g \models & \phi \vee \psi & \text{iff } M, g \models \phi \text{ or } M, g \models \psi \\
M, g \models & \exists x[\phi] & \text{iff there is some } u \in D \text{ such that } M, g[x/u] \models \phi \\
M, g \models & \exists X[\phi] & \text{iff there is some } u \subseteq D \text{ such that } M, g[X/u] \models \phi
\end{array}
$$

B.4.2 Ancillary Predicates

In addition to the unary and binary predicates introduced so far I use a number of predicates as convenient shorthands for defining the MDTL of an MG with $(\Sigma, Feat)$-lexicon *Lex*.

Let $Lex^{(n)}$ be the set of all LIs in *Lex* that have n positive polarity features. Then $|\gamma|$ is the greatest n such that $Lex^{(n)} \neq \emptyset$. That is to say, $|\gamma|$ is the maximum number of positive polarity features on an LI. Similarly, $|\delta|$ is the maximum number of licensee features on an LI (so no LI has more than $|\gamma| + |\delta| + 1$ features). Building on these thresholds, I use formula schemes to instantiate a number of predicates in order to

describe the feature make-up of LIs.

$$lex(l,n) \iff \bigvee_{\lambda \in Lex^{(n)}} \lambda(l)$$

$$lex(l) \iff \bigvee_{0 \le i \le |\gamma|} lex(l,n)$$

$$\bigwedge_{\substack{1 \le n \le |\gamma|+|\delta|+1 \\ f \in Feat}} \Big(f(l,n) \iff \bigvee_{\lambda \text{ has } f \text{ as its } n\text{-th feature}} \lambda(l) \Big)$$

$$\bigwedge_{\substack{f \in Feat \\ 1 \le n \le |\delta|}} \Big(-f(l,n) \iff \bigvee_{\lambda \text{ has } f \text{ as its } n\text{-th licensee feature}} \lambda(l) \Big)$$

$$\bigwedge_{f \in Feat} \Big(f(l,cat) \iff \bigvee_{\lambda \text{ has category feature } f} \lambda(l) \Big)$$

Proper dominance $\triangleleft^+$ and reflexive dominance $\triangleleft^*$ are defined implicitly. Even though implicit definitions can increase the power of MSO, this is not the case here.

$$closed(R,X) \iff \forall x, y\, [X(x) \wedge x\, R\, y \rightarrow X(y)]$$

$$x \triangleleft^+ y \iff \forall X\, [closed(\triangleleft, X) \wedge X(x) \rightarrow X(y)]$$

$$x \triangleleft^* y \iff x \triangleleft^+ y \vee x \approx y$$

The predicate $x \triangleleft_i y$ holds iff x properly dominates y and the two are separated by i edges of the immediate dominance relation.

$$\bigwedge_{1 \le n < |\gamma|} \Big(x \triangleleft_{n+1} y \iff \exists z_1, z_2, \ldots, z_{n-1}, z_n \Big[x \triangleleft z_1 \wedge z_n \triangleleft y \wedge \bigwedge_{1 \le i < n} z_i \triangleleft z_{i+1} \Big] \Big)$$

$$x \triangleleft_1 y \iff x \triangleleft y$$

$$x \triangleleft_0 y \iff x \approx y$$

Dedicated predicates are also used to pick out the root and the leafs of a set of

nodes.

$$\begin{aligned} root(x,X) &\iff X(x) \wedge \forall y[X(y) \rightarrow x \triangleleft^* y] \\ root(x) &\iff \exists X \forall y[X(y) \wedge root(x,X)] \\ leaf(x,X) &\iff X(x) \wedge \forall y[X(y) \rightarrow \neg(x \triangleleft^+ y)] \\ leaf(x) &\iff \exists X \forall y[X(y) \wedge leaf(x,X)] \end{aligned}$$

B.4.3 Free Slice Language

Mirroring the standard definition of MDTLs in terms of restricted free slice languages, I first define slices as particular sets of nodes and then require that every tree can be partitioned into slices. This gives us a model-theoretic description of free slice languages, which is then further restricted by the constraints of the feature calculus. As a slight improvement on the definition in Sec. B.2 I do not stipulate that slices are right-branching.

The set X is the slice of l iff l is an LI with n positive polarity features and X contains only l and the n lowest nodes that properly dominate l.

$$\begin{aligned} slice(X,l) &\iff \bigvee_{n \leq |\gamma|} \left(lex(l,n) \wedge \forall x \left[X(x) \leftrightarrow \bigvee_{0 \leq i \leq n} x \triangleleft_i l \right] \right) \\ slice(X) &\iff \exists l[slice(X,l)] \end{aligned}$$

It is easy to see that a tree is partitioned into slices iff every node belongs to exactly one slice.

$$\forall x \exists! X[slice(X) \wedge X(x)] \qquad \textbf{(Partition)}$$

Now it must be ensured that the shape of the slices matches the feature specifications of their LIs. This requires us to ensure that:

- all nodes are labeled by some $p \in P$,

- LIs occur on leafs, and only there,
- Move occurs on unary branching nodes, and only there,
- Merge occurs on binary branching nodes, and only there,
- the label of an interior node matches the operation type of the feature it is associated to.

The first four clauses are quickly taken care of.

$$\forall x \Big[\big(lex(x) \vee \text{Move}(x) \vee \text{Merge}(x)\big) \wedge \big(lex(x) \leftrightarrow leaf(x)\big) \wedge \big(\text{Move}(x) \leftrightarrow \exists! y\,[x \triangleleft y]\big) \wedge \big(\text{Merge}(x) \leftrightarrow \exists! y \exists! z\,[x \triangleleft y \wedge x \triangleleft z \wedge y \not\approx z]\big)\Big] \quad \textbf{(Arity)}$$

The last clause requires determining for each node in a slice which positive polarity feature it is associated to. Inspection of the function ζ in Sec. B.2 reveals that a node that is separated by n edges from the LI l of its slice is associated to the n-th positive polarity feature of l. This is captured by the predicate $\widetilde{\triangleleft}_i$, which is defined in terms of $\triangleleft_i$ and the *slice mate* relation $\sim$.

$$x \sim y \iff \exists X\,[slice(X) \wedge X(x) \wedge X(y)]$$

$$\bigwedge_{0 \leq i \leq \gamma} \Big(x \mathrel{\widetilde{\triangleleft}_i} y \iff x \sim y \wedge x \triangleleft_i y \Big)$$

$$\bigwedge_{f \in Feat} \Big(f(x) \iff \exists l \Big[\bigvee_{1 \leq n \leq |\gamma|} x \mathrel{\widetilde{\triangleleft}_n} l \wedge f(l,n) \Big] \Big)$$

Notice that $f(x)$ only holds of interior node and thus never for LIs. Hence the

distribution of Merge and Move can be tied directly to these feature predicates.

$$\forall x \Big[\Big(\text{Merge}(x) \leftrightarrow \bigvee_{\omega(f)=merge} f(x) \Big) \wedge \Big(\text{Move}(x) \leftrightarrow \bigvee_{\omega(f)=move} f(x) \Big) \Big] \quad \textbf{(Association)}$$

Now it only remains to regulate how slices may be combined.

B.4.4 Minimalist Derivation Tree Languages

An MDTL is a free slice language that obeys the constraints the feature calculus imposes on Merge and Move. The standard definition expresses this via the well-formedness conditions **Final**, **Merge**, **Move**, **SMC**, which can easily be recast in terms of MSO (or $L^2_{K,P}$, more specifically).

Once more I introduce some auxiliary predicates first. The slice root and the definition of *match* play an important role in restricting both Merge and Move.

$$sliceroot(x,l) \iff \exists X [slice(X,l) \wedge root(x,X)]$$

$$\bigwedge_{f \in Feat} \Big(match(x,f) \iff \bigvee_{\substack{g \in Feat \\ \nu(f)=\nu(g) \\ \omega(f)=\omega(g) \\ \pi(f) \neq \pi(g)}} g(x) \Big)$$

The occurrence-predicates provide the means for picking out the Move nodes that check an LI's licensee features. Recall that the i-th occurrence of LI l is simply the lowest node that matches the i-th licensee feature of l and properly dominates the $(i-1)$-th occurrence of l. The zero occurrence is identified with the slice root so

that the Move nodes associated to l cannot check any licensee features of l.

$$
\begin{aligned}
occ_0(x,l) &\iff sliceroot(x,l)\\
\bigwedge_{1\leq i\leq|\delta|}\Big(occ_i(x,l) &\iff \bigvee_{f\in Feat}\Big(-f(l,i)\wedge match(x,f)\wedge\\
&\qquad\qquad \exists y\big[occ_{i-1}(y,l)\wedge x\lhd^{+} y\wedge\\
&\qquad\qquad\quad \neg\exists z[x\lhd^{+} z\wedge z\lhd^{+} y\wedge match(z,f)]\big]\Big)\Big)\\
occ(x,l) &\iff \bigvee_{1\leq i\leq|\delta|} occ_i(x,l)
\end{aligned}
$$

The constraints from Sec. B.2 can now be translated almost literally into $L^2_{K,P}$ formulas.

$$\forall x,l[root(x)\wedge x\sim l\wedge lex(l)\rightarrow c(l,cat)] \qquad \textbf{(Final)}$$

$$\forall x\Big[\text{Merge}(x)\rightarrow\exists y,l\Big[x\lhd y\wedge sliceroot(y,l)\wedge \bigvee_{f\in Feat}\big(f(l,cat)\wedge match(x,f)\big)\Big]\Big] \qquad \textbf{(Merge)}$$

$$\forall x\Big[\bigwedge_{1\leq i\leq|\delta|}\Big(-f(x,i)\rightarrow\exists y\big[occ_i(x,l)\big]\Big)\Big] \qquad \textbf{(Move)}$$

$$\forall x\Big[\text{Move}(x)\rightarrow\exists!l\big[occ(x,l)\big]\Big] \qquad \textbf{(SMC)}$$

A tree is a well-formed Minimalist derivation tree iff it is a model of the conjunction of **Partition**, **Arity**, **Association**, **Final**, **Merge**, **Move**, and **SMC**.

B.4.5 Mapping to Multi-Dominance Trees

The mapping from Minimalist derivation to derived structures is specified via an MSO transduction. I first define the mapping to multi-dominance trees, which is then modified to create a transduction to the trees created by canonical MGs: phrase

structure trees with non-indexed traces.

In both cases the mapping to the derived structures is a total, deterministic MSO graph transduction as defined in Bloem and Engelfriet (2000): A *graph* $G := \langle D, \ell_\Sigma, R \rangle$ is a set D of nodes such that each $d \in D$ is assigned exactly one element of Σ by the function $\ell_\Sigma : D \to \Sigma$, and R is a finite set of relations $r_i \subseteq D \times D$. The class of all graphs over alphabet Σ and class R of relations is denoted $G[\Sigma, R]$, and the MSO-language with unary predicates for all $\sigma \in \Sigma$ and binary predicates for all $r \in R$ is denoted $\mathrm{MSO}[\Sigma, R]$. An *MSO graph transduction* from $G[\Sigma, R]$ to $G[\Omega, S]$ is a triple $\langle C, \Psi, X \rangle$, where C is a finite set of *copy names*, Ψ the family of *node formulas*, and X the family of *edge formulas*.

The set C is only used for transductions that increase the size of the input tree. In this case, multiple copies of the input are created which are then relabeled, partially deleted and recombined to yield the output tree. This assembly procedure is mediated by the node and edge formulas. The family Ψ contains a node formula $\psi_{\omega,c}(x) \in \mathrm{MSO}[\Sigma, R]$ for every label $\omega \in \Omega$ and every $c \in C$. The idea is that node x in copy c is labeled ω iff $\psi_{\omega,c}(x)$ holds in the input graph. The node formulas specify the distribution of labels in the output structure. The family χ contains an edge formula $\chi_{s,c,c'}(x, y) \in \mathrm{MSO}[\Sigma, R]$ for every relation $s \in S$ and all $c, c' \in C$. Here node x in copy c is related to y in copy c' via relation s iff $\chi_{s,c,c'}(x, y)$ is true in the input graph.

The MSO graph transduction Φ_{gr} from $G[\Sigma, R]$ to $G[\Omega, S]$ maps $i := \langle I, \ell_\Sigma, R \rangle$ to the graph $o := \langle O, \ell_\Omega, S \rangle$ with

- $O := \{ \langle c, u \rangle \mid c \in C, u \in I$, and there is exactly one $\omega \in \Omega$ such that $I \models \psi_{\omega,c}(x)[x/u] \}$,
- $\forall s \in S, s := \{ \langle \langle c, u \rangle, \langle c', u' \rangle \rangle \mid \langle c, u \rangle, \langle c, u' \rangle \in O$ and $I \models \chi_{s,c,c'}(x, y)[x/u, y/u'] \}$,
- $\ell_\Omega := \{ \langle \langle c, u \rangle, \omega \rangle \mid \langle c, u \rangle \in O, \omega \in \Omega$, and $I \models \psi_{\omega,c}(x)[x/u] \}$

In the case of multi-dominance trees, the output tree is exactly the size of the input tree, so C is redundant. Hence we only have to provide node formulas ψ_ω for every label that may occur in an output tree and edge formulas χ_s for every relation in the output tree. The set of output labels consists of $<, >$ for interior nodes and the phonetic exponent σ of every LI l of the MG with (Σ, Feat)-lexicon whose MDTL is being transduced. The output relations are just immediate dominance $\blacktriangleleft$ and left-sibling — proper dominance and linear precedence can be obtained from these more primitive relations.

$$\begin{aligned}
\psi_{>}(x) &\iff \mathrm{Move}(x) \vee \big(\mathrm{Merge}(x) \wedge \neg\exists y[x \mathrel{\tilde{\triangleleft}} y \wedge \mathit{lex}(y)]\big)\\
\psi_{<}(x) &\iff \mathrm{Merge}(x) \wedge \exists y[x \mathrel{\tilde{\triangleleft}} y \wedge \mathit{lex}(y)]\\
\bigwedge_{\sigma\in\Sigma}\Big(\psi_\sigma(x) &\iff \bigvee_{l:=\sigma::\mathrm{f}_1\cdots\mathrm{f}_n\in \mathit{Lex}} l(x)\Big)\\
\chi_{\blacktriangleleft}(x,y) &\iff x \triangleleft y \vee \exists l\Big[\mathit{occ}(x,l) \wedge \mathit{sliceroot}(y,l)\Big]\\
\chi_{\prec}(x,y) &\iff \exists l\exists z[\mathit{occ}(z,l) \wedge \mathit{sliceroot}(x,l) \wedge z \triangleleft y] \vee\\
&\qquad\qquad \exists z[\mathrm{Merge}(z) \wedge z \triangleleft x \wedge z \triangleleft y \wedge (x \sim z \rightarrow \mathit{lex}(x))]
\end{aligned}$$

If desired, proper dominance and reflexive dominance in the output structure can be defined from $\blacktriangleleft$ similar to how $\triangleleft^+$ was obtained from $\triangleleft$:

$$\begin{aligned}
x \blacktriangleleft^+ y &\iff \forall X[\mathit{closed}(\blacktriangleleft, X) \wedge X(x) \rightarrow X(y)]\\
x \blacktriangleleft^* y &\iff x \blacktriangleleft^+ y \vee x \approx y
\end{aligned}$$

The linear precedence relation $\prec^*$ is inherited from the interaction of $\blacktriangleleft^+$ and $\prec$.

$$x \prec^* y \iff \exists z, z'[z \blacktriangleleft^* x \wedge z' \blacktriangleleft^* y \wedge z \prec z']$$

Note that this order may be reflexive in a multi-dominance tree and is not guaranteed to be antisymmetric. In order to define a total order this way, additional assumptions

need to be made about the linearization of multi-dominance structures. I will not pursue this issue any further here since any reasonable linearization procedure can be defined in terms of MSO (probably even first-order logic).

It is interesting to note that if $\triangleleft^+$ is part of our logical signature, both MDTLs and their mapping Φ_{gr} to multi-dominance graphs are first-order definable. Graf (2012b) shows this for MDTLs, but not for the mapping to the derived structures. However, a closer examination of our definitions reveals that set quantification and set predicates are only involved in the predicate $slice(X, l)$ and everything that builds on it, in particular the slice mate relation $\sim$ and slice-relativized dominance $\widetilde{\triangleleft}_i$, and $sliceroot(x, l)$. Since the size of slices cannot exceed $|\gamma| + 1$, there are only finitely many slice configurations to consider, so all these predicates can be defined in first-order logic by listing these configurations. Therefore all aspects of MGs are first-order definable if the basic structural relation is $\triangleleft^+$ rather than $\triangleleft$.

B.4.6 Mapping to Phrase Structure Trees

Mapping derivations to phrase structure trees with non-indexed traces means that the output tree might be bigger than the input tree. However, since every movement step causes the creation of exactly one trace the number of traces is identical to the number of movement nodes. Hence it suffices to have two copies of the input tree. The first copy is our primary tree that represents the complete output tree *modulo* traces. The second one will be mostly ignored, except for its Move nodes which are relabeled as traces and connected into the output tree in a suitable fashion.

The MSO tree transduction Φ_{tr} is the triple $\langle \{1, 2\}, \Psi, X \rangle$ that maps every derivation tree with labels drawn from $\{\text{Merge}, \text{Move}\} \cup Lex$ and nodes ordered by $\triangleleft$ to a tree where nodes are ordered by $\blacktriangleleft$ and $\prec$ and labeled with symbols drawn from $\{<, >, t\} \cup \{\sigma \mid \sigma :: \text{f}_1 \cdots \text{f}_n \in Lex\}$.

The node formulas for the first copy are just the node formulas from the multi-

dominance transduction.

$$\psi_{>,1}(x) \iff \text{Move}(x) \vee \big(\text{Merge}(x) \wedge \neg\exists y[x \mathrel{\tilde{\triangleleft}} y \wedge \mathit{lex}(y)]\big)$$
$$\psi_{<,1}(x) \iff \text{Merge}(x) \wedge \exists y[x \mathrel{\tilde{\triangleleft}} y \wedge \mathit{lex}(y)]$$
$$\bigwedge_{\sigma\in\Sigma}\Big(\psi_{\sigma,1}(x) \iff \bigvee_{l:=\sigma::\mathrm{f}_1\cdots\mathrm{f}_n\in \mathit{Lex}} l(x)\Big)$$

Since the output alphabet now also contains a label t for traces, we need to add the corresponding formulas. For the first copy, this formula is always false, denoted $\bot$. For the second copy it holds for all nodes that are exactly in the location where traces should show up in the output tree: at the slice root of a moving LI or immediately below all Move nodes that are not *final occurrences*, where $f\text{-}occ(x,l) \iff occ(x,l) \wedge \forall y[occ(y,l) \rightarrow x \triangleleft^* y]$.

$$\psi_{t,1}(x) \iff \bot$$
$$\psi_{t,2}(x) \iff \exists l,z\Big[occ(z,l) \wedge \Big(\mathit{sliceroot}(x,l) \vee \big(\neg f\text{-}occ(z,l) \wedge z \triangleleft x\big)\Big)\Big]$$

The node formulas for all other labels are also false in the second copy: $\psi_{>,2}(x) = \psi_{<,2}(x) = \psi_{\sigma,2}(x) = \bot$. This leaves us with only a few nodes in the second copy that actually belong to the domain of the output tree. All unlabeled nodes in the second copy can be ignored in the edge formulas.

There are now eight edge formulas, four for $\blacktriangleleft$ and four for $\prec$. For nodes in the first copy, $x \blacktriangleleft y$ holds if $x \triangleleft y$ and either y is not the slice root of a mover or y is the slice root of a mover and x is the final occurrence of the moving LI. We also need an edge formula $\chi_{\blacktriangleleft,1,2}$ to ensure that the traces in the second copy are related to the correct nodes in the first copy. Fortunately this is straight-forward because we used a rather elaborate formula for $\psi_{t,2}$ to pick out only those nodes that are

exactly in the right position for the traces.

$$\chi_{\blacktriangleleft,1,1}(x,y) \iff \Big(x \triangleleft y \wedge \neg\exists l \exists m \Big[occ(m,l) \wedge sliceroot(y,l)\Big]\Big) \vee \exists l \Big[sliceroot(y,l) \wedge f\text{-}occ(x,l)\Big]$$

$$\chi_{\blacktriangleleft,1,2}(x,y) \iff x \triangleleft y$$

Since traces do not immediately dominate any other nodes we do not need to worry about the cases where the first argument of $\blacktriangleleft$ is taken from the second copy of the input, wherefore $\chi_{\blacktriangleleft,2,1}(x,y) = \chi_{\blacktriangleleft,2,2}(x,y) = \bot$.

The left-sibling relation $\prec$ also has to be defined more carefully now. As traces cannot be siblings, it is safe to let $\chi_{\prec,2,2} = \bot$. The formula $\chi_{\prec,1,2}$ captures the cases where a trace is the right daughter of a node, which is limited to traces left behind by the very first movement step of a complement. The formula $\chi_{\prec,2,1}$ subsumes all cases where movement takes place from a specifier. The formula $\chi_{\prec,1,1}$ establishes the order of nodes that aren't traces.

$$\chi_{\prec,1,1}(x,y) \iff \exists l \exists z [f\text{-}occ(z,l) \wedge sliceroot(x,l) \wedge z \triangleleft y] \vee \exists z \big[\text{Merge}(z) \wedge z \triangleleft x \wedge z \triangleleft y \wedge (x \sim z \rightarrow lex(x)) \wedge \neg\exists l \exists m [occ(m,l) \wedge sliceroot(y,l)]\big]$$

$$\chi_{\prec,1,2}(x,y) \iff \exists z [z \triangleleft x \wedge z \triangleleft y \wedge x \not\approx y \wedge lex(x)]$$

$$\chi_{\prec,2,1}(x,y) \iff \exists z \Big[z \triangleleft x \wedge z \triangleleft y \wedge \Big(\text{Move}(z) \vee \big(\text{Merge}(z) \wedge \neg lex(y)\big)\Big)\Big]$$

The mapping to phrase structure trees is more complicated than the one to multi-dominance trees, but just like the latter it can be expressed without any kind of set quantification and is thus first-order definable. Both types of mappings are also direction preserving, that is to say, $x \blacktriangleleft y$ implies $x \triangleleft^+ y$.

BIBLIOGRAPHY

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.

Adger, David. 2010. A minimalist theory of feature structure. In *Features: Perspectives on a key notion in linguistics*, ed. Anna Kibort and Greville G. Corbett, 185–218. Oxford: Oxford University Press.

Aoun, Joseph, Lina Choueiri, and Norbert Hornstein. 2001. Resumption, movement and derivational economy. *Linguistic Inquiry* 32:371–403.

Backofen, Rolf, James Rogers, and K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. Technical Report RR-95-05, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Erwin-Schrödinger Straße, Postfach 2080, 67608 Kaiserslautern, Germany.

Béjar, Susana, and Milan Rezac. 2009. Cyclic agree. *Linguistic Inquiry* 40:35–73.

den Besten, Hans, and Gert Webelhuth. 1990. Stranding. In *Scrambling and barriers*, ed. Günther Grewendworf and Wolfgang Sternefeld, 77–92. New York: Academic Press.

Blackburn, Patrick, Claire Gardent, and 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.

Blackburn, Patrick, and Wilfried Meyer-Viol. 1994. Linguistics, logics, and finite trees. *Logic Journal of the IGPL* 2:3–29.

Blackburn, Patrick, Maarten de Rijke, and Yde Venema. 2002. *Modal logic*. Cambridge: Cambridge University Press.

Bloem, Roderick, and 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.

Borer, Hagit. 1984. *Parametric syntax: Case studies in semitic and romance languages*. Dordrecht: Foris.

Brody, Michael. 1995. *Lexico-logical form: A radically Minimalist theory*. Cambridge, Mass.: MIT Press.

Brody, Michael. 2000. Mirror theory: Syntactic representation in perfect syntax. *Linguistic Inquiry* 31:29–56.

Brody, Michael. 2002. On the status of representations and derivations. In *Derivation and explanation in the minimalist program*, ed. Samuel D. Epstein and Daniel T. Seely, 19–41. Oxford: Blackwell.

Büchi, J. Richard. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6:66–92.

Castillo, Juan Carlos, John E. Drury, and Kleanthes K. Grohmann. 2009. Merge over Move and the extended projection principle: MOM and the EPP revisited. *Iberia* 1:53–114.

Chomsky, Noam. 1957. *Syntactic structures*. The Hague: Mouton.

Chomsky, Noam. 1964. *Current issues in linguistic theory*. The Hague: Mouton.

Chomsky, Noam. 1965. *Aspects of the theory of syntax*. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 1973. Conditions on transformations. In *A festschrift for Morris Halle*, ed. Stephen Anderson and Paul Kiparsky, 232–286. New York: Holt, Rinehart, and Winston. Reprinted in Chomsky (1977).

Chomsky, Noam. 1977. On *wh*-movement. In *Formal syntax*, ed. Peter Culicover, T. Wasow, and A. Akmajian, 71–132. New York: Academic Press.

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

Chomsky, Noam. 1986. *Barriers*. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 1990. On formalization and formal linguistics. *Natural Language and Linguistic Theory* 8:143–1477.

Chomsky, Noam. 1991. Some notes on economy of derivation and representation. In *Principles and parameters in comparative grammar*, ed. Robert Freidin, 417–454. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 1993. A minimalist program for linguistic theory. In *The view from building 20*, ed. Kenneth Hale and Samuel Jay Keyser, 1–52. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 1995a. Bare phrase structure. In *Government and binding theory and the minimalist program*, ed. Gert Webelhuth, 383–440. Oxford: Blackwell.

Chomsky, Noam. 1995b. Categories and transformations. In *The minimalist program*, chapter 4, 219–394. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 1995c. *The minimalist program*. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 2000. Minimalist inquiries: The framework. In *Step by step: Essays on minimalist syntax in honor of Howard Lasnik*, ed. Roger Martin, David Michaels, and Juan Uriagereka, 89–156. Cambridge, Mass.: MIT Press.

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

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

Chomsky, Noam. 2004b. *The generative enterprise revisited. Discussions with Riny Huybregts, Henk van Riemsdijk, Naoki Fukui and Mihoko Zushi*. Berlin: Mouton de Gruyter.

Chomsky, Noam. 2005. Three factors in language design. *Linguistic Inquiry* 36:1–22.

Cinque, Guglielmo. 2005. Deriving Greenberg's universal 20 and its exceptions. *Linguistic Inquiry* 36:315–332.

Collins, Chris. 1994. Economy of derivation and the generalized proper binding condition. *Linguistic Inquiry* 25:45–61.

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

Collins, Chris. 2002. Eliminating labels. In *Derivation and explanation in the minimalist program*, ed. Samuel D. Epstein and Daniel T. Seely, 42–64. Oxford: Blackwell.

Comon, H., M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. TommasiK. 2008. Tree automata: Techniques and applications. Available online: http://www.grappa.univ-lille3.fr/tata. Release from November 18, 2008.

Cornell, Thomas, and James Rogers. 1998. Model theoretic syntax. In *The Glot International State of the Article Book*, volume 1 of *Studies in Generative Grammar 48*, 101–125. Mouton de Gruyter.

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

Davey, Brian A., and Hilary Priestley. 2002. *Lattices and order*. Oxford: Cambridge University Press, second edition edition.

Dickinson, Michael. 2001. Solving the mystery of insect flight. *Scientific American* 284:48–57.

Doner, John. 1970. Tree acceptors and some of their applications. *Journal of Computer and System Sciences* 4:406–451.

Ebbinghaus, Heinz-Dieter, Jörg Flum, and Wolfgang Thomas. 1996. *Mathematical logic*. Springer.

Enderton, Herbert. 2001. *A mathematical introduction to logic*. San Diego, CA: Academic Press, 2^{nd} edition.

Engelfriet, Joost. 1975. Bottom-up and top-down tree transformations — a comparison. *Mathematical Systems Theory* 9:198–231.

Engelfriet, Joost. 1997. Context-free graph grammars. In *Handbook of formal languages, Vol III: Beyond words*, ed. Gregorz Rozenberg and Arto Salomaa, 125–213. Berlin: Springer.

Engelfriet, Joost, and L.M. Heyker. 1991. The string generating power of context-free hypergraph grammars. *Journal of Computational System Science* 43:328–360.

Epstein, Samuel D., Erich M. Groat, Ruriko Kawashima, and Hisatsugu Kitahara. 1998. *A derivational approach to syntactic relations*. Oxford: Oxford University Press.

Epstein, Samuel D., and Daniel T. Seely. 2002. Rule applications as cycles in a level-free syntax. In *Derivation and explanation in the minimalist program*, ed. Samuel D. Epstein and Daniel T. Seely, 65–89. Oxford: Blackwell.

Epstein, Samuel D., and Daniel T. Seely. 2006. *Derivations in minimalism*. Cambridge, Mass.: Cambridge University Press.

Fillmore, Charles J. 1968. The case for case. In *Universals in linguistic theory*, ed. Emmon Bach and R.T. Harms, 1–88. New York: Holt, Rinehart and Winston.

Fox, Chris, and Shalom Lappin. 2005. *Foundations of intensional semantics*. Oxford: Wiley-Blackwell.

Fox, Danny. 1995. Economy and scope. *Natural Language Semantics* 3:283–341.

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

Frank, Robert, and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics* 24:307–315.

Frey, Werner, and Hans-Martin Gärtner. 2002. On the treatment of scrambling and adjunction in minimalist grammars. In *Proceedings of the Conference on Formal Grammar (FGTrento)*, 41–52. Trento.

Fujiyoshi, Akio, and Takumi Kasai. 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems* 33:59–83.

Gamut, L. T. F. 1990a. *Logic, language, and meaning, volume 1: Introduction to logic*. Chicago: University of Chicago Press.

Gamut, L. T. F. 1990b. *Logic, language, and meaning, volume 2: Intensional logic and logical grammar*. Chicago: University of Chicago Press.

Gärtner, Hans-Martin. 2002. *Generalized transformations and beyond: Reflections on minimalist syntax*. Berlin: Akademie-Verlag.

Gärtner, Hans-Martin, and Jens Michaelis. 2005. A note on the complexity of constraint interaction. In *Logical aspects of computational linguistics (LACL'05)*, ed.

P. Blache, E. Stabler, and J. Busquets, number 3492 in Lecture Notes in Artificial Intelligence, 114–130. Berlin: Springer.

Gärtner, Hans-Martin, and Jens Michaelis. 2007. Some remarks on locality conditions and Minimalist Grammars. In *Interfaces + recursion = language? Chomsky's minimalism and the view from syntax-semantics*, ed. Uli Sauerland and Hans-Martin Gärtner, 161–196. Berlin: Mouton de Gruyter.

Gärtner, Hans-Martin, and Jens Michaelis. 2010. On the treatment of multiple-wh-interrogatives in minimalist grammars. In *Language and logos*, ed. Thomas Hanneforth and Gisbert Fanselow, 339–366. Berlin: Akademie Verlag.

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

Gorn, Saul. 1967. Explicit definitions and linguistic dominoes. In *Systems and Computer Science, Proceedings of the Conference held at University of Western Ontario, 1965*. Toronto: University of Toronto Press.

Graehl, Jonathan, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics* 34:391–427.

Graf, Thomas. 2010a. *Logics of phonological reasoning*. Master's thesis, University of California, Los Angeles.

Graf, Thomas. 2010b. A tree transducer model of reference-set computation. *UCLA Working Papers in Linguistics* 15:1–53.

Graf, Thomas. 2011. Closure properties of minimalist derivation tree languages. In *LACL 2011*, ed. Sylvain Pogodalla and Jean-Philippe Prost, volume 6736 of *Lecture Notes in Artificial Intelligence*, 96–111. Heidelberg: Springer.

Graf, Thomas. 2012a. An algebraic perspective on the person case constraint. In *Theories of Everything. In Honor of Ed Keenan*, ed. Thomas Graf, Denis Paperno,

Anna Szabolcsi, and Jos Tellings, volume 17 of *UCLA Working Papers in Linguistics*, 85–90.

Graf, Thomas. 2012b. Locality and the complexity of minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. Philippe de Groot and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227. Heidelberg: Springer.

Graf, Thomas. 2012c. Movement-generalized minimalist grammars. In *LACL 2012*, ed. Denis Béchet and Alexander J. Dikovsky, volume 7351 of *Lecture Notes in Computer Science*, 58–73.

Graf, Thomas. 2012d. Reference-set constraints as linear tree transductions via controlled optimality systems. In *Formal Grammar 2010/2011*, ed. Philippe de Groote and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 97–113. Heidelberg: Springer.

Graf, Thomas. 2012e. Tree adjunction as minimalist lowering. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, 19–27.

Graf, Thomas, and 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. Paris, France. URL http://www.aclweb.org/anthology-new/W/W12/W12-4622.

Grodzinsky, Yosef, and Tanja Reinhart. 1993. The innateness of binding and coreference. *Linguistic Inquiry* 24:69–102.

Grune, Dick, and Ceriel J.H. Jacobs. 2008. *Parsing techniques. A practical guide*. New York: Springer, second edition.

Gécseg, Ferenc, and Magnus Steinby. 1984. *Tree automata*. Budapest: Academei Kaido.

Gécseg, Ferenc, and Magnus Steinby. 1997. Tree languages. In *Handbook of formal languages*, ed. Gregorz Rozenberg and Arto Salomaa, volume 3, 1–68. New York: Springer.

Harkema, Henk. 2001a. A characterization of minimalist languages. In *Logical aspects of computational linguistics (LACL'01)*, ed. Philippe de Groote, Glyn Morrill, and Christian Retoré, volume 2099 of *Lecture Notes in Artificial Intelligence*, 193–211. Berlin: Springer.

Harkema, Henk. 2001b. *Parsing minimalist languages*. Doctoral Dissertation, University of California.

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

Heim, Irene. 2009. Forks in the road to Rule I. In *Proceedings of NELS 38*, 339–358.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry* 41:623–661.

Hopcroft, John E., and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Reading, Mass.: Addison Wesley.

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

Huang, C.-T. James. 1982. *Logical relations in Chinese and the theory of grammar*. Doctoral Dissertation, MIT.

Huybregts, Riny, and Henk van Riemsdijk, ed. 1982. *Noam Chomsky on the generative enterprise: A discussion with Riny Huybregts and Henk van Riemsdijk*. Dordrecht: Foris.

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

Joshi, Aravind. 1990. Processing crossed and nested dependencies: An automaton perspective on the psycholinguistic results. *Language and Cognitive Processes* 5:1–27.

Jäger, Gerhard. 2002. Some notes on the formal properties of bidirectional optimality theory. *Journal of Logic, Language, and Information* 11:427–451.

Kallmeyer, Laura. 2009. A declarative characterization of different types of multicomponent tree adjoining grammars. *Research on Language and Computation* 7:55–99.

Kanazawa, Makoto, Jens Michaelis, Sylvain Salvati, and Ryo Yoshinaka. 2011. Well-nestedness properly subsumes strict derivational minimalism. In *LACL 2011*, ed. Sylvain Pogodalla and Jean-Philippe Prost, volume 6736 of *Lecture Notes in Artifical Intelligence*, 112–128. Berlin: Springer.

Kanazawa, Makoto, and Sylvain Salvati. 2010. The copying power of well-nested multiple context-free grammars. In *LATA 2010*, volume 6031 of *LNCS*, 344–355.

Kanazawa, Makoto, and Sylvain Salvati. 2012. MIX is not a tree-adjoining language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 666–674.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kasper, Robert, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. 1995. Compilation of HPSG to TAG. In *Proceedings of the 33rd annual meeting of the Association for Computational Linguistics*, 92–99.

Kasprzik, Anna. 2007. *Two equivalent regularizations of Tree Adjoining Grammars*. Master's thesis, University of Tübingen.

Kayne, Richard S. 1994. *The antisymmetry of syntax*. Cambridge, Mass.: MIT Press.

Keenan, Edward, and Larry Moss. 2008. Mathematical structures in language. Ms., UCLA and Indiana University.

Kepser, Stephan. 2008. A landscape of logics for finite unordered unranked trees. In *FG-2008*, ed. Philippe de Groote, Laura Kallmeyer, Gerald Penn, and Giorgio Satta.

Kepser, Stephan, and Uwe Mönnich. 2006. Closure properties of linear context-free tree languages with an application to optimality theory. *Theoretical Computer Science* 354:82–97.

Kepser, Stephan, and Jim Rogers. 2011. The equivalence of tree adjoining grammars and monadic linear context-free tree grammars. *Journal of Logic, Language and Information* 20:361–384.

Keshet, Ezra. 2010. Situation economy. *Natural Language Semantics* 18:385–434.

Kisseberth, Charles. 1970. On the functional unity of phonological rules. *Linguistic Inquiry* 1:291–306.

Kobele, Gregory M. 2005. Features moving madly: A formal perspective on feature percolation in the minimalist program. *Research on Language and Computation* 3:391–410.

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

Kobele, Gregory M. 2008. Across-the-board extraction and minimalist grammars. In *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Frameworks*.

Kobele, Gregory M. 2010a. A formal foundation for A and A-bar movement. In *The mathematics of language*, ed. Christan Ebert, Gerhard Jäger, and Jens Michaelis, volume 6149 of *Lecture Notes in Computer Science*, 145–159. Heidelberg: Springer.

Kobele, Gregory M. 2010b. Without remnant movement, MGs are context-free. In *MOL 10/11*, ed. Christian Ebert, Gerhard Jäger, and Jens Michaelis, volume 6149 of *Lecture Notes in Computer Science*, 160–173.

Kobele, Gregory M. 2011. Minimalist tree languages are closed under intersection with recognizable tree languages. In *LACL 2011*, ed. Sylvain Pogodalla and Jean-Philippe Prost, volume 6736 of *Lecture Notes in Artificial Intelligence*, 129–144.

Kobele, Gregory M. 2012. Deriving reconstruction asymmetries. In *Local modeling of non-local dependencies*, ed. Artemis Alexiadou, Tibor Kiss, and Gereon Müller, 477–500. Berlin: Mouton de Gruyter.

Kobele, Gregory M., Sabrina Gerth, and John T. Hale. 2012. Memory resource allocation in top-down minimalist parsing. In *Proceedings of Formal Grammar 2012*.

Kobele, Gregory M., and Jens Michaelis. 2005. Two type-0 variants of minimalist grammars. In *FG-MoL 2005. The 10th conference on Formal Grammar and the 9th Meeting on Mathematics of Language*, 81–93. Edinburgh.

Kobele, Gregory M., and Jens Michaelis. 2011. Disentangling notions of specifier impenetrability. In *The Mathematics of Language*, ed. Makoto Kanazawa, András Kornia, Marcus Kracht, and Hiroyuki Seki, volume 6878 of *Lecture Notes in Artificial Intelligence*, 126–142.

Kobele, Gregory M., and Jens Michaelis. 2012. On the form-meaning relations definable by CoTAGs. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, 207–213. Paris, France. URL http://www.aclweb.org/anthology-new/W/W12/W12-4624.

Kobele, Gregory M., Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 71–80.

Kolb, Hans-Peter. 1999. Macros for minimalism: Towards weak descriptions of strong structures. In *Mathematics of syntactic structure*, ed. Hans-Peter Kolb and Uwe Mönnich, 232–259. Berlin: Walter de Gruyter.

Kozen, Dexter C. 1997. *Automata and computability*. Springer.

Kracht, Marcus. 1995a. Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy* 18:401–458.

Kracht, Marcus. 1995b. Syntactic codes and grammar refinement. *Journal of Logic, Language and Information* 4:41–60.

Kracht, Marcus. 1997. Inessential features. In *Logical aspects of computational linguistics*, ed. Alain Lecomte, F. Lamarche, and G. Perrier. Berlin: Springer.

Kuratowski, Kazimierz. 1921. Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta Mathematica* 2:161–171.

Magnan, Antoine. 1934. *Le vol des insectes*. Paris: Hermann et Cle.

Maletti, Andreas. 2008. Compositions of extended top-down tree transducers. *Information and Computation* 206:1187–1196.

Matushansky, Ora. 2006. Head movement in linguistic theory. *Linguistic Inquiry* 37:69–109.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. *Lecture Notes in Artificial Intelligence* 2014:179–198.

Michaelis, Jens. 2001. Transforming linear context-free rewriting systems into minimalist grammars. *Lecture Notes in Artificial Intelligence* 2099:228–244.

Michaelis, Jens. 2004. Observations on strict derivational minimalism. *Electronic Notes in Theoretical Computer Science* 53:192–209.

Michaelis, Jens. 2009. An additional observation on strict derivational minimalism. In *FG-MOL 2005*, ed. James Rogers, 101–111.

Michaelis, Jens, and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Artifical Intelligence*, 329–345. Springer.

Mönnich, Uwe. 1997. Adjunction as substitution. In *Formal Grammar '97*, 169–178.

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

Mönnich, Uwe. 2007. Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 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. URL http://www.aclweb.org/anthology-new/W/W12/W12-4605.

Morawietz, Frank. 2003. *Two-step approaches to natural language formalisms*. Berlin: Walter de Gruyter.

Müller, Gereon. 2005. Constraints in syntax. Lecture Notes, Universität Leipzig.

Müller, Gereon, and Wolfgang Sternefeld. 1996. A-bar chain formation and economy of derivation. *Linguistic Inquiry* 27:480–511.

Müller, Gereon, and Wolfgang Sternefeld. 2000. The rise of competition in syntax: A synopsis. In *Competition in syntax*, ed. Wolfgang Sternefeld and Gereon Müller, 1–68. Berlin: Mouton de Gruyter.

Nakamura, Masanori. 1997. Object extraction in Bantu applicatives: Some implications for minimalism. *Linguistc Inquiry* 28:252–280.

Nunes, Jairo. 2000. Erasing erasure. *D.E.L.T.A.* 16:415–429.

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

Partee, Barbara, Alice ter Meulen, and Robert Wall. 1990. *Mathematical methods in linguistics*. Dordrecht: Kluwer.

Peters, Stanley, and Robert W. Ritchie. 1971. On restricting the base component of transformational grammars. *Information and Control* 18:483–501.

Peters, Stanley, and Robert W. Ritchie. 1973a. Non-filtering and local-filtering transformational grammars. In *Approaches to natural language*, ed. Jaakko Hintikka, J.M.E. Moravcsik, and Patrick Suppes, 180–194. Dordrecht: Reidel.

Peters, Stanley, and Robert W. Ritchie. 1973b. On the generative power of transformational grammars. *Information Sciences* 6:49–83.

Plummer, Andrew, and Carl Pollard. 2012. Agnostic possible world semantics. In *LACL 2012*, ed. Denis Béchet and Alexander Dikovsky, number 7351 in Lecture Notes in Computer Science, 201–212. Heidelberg: Springer.

Potts, Christopher. 2001. Three kinds of transderivational constraints. In *Syntax at Santa Cruz*, ed. Séamas Mac Bhloscaidh, volume 3, 21–40. Santa Cruz: Linguistics Department, UC Santa Cruz.

Pullum, Geoffrey K. 1983. The revenge of the methodological moaners. *Natural Language and Linguistic Theory* 1:583–588.

Pullum, Geoffrey K. 2007. The evolution of model-theoretic frameworks in linguistics. In *Model-Theoretic Syntax @ 10*, ed. James Rogers and Stephan Kepser, 1–10.

Pullum, Geoffrey K., and Barbara C. Scholz. 2001. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In *Logical aspects of computational linguistics: 4th international conference*, ed. Philippe de Groote, Glyn Morrill, and Christian Retoré, number 2099 in Lecture Notes in Artificial Intelligence, 17–43. Berlin: Springer.

Quine, Willard V. O. 1995/1941. Whitehead and the rise of modern logic. In *Selected logic papers*, 3–36. Cambrdige, Mass.: Harvard University Press.

Rabin, Michael O. 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society* 141:1–35.

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

Raoult, Jean-Claude. 1997. Rational tree relations. *Bulletin of the Belgian Mathematical Society* 4:149–176.

Reinhart, Tanya. 2006. *Interface strategies: Optimal and costly computations*. Cambridge, Mass.: MIT Press.

Rezac, Milan. 2007. Escaping the person case constraint: Reference-set computation in the ϕ-system. *Linguistic Variation Yearbook* 6:97–138.

Rizzi, Luigi. 1990. *Relativized minimality*. Cambridge, Mass.: MIT Press.

Rizzi, Luigi. 1997. The fine-structure of the left periphery. In *Elements of grammar*, ed. Liliane Haegeman, 281–337. Dordrecht: Kluwer.

Rizzi, Luigi. 2004. Locality and left periphery. In *The cartography of syntactic structures*, ed. Adriana Belletti, volume 3, 223–251. New York: Oxford University Press.

Roberts, Ian. 2010. *Agreement and head movement: Clitics, incorporation, and defective goals*. Cambridge, Mass.: MIT Press.

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. Syntactic structures as multi-dimensional trees. *Research on Language and Computation* 1:265–305.

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Vischer, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In *The mathematics of language*, ed. Christan Ebert, Gerhard Jäger, and Jens Michaelis, volume 6149 of *Lecture Notes in Artificial Intelligence*, 255–265. Heidelberg: Springer.

Rogers, James, and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* 20:329–342.

Ross, John R. 1967. *Constraints on variables in syntax*. Doctoral Dissertation, MIT.

Rounds, William C. 1970. Mappings on grammars and trees. *Mathematical Systems Theory* 4:257–287.

Salvati, Sylvain. 2011. Minimalist grammars in the light of logic. In *Logic and grammar — essays dedicated to Alain Lecomte on the occasion of his 60th birthday*,

ed. Sylvain Pogodalla, Myriam Quatrini, and Christian Retoré, number 6700 in Lecture Notes in Computer Science, 81–117. Berlin: Springer.

Schütze, Carson. 1997. *INFL in child and adult language: Agreement, case and licensing*. Doctoral Dissertation, MIT.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88:191–229.

Shieber, Stuart M. 2004. Synchronous grammars as tree transducers. In *TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, 88–95.

Shima, Etsuro. 2000. A preference for Move over Merge. *Linguistic Inquiry* 375–385.

Sipser, Michael. 2005. *Introduction to the theory of computation*. Course Technology, second edition.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.

Stabler, Edward P. 1999. Remnant movement and complexity. In *Constraints and resources in natural language syntax and semantics*, ed. Gosse Bouma, Geert-Jan M. Kruijff, Erhard Hinrichs, and Richard T. Oehrle, 299–326. Stanford, CA: CSLI Publications.

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

Stabler, Edward P. 2006. Sidewards without copying. In *Formal Grammar '06, Proceedings of the Conference*, ed. Gerald Penn, Giorgio Satta, and Shuly Wintner, 133–146. Stanford: CSLI Publications.

Stabler, Edward P. 2011. Computational perspectives on minimalism. In *Oxford handbook of linguistic minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

Stabler, Edward P. 2012. Bayesian, minimalist, incremental syntactic analysis. Ms., UCLA.

Stabler, Edward P., and Edward Keenan. 2003. Structural similarity. *Theoretical Computer Science* 293:345–363.

Sternefeld, Wolfgang. 1996. Comparing reference-sets. In *The role of economy principles in linguistic theory*, ed. Chris Wilder, Hans-Martin Gärtner, and Manfred Bierwisch, 81–114. Berlin: Akademie Verlag.

Szendrői, Kriszta. 2001. *Focus and the syntax-phonology interface*. Doctoral Dissertation, University College London.

Thatcher, James W. 1967. Characterizing derivation trees for context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences* 1:317–322.

Thatcher, James W., and J. B. Wright. 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory* 2:57–81.

Toivonen, Ida. 2001. *On the phrase-structure of non-projecting words*. Doctoral Dissertation, Stanford, CA.

Travis, Lisa. 1984. *Parameters and effects of word order variation*. Doctoral Dissertation, MIT.

Vijay-Shanker, K., and David J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* 27:511–545.

Wartena, Christian. 2000. A note on the complexity of optimality systems. In *Studies in optimality theory*, ed. Reinhard Blutner and Gerhard Jäger, 64–72. Potsdam, Germany: University of Potsdam.

Weir, David. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science* 104:235–261.

Changelog

v1.1 - 2013/10/11 - first widely distributed version

- new section “Adding Head Movement and Affix Hopping”
- added corresponding bullet point to Advanced MG topics summary
- new appendix “Basic Mathematics”
- new appendix “Formal Definitions”
- added references to appendix throughout thesis
- added list of figures and list of examples
- each example now has a title
- moved “Symbols and Abbreviations” from end to beginning, added it to toc
- some layout tweaks for front matter
- fix some unfortunate omissions in the acknowledgments
- changed caption of Fig 4.7 to lower caps
- fixed some typos
- typo: $=$v instead of $=\nu$
- corrected a labeling error in example “An MG for the copy language”
- expanded title of section 3.4
- fixed definition of TP in example 3.4

v1.0 - 2013/06/03 - filed version