ABSTRACT

Title of dissertation: A STRUCTURAL THEORY OF DERIVATIONS

Zachary Stone
Doctor of Philosophy, 2018

Dissertation directed by: Professor Howard Lasnik
Department of Linguistics

Operations which take in tuples of syntactic objects and assign them output syntactic objects are used to formalize the generative component of most formal grammars in the minimalist tradition. However, these models do not usually include information which relates the structure of the input and output objects explicitly. We develop a very general formal model of grammars which includes this structural change data, and also allows for richer dependency structures such as feature geometry and feature-sharing. Importantly, syntactic operations involving phrasal attachment selection, agreement, licensing, head-adjunction, etc. can all be captured as special kinds of structural changes, and hence we can analyze them using a uniform technique.

Using this data, we give a rich theory of isomorphisms, equivalences, and substructures of syntactic objects, structural changes, derivations, rules, grammars, and languages. We show that many of these notions, while useful, are technically difficult or impossible to state in prior models. It is immediately possible to define grammatical notions like projection, agreement, selection, etc. structurally in a manner preserved under equivalences of various sorts. We use the richer structure of syntactic objects to give a novel characterization of c-command naturally arising from this structure. We use the richer structure of rules to give a general theory of structural analyses and generating structural changes. Our theory of structural analyses makes it possible to extract from productions what structure is targeted by a rule and what conditions a rule can apply in, regardless of the underlying structure of syntactic objects or the kinds of phrasal and featural manipulations performed, where other formal models have difficulty incorporating such structure-sensitive rules. This knowledge of structural changes also makes it possible to extend rules to new objects straightforwardly. Our theory of structural changes allows us to deconstruct them into component parts and show relationships between operations which are missed by models lacking this data.

Finally, we extend the model to a copying theory of movement. We implement a traditional model of copying 'online', where copies and chains are formed throughout the course of the derivation (while still admitting a feature calculus in the objects themselves). Part of what allows for this is having a robust theory of substructures of derived objects and how they are related throughout a derivation. We show consequences for checking features in chains and feature-sharing.

A STRUCTURAL THEORY OF DERIVATIONS

by

Zachary Stone

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Dr. Howard Lasnik, Chair/Advisor
Dr. Tim Hunter
Dr. Paul Pietroski
Dr. Georges Rey
Dr. Juan Uriagereka

Dedication

To Joe B. and my parents.

Acknowledgments

I want to thank Howard Lasnik, my advisor, who is not only the greatest educator I have ever met, but manages to effortlessly integrate highly technical aspects of the field, its history, and the empirical wonder of Natural Language, and is a willing copyeditor nonpareil. I am greatly indebted to Paul Pietroski, with his amazing ability to pierce through the dense mathematics of my proposals and extract the interesting and practical aspects of it. To linguistic researchers like Ed Keenan, Ed Stabler, Tim Hunter, Greg Kobele, Sylvain Salvati, and Chris Collins, and mathematicians like Alexander Grothendieck, Saunders Mac Lane, and Hermann Weyl, who not only laid the groundwork for the research presented here and showed me what was possible through formal study and gave me hope for its place in scientific history, but saw a marriage of mathematics and science not only for the sake of formalization, but for mathematics as a tool to transmute our intuitions into alchemical theoretical gold. To Mark Baltin, who turned me onto linguistics as a naïve art student. And to my undergraduate advisor Ray Dougherty, who I see as one of the great philosophers of linguistics, biolinguistics, and semiotics. To Chris Neufeld and Jon Beardsley, who managed to encourage me and smile through hours of me explaining half-baked ideas. And finally, my parents Deborah Baron and Winston Stone, who I love very much and have always supported me, despite not knowing what I'm on about 95% of the time.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Background and Motivation

The transformational machinery of classical generative grammar relied on two core notions: (1) a structural analysis (SA), and (2) a structural change (SC) [1]. In that theory, grammatical 'structure' was given in terms of (sets of) strings. The SA described - in terms of precedence - the relative positions of terminal and nonterminal symbols which must hold in order to perform a transformation. For example, the *passive* transformation applied to a phrase marker when one could find a string in that phrase marker of the form *NP* - *Aux* - *V* - *NP*. A SC was given as a rewrite rule, which could refer to symbols in the SA, which usually permuted, deleted, inserted, or adjoined symbols. Indexing the four symbols, the SC associated to the passive transformation replaced $NP_1 - Aux_2 - V_3 - NP_4$ with $NP_4 - Aux_2 + be + en - V_3 - by + NP_1$. Qualitatively, this rule rearranges the NPs, while inserting symbols corresponding to passive morphology and the by-phrase.

Modern syntactic theory manipulates hierarchical structure directly. In formal Minimalist Grammars (MGs) [2], these objects are ordered trees together with feature data associated to the nodes, while Bare Phrase Structure (BPS) encodes hierarchical information using the $\in$ relation of set theory.

In this thesis, we sketch a formal model of SAs and SCs for hierarchically structured objects. This model is developed in great generality, to accommodate more nuanced morphosyntactic objects and SCs, such as those with feature geometry and those with feature relations like feature sharing. With this more nuanced structure, we can give theories of rules in terms of the SCs they induce and what grammatical relations they lead to. The mathematical formalism is situated in a category theoretic framework which admits 'good' notions of embeddings, equivalences, and isomorphisms at various levels, such as derived syntactic objects (DSOs), derivations of them, rules manipulating them, as well as entire languages. The model avoids many technical limitations and undesirable formal properties of MGs and BPS while also generalizing over them.

Like many instances of formalizing scientific theory, the purpose of this model is to make more precise many folkloric notions in linguistics in a general setting which is amenable to incorporating many proposals about syntactic structure and syntactic operations. Most importantly, these notions can often be unified under a single theory, and we can prove that general reasoning techniques are always possible under certain simple assumptions. We describe some specific applications below.

Common sense notions of when two derived objects are 'structurally identical', or, at a higher level, when SCs, rules, derivations, or languages are, can be unified. Once a model (category) of DSOs has been fixed, the theory given here will return the relevant notions of isomorphisms and substructures of those DSOs, as well as SCs between DSOs, and hence derivations, isomorphisms and substructures of derivations, finally trickling up to strong extensional equivalences between languages. This extends the research program outlined by Stabler and Keenan [3,4].

Given a particular grammatical operation - assignments of SCs to tuples of input DSOs - the theory given here can also *extract* from the rules the 'context' a rule applies in - that is, what 'structural configurations' the rule cares about in terms of dominance and syntactic category (including minimality), precedence, or whatever other structure the DSOs are framed in terms of. Additionally, in many cases we can describe a small set of 'generating' SCs - canonical SCs which all other SCs are based on, which reduces many of the constructions to those found in Graph Grammars [5]. By analyzing these generating SCs, we can make precise notions like the Inclusiveness Condition and the Extension Condition [6].

Generally, formal grammar models of natural language eschew structural constraints in the level of detail that is used by generative syntacticians, while semi-formal models of minimalist syntax often formalize these structural conditions on rules in idiosyncratic ways, not always stating them precisely in terms of the underlying formalism. We unify programs studying structural contexts a rule can apply in and what they do to structure, dating back to *Syntactic Structures* [1] and continuing into minimalism where representations are hierarchical such as in Rizzi [7].

It is also often possible to deconstruct rules into component parts. For example, ARGUMENT-MERGE might involve PHRASAL-ATTACHMENT and SELECTION while AGREEING-SPECIFIER-MERGE might involve phrasal-attachment and AGREE, showing that the operations have an attachment component in common, though they differ in terms of how they manipulate features. This generalizes results and observations from research like that in Hunter [8]. We will present a general theory for how to deconstruct rules using the SC data.

Flipping the question of rules and SCs around, we can give a general theory of grammatical relations *in terms of the SCs induced*. In simple cases, these can be ordered by 'connectivity', making precise how PHRASAL-ATTACHMENT *plus* SELECTION brings a phrase closer to a head than simply PHRASAL-ATTACHMENT. In this way, many of the core relations between parts of a derivation and DSOs like specifier, complement, adjunct, projection, etc. can be captured relativistically, dating back to observations in "Categories & Transformations", [9] and Muysken [10].

Finally, we also extend these results to research on copying in a derivational language, building on work by Kobele [11]. We will resolve some issues of formally implementing chains and copies in derivations, including showing algebraic constraints which induce simultaneous valuation of all elements in a chain. We construct a formal model of copying which aligns with the more traditional view of copying happening 'online' during the derivation, cf. Kobele [11].

In summary, given any proposal about what the structure of DSOs is and what SCs the rules assign, we give a general theory for how to extract from that proposal what the isomorphisms, substructures, grammatical relations, SAs and SCs, derivations, etc. are, as well has how to compare languages built with them. Many aspects of these research programs are subsumed under unifying algebraic methods, while also generalizing to allow for structures such as feature sharing [12,13], Mirror Theory [14], and nanosyntax [15].

1.2 Organization of This Thesis

We start with a general overview of most aspects of the theory in §1.3.

Chapter 2 focuses mainly on introducing category theory to the reader as it will be used to describe derived syntactic objects and relations between them, such as isomorphisms and embeddings. §2.4 gives example 'categorifications' of models from the literature. The definitions for categories, isomorphisms, functors, constructs and concrete categories, embeddings, natural transformations, and representable functors occur in this chapter, along with many examples of them. The primary original results of this chapter are in §2.5.2, which gives a novel description of c-command in terms of constituent-preserving maps between hierarchically structured objects. Specifically, Claim 2.5.4 shows that there is a tight relations between constituent-preserving morphisms and a description of c-command in terms of a negation operator.

Chapter 3 introduces the main contribution of this research - structural changes and derivations which include SC data. §3.2 gives an overview of example SCs and how they can be formalized, as well as the 'dual' notion of grammatical relations. The characterization of grammatical relations given here is novel, and allows for a systematic description of how certain grammatical relations are 'closer' than others in terms of the dependencies formed. §3.3 introduces a naïve model of derivations, isomorphisms of derivations, and functors relating DSOs and derivations. §3.3.3 applies properties of the model to precisely describe grammars which recursively construct derivations, as well as ways to formalize equivalences and isomorphisms of grammars and languages. Claim 3.3.7 is one of the simple but significant results of the section. It shows that for recursively constructed languages, there is a close relation between (a) an induced notion of equivalence between languages and (b) isomorphisms between lexical items and structural changes in the grammars generating them. §3.4 contains a very important aside on adjoint functors in category theory, which unifies many constructions given up to that point, and will be used heavily in subsequent sections. §3.5 illustrates some technical mathematical issues with the naïve model, and introduces the better-behaved model which will be used for the remainder of the thesis. Much of that section is dedicated to proving that this category of derivations is in fact well-behaved as a category of 'structured sets', and recreates many of the basic recursive constructions necessary for recursively constructing languages from grammars. Claim 3.5.7 summarizes most of these 'good' properties.

Chapter 4 contains an in-depth study of rules and SCs. §4.2 gives a very general method for describing what structure a given rules 'cares about' when determining where it is to be applied; that is, it gives a way to extract a *structural analysis* from the rule by looking at all of the productions associated to that rule. It also contains a description and proof of the fundamental *Pushout Lemma*. §4.3 then describes when it is possible to generate a rule from a finite set of basic SCs. Examples from our model as well as the literature are given in this section. §4.3.1 contains a brief digression into properties of rules, namely formalizations of what it means for a rule to meet the Inclusiveness and Extension Conditions. §4.3.2 works through an application of the theory to an existing Minimalist Grammar formalization building dependency trees [16]. This section gives examples of extracting a structural analysis out of a MERGE rule, as well as how to apply a basic SC in context for richly structured objects. §4.4 gives a theory of how to *compile* many structural changes into a single one, giving a much more general theory of deconstruction of rules into component parts as developed in Hunter [8]. This will allow us to analyze what properties different operations have in common in terms of the structure they introduce. In particular, we give examples for the basic local MERGE operations, as well as a comparison between local and long-distance agreement in sections §4.4.5 and §4.4.6.

Chapter 5 is the last chapter of original theory, extending the model to include 'true copying' and 'chain' information. After giving a review of prior formal models of movement in §5.1, we proceed to develop a model in terms of structured derivations in §5.2. We conclude with an example from Greek which shows how the same technology which keeps track of chains automatically induces simultaneous valuation effects throughout a chain.

Finally, Chapter 6 compares properties of the theory developed in the thesis to existing theories. We show certain undesirable formal properties of existing models, as well as restrictions on developing a rich theory of language within them along the lines of the preceding chapters. The chapter concludes with reflections on benefits of the model developed here and how they were achieved, as well as how it admits other models in the literature as special cases.

1.3 Overview of the Theory

To not lose the forest for the trees, we give a brief overview of the whole project. Recall that we are leaving *open* what the basic model of the DSOs should be (though we will eventually require that it meet some very general axioms). However, upon fixing the DSOs, the rest of the theory can be deployed straightforwardly.

We first sketch a formal theory of derived syntactic objects (DSOs), and then move on to a model of derivations of them. We view derived syntactic objects as 'structured sets': sets of nodes together with dependency information, or other information related to syntactic type or feature-calculus. We might encode dependency information using (directed or undirected) graph structure, for example, or an ordering relation. See Fig. 1.1.

We might want to endow these sets with additional data. For example, we could have syntactic type information (N, V, wh, ϕ, ...) about the nodes, or precedence relations between them.[1] We might also include chain data as a relation between a node and the nodes of which it is a copy. If our dependency structures

[1] Models such as those in Chomsky [17] included linear order as part of the structure of syntactic objects. Many models like Bare Phrase Structure do not include linear order as part of the syntactic structure. Versions of Minimalist Grammars like in Stabler & Keenan [4] manipulate (tuples of) strings directly, which can be seen as precedence orders. It is not obvious how Bare Phrase Structure encodes syntactic type, but Minimalist Grammars encode it by having a generating set of basic syntactic types. Most Minimalist Grammars, such as in Stabler & Keenan [4] and Boston, et al. [16] also partition the syntactic objects into 'components' which are used as stacks for movement.

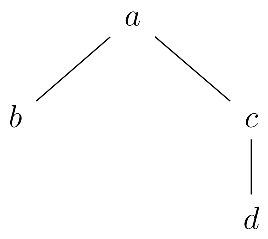
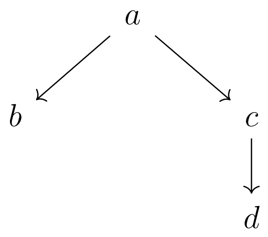
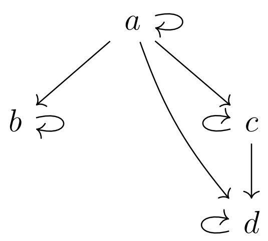
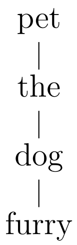
	Example	Definition	$V = \{a, b, c, d\}$
Graph		A set V of vertices together with a set E of 2-element subsets of V.	$E = \{\{a, b\}, \{a, c\}, \{c, d\}\}$
Directed Graph		A set V of vertices together with a subset $E \subset V \times V$ of edges (i.e. a binary relation on V).	$E = \{(a, b), (a, c), (c, d)\}$
Preorder		A set V of vertices together with a binary relation $\leq$ such that: 1. For all $a \in V$, $a \leq a$ 2. For all $a, b, c \in V$, $a \leq b$ and $b \leq c$ imply $a \leq c$	$a \leq a$, $a \leq b$, $a \leq c$, $a \leq d$, $b \leq b$, $c \leq c$, $c \leq d$, $d \leq d$

Figure 1.1: Mathematical structures used to model dependencies



$$\text{pet} \preceq \text{the} \preceq \text{furry} \preceq \text{dog}$$
$$V(\text{pet}) = D(\text{the}) = N(\text{dog}) = \mathit{Adj}(\text{furry}) = \text{true}$$

Figure 1.2: An example DSO described by dominance, precedence, and syntactic typing data.

include features, we might have information which tells us if the features are still active, or the order that they must be checked in. In all of these cases, the DSOs are ‘sets of nodes with extra structure’. For example, we give in Fig. 1.2 a DSO representing ‘pet the furry dog’ given with dominance, precedence, and syntactic type information.[2]

Given a model of DSOs, we will have induced notions of isomorphisms (special bijections) and substructure embeddings (special subset inclusions). The basic idea behind isomorphisms is that they are mutually inverse bijections which preserve the relevant structure in both directions; the basic idea behind a substructure embedding is that it is a subset inclusion such that the subset has ‘as much structure as possible’ such that the inclusion preserves structure. Examples of these are given in Fig. 1.3 and Fig. 1.4, respectively.

Structural changes (SCs) can be represented as tuples of functions (defined on the sets of nodes) from a tuple of DSOs into a new DSO. These functions must preserve certain structure, like dominance and precedence, and such functions are

[2] Here, the dominance relation could be modeled by any of the structures in Fig 1.1. We write a precedence relation using a preordering $\preceq$, and we consider syntactic types as unary predicates on the set of nodes. Assume that the determinations are false unless indicated otherwise.

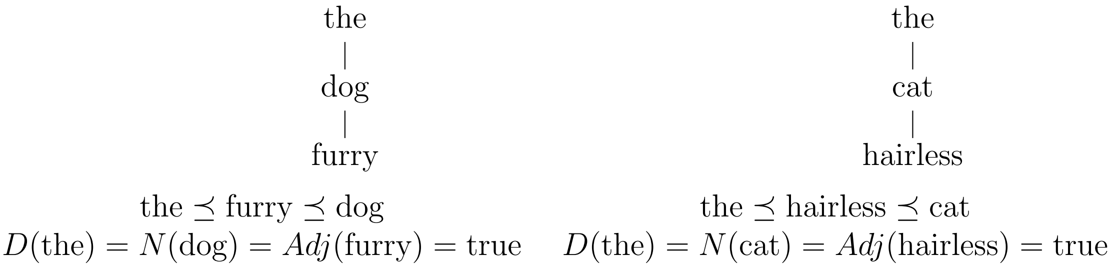

the
dog
furry
the ≼ furry ≼ dog
D(the) = N(dog) = Adj(furry) = true
the
cat
hairless
the ≼ hairless ≼ cat
D(the) = N(cat) = Adj(hairless) = true

Figure 1.3: Two isomorphic DSOs

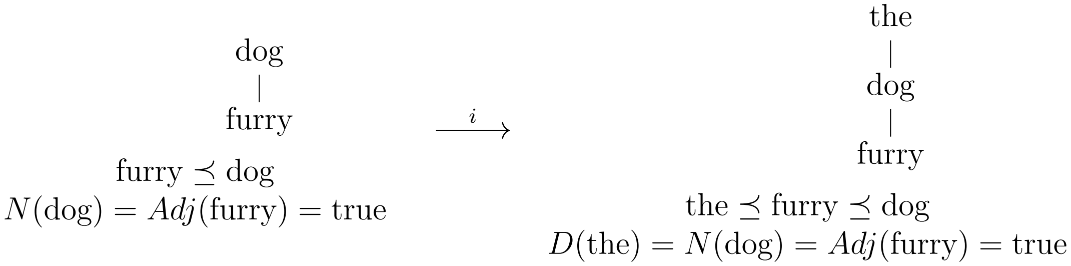

dog
furry
furry ≼ dog
N(dog) = Adj(furry) = true
i
the
dog
furry
the ≼ furry ≼ dog
D(the) = N(dog) = Adj(furry) = true

Figure 1.4: An example substructure embedding.

referred to as *morphisms*. In Fig. 1.5, we have a pair of DSOs ('her parents', 'pet the furry dog') mapping into the DSO 'her parents pet the furry dog'. However, it is not simply the case that we have assigned this output DSO to this pair: the pair of functions f and g between sets of nodes explicitly map nodes of the input trees to nodes of the output tree, e.g. the node for 'dog' in the input DSO is mapped to a corresponding node for 'dog' in the output DSO by g. Moreover, f and g 'preserve the structure' of the input DSOs: e.g., since we have the precedence relation 'her $\preceq$ parents' in the input DSO, so do we have the precedence relation 'f(her) $\preceq$ f(parents)' in the output DSO; since the node 'pet' in the input DSO is of syntactic type V, so is g(pet) in the output DSO; since we have the dominance relation 'pet $\leq$ the' in the input DSO, we also have the dominance relation 'g(pet) $\leq$ g(the)' in the output DSO; and so on for each precedence and dominance relation and syntactic type determination.

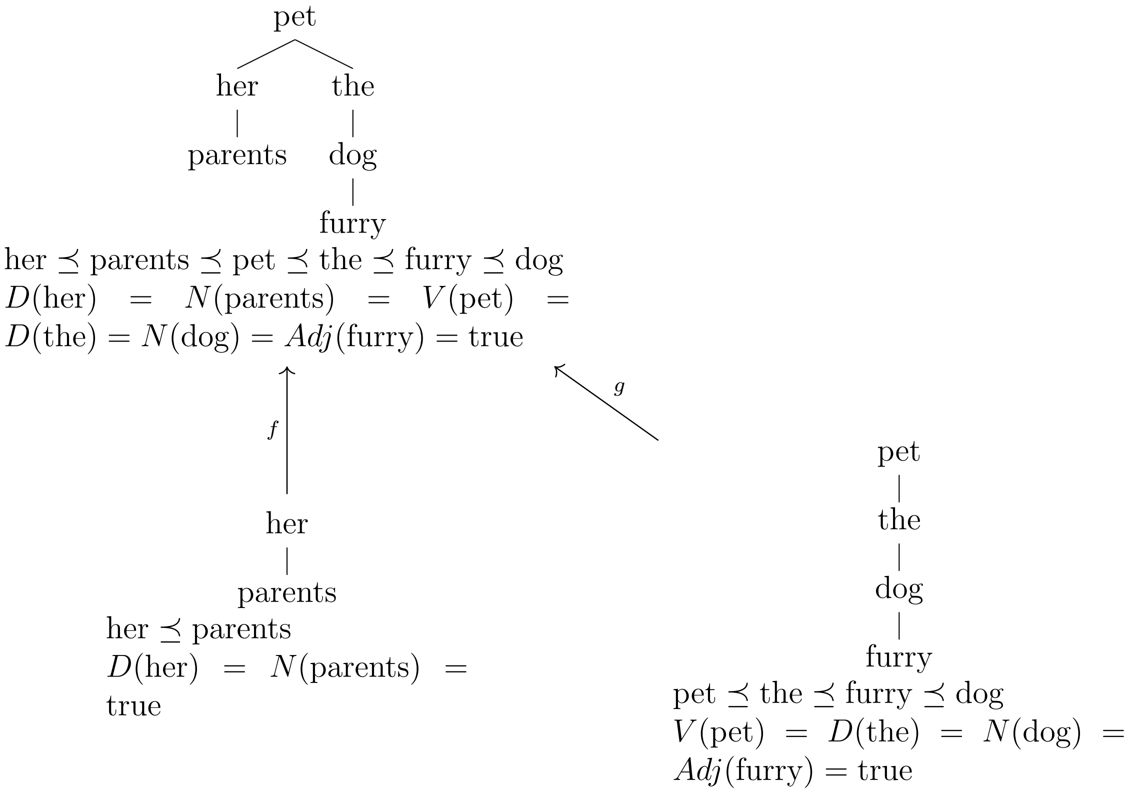

pet
her
the
parents
dog
furry
her $\preceq$ parents $\preceq$ pet $\preceq$ the $\preceq$ furry $\preceq$ dog
D(her) = N(parents) = V(pet) = D(the) = N(dog) = Adj(furry) = true
f
g
her
parents
her $\preceq$ parents
D(her) = N(parents) = true
pet
the
dog
furry
pet $\preceq$ the $\preceq$ furry $\preceq$ dog
V(pet) = D(the) = N(dog) = Adj(furry) = true

Figure 1.5: SCs mapping two DSOs into a new DSO.

In this way, by representing DSOs as ‘structured sets of nodes’, SCs between DSOs are simply ‘functions between sets of nodes which preserve that structure’. A *derivation* is essentially a diagram of sequences of DSOs linked by SCs. This has many technical advantages. Such a model of derivations includes in it the ‘obvious’ but useful information that associates prior DSOs with subobjects of later DSOs. For example, we can tell just by looking at the set-theoretic image that ‘her parents’ corresponds to a specific subpart of the DSO ‘her parents pet the furry dog’. It is in fact the part of the tree which *became a dependent* of the element that ‘pet’ mapped to. Projection can be read off of this structure: we can tell that it is the head of the DSO ‘pet the furry dog’ (and not ‘her parents’) which projects, simply by noting that it is the root of this DSO which maps to the root of the output DSO. This gives a primitive theory of projection and ‘derivational’ dependency/argument structure. Our theory of grammatical relations (§3.2.1) will essentially be a measurement of how much heads and features of them come to be dependent on each other in later DSOs ‘over’ the SCs which link them.

If our DSOs are more rich and include features, we can get subtler relations between DSOs and their parts, leading to a proportionally richer theory of grammatical relations. For example, if we have gender features ϕ and ψ, we may represent ϕ getting its value from ψ by becoming dependent on it - i.e. by introducing the relation $\phi \leq \psi$.

In Fig. 1.6, we iteratively attach adjuncts and add a dependency relation from each adjunct’s ϕ-feature to that of the head noun. This actually leads to an (asymmetric) feature-sharing structure [12, 13, 18], and we can again read the

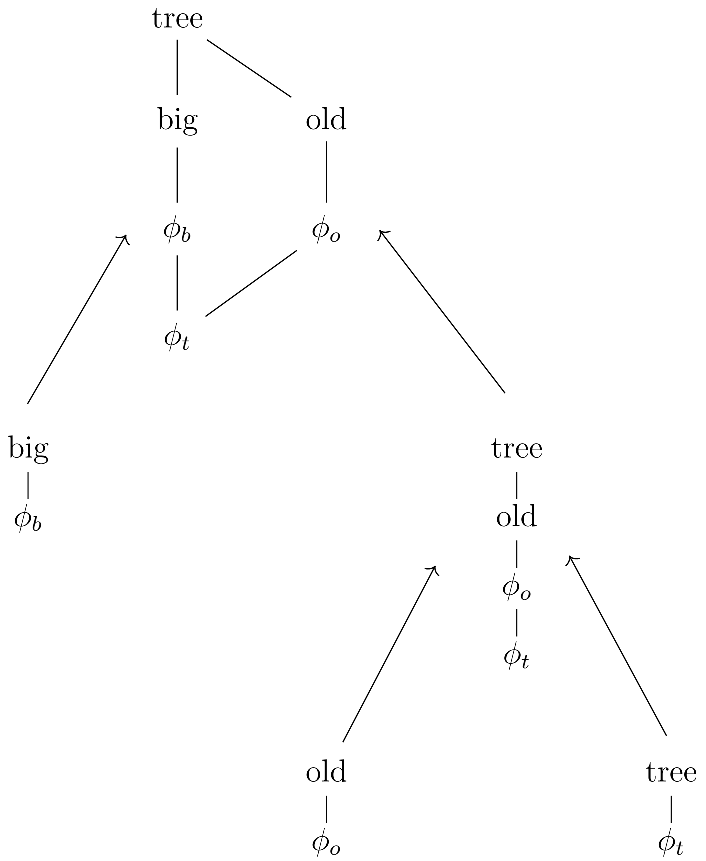


Figure 1.6: Iterative applications of agreement, leading to a feature-sharing structure

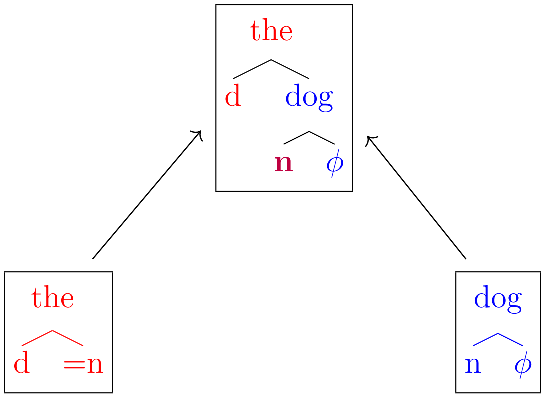


Figure 1.7: Selection as identification of features

agreement relation off of the derivation just by looking at images of ϕ-features and the dependencies introduced. We could construct a similar derivation involving case features. For that derivation, we would have the correct result that, by transitivity of domination/dependence, if the case feature of the head noun gets valued via dependence, all of the adjuncts' case features would also be dependent on this valuing feature. Feature-sharing is a straightforward model of concord and such 'delayed-valuation' effects involving concord. Such an example is worked out in §5.3.

We could similarly add features for syntactic selection. In this case, we could use a different relationship - such as identification of features - to indicate a different degree of 'derivational connectivity' between DSOs to distinguish selection from agreement or licensing. An example of this is given in Fig. 1.7.

We view a grammatical *rule* as an assignment of tuples of SCs mapping to a common target DSO to tuples of input DSOs. Since rules will be embellished with this SC-data, we can then give a theory of rules in terms of the structure they introduce. For example, f and g of Fig. 1.5 intuitively add a dominance relation from 'pet' to 'her', add precedence relations from 'her' and 'parents' to 'pet', 'the', 'furry', and 'dog', plus all of the relations induced by these relations under transitivity, while leaving all other structure alone. Each instance of the 'adjoin-and-agree' SCs of Fig. 1.6 intuitively adds a dependency from the head noun to the adjunct, while also adding a dependency from the ϕ-feature of the adjunct to the ϕ-feature of that same head noun. Using the SC data associated to rules, we can give a theory of when a rule is 'generated' by some primitive SC applied in some restricted context in general. These contexts are like the SAs of classical transformational grammar, abstracted to more richly structured DSOs. The process of going from a rule (the collection of all 'productions' associated to that rule) to a set of primitive generators is the 'structured' version of going from a set of productions to a set of generating replacement rules in string-based grammars. This method is closely related to the single-pushout method of Algebraic Graph Grammars [5]. In this form, we can study the *conditions* under which a rule can apply and what structure it targets (roots, nearest feature of type x up to relativized minimality, etc.). We can then deconstruct a rule in terms of each piece of structure it changes (adding dominance or precedence relations, deactivation of features, etc.). For example, the SC in Fig. 1.7 can be deconstructed into (1) attaching the roots, and (2) identifying the selection features.

Since all of our theory is stated in terms of structured sets, we may take advantage of all of the mathematical methodology for handling such objects. In particular, we showed how it becomes easy to describe substructure embeddings and isomorphisms between DSOs and derived objects using standard mathematical definitions, and these trickle up to good notions of subderivations and equivalences of languages in terms of the structure of DSOs and SCs a grammar may build. We extend isomorphisms of DSOs to isomorphisms of SCs. An isomorphism between two SCs is simply an isomorphism between each DSO involved, such that those isomorphisms 'commute' with the SCs. More generally, take a *derivation* to be a diagram of DSOs linked by morphisms. Maps between derivations are essentially collections of morphisms between DSOs which are compatible with SCs. In particular, two derivations Δ and Γ are isomorphic if and only if there is an exact correspondence between the DSOs occurring in them, and we are given an isomorphism $f : D \rightleftharpoons G : g$ between each pair of DSOs (D, G) in correspondence which preserves the SCs. This last condition means that for any SC $k : D \to D'$ in Δ with $j : G \to G'$ the SC between the corresponding DSOs in Γ, the isomorphisms $f : D \to G$ and $f' : D' \to G'$ carry k to j isomorphically. In other words, $f'k = jf$, where f and f' are isomorphisms. For example, there is an isomorphism between the (fragments of) derivations in Fig. 1.5 and Fig. 1.8. For more general morphisms of derivations, we drop the requirement that the component morphisms $f, f', \ldots$ are isomorphisms. We will similarly be able to recover a notion of embeddings of subderivations as morphisms of derivations which are special subset inclusions.

Finally, these isomorphisms between SCs, DSOs, and derivations give rise to a notion of *strong extensional equivalence of languages*. A language can be viewed as a collection of derivations. Two languages $\mathscr{L}$ and $\mathscr{M}$ can be said to be *equivalent* if whenever there is a derivation Δ in one, there is an isomorphic derivation Γ in the other, and conversely. This is weaker than requiring a bijection between the languages, and hence allows a characterization of equivalence between languages of

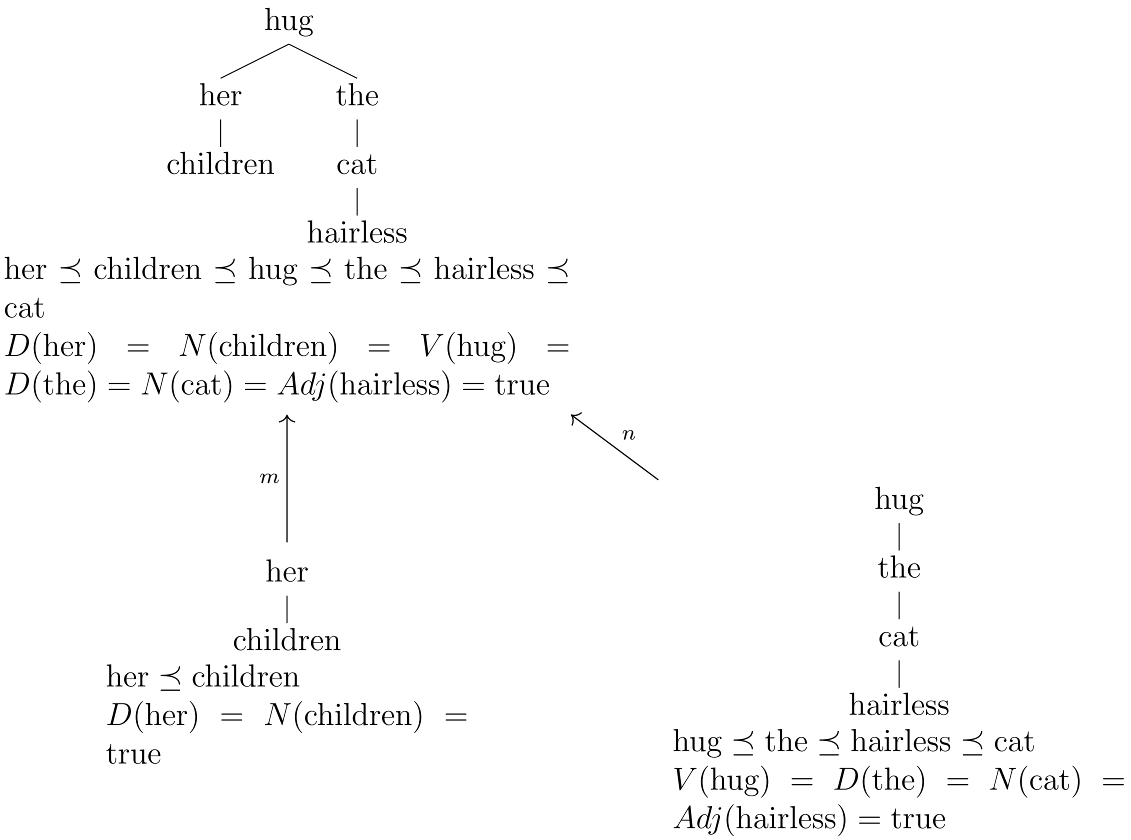

hug
her
the
children
cat
hairless
her ≼ children ≼ hug ≼ the ≼ hairless ≼ cat
D(her) = N(children) = V(hug) = D(the) = N(cat) = Adj(hairless) = true
m
n
her
children
her ≼ children
D(her) = N(children) = true
hug
the
cat
hairless
hug ≼ the ≼ hairless ≼ cat
V(hug) = D(the) = N(cat) = Adj(hairless) = true

Figure 1.8: A derivation isomorphic to that in Fig. 1.5.

different sizes with lexicons of different cardinality, so long as for every DSO in one, there is some isomorphic DSO in the other, and similarly for the SCs.

Chapter 2: Derived Syntactic Objects

2.1 Overview

The main function of this chapter is to introduce basic category theory as we will use it to model derived syntactic objects (DSOs). Intuitively, DSOs are hierarchically structured objects, possibly with other data like precedence, syntactic typing, or other information related to the feature calculus such as feature activity.[1] We will characterize the example DSO models in Fig. 1.1 in terms of categories, and then introduce the notion of isomorphisms in categories. We then introduce (representably) concrete categories, which are roughly categories of 'sets with structure'. Given this set-structure, it becomes possible to talk about structure *restricted to a*

[1] Throughout, we make no commitment to whether precedence information is in the syntax or not. Early models like Chomsky [17] and Barker & Pullum [19] included precedence order as part of the structure of a DSO. Models like Kayne [20] included precedence order, though it was totally determined by hierarchical structure. Modern Bare Phrase Structure [21] has no linear order in the syntax, and recovers it for PF from hierarchy, while many formal minimalist grammars still include linear order information (and in some cases, there is *no hierarchical structure* in the DSOs) [22]. Similar observations can be made about many other structural assumptions about the DSOs, such as syntactic type, information about whether a feature is active or not, etc. We will develop a theory in generality where it works both for objects with, e.g., precedence data and without.

subset, commonly called a *substructure embedding*. We then give examples from the literature and show how they can be reinterpreted as categories of DSOs. Finally, we conclude with some specialized results for order-theoretic models of DSOs, and show that there is a deep connection between c-command and constituency. This last section contains the only 'new results' presented in this chapter, with the preceding material mostly intended as an introduction to category theory as it can be applied to the analysis of syntactic objects.

2.2 Categories of DSOs

We start with some examples of mathematical structures (graphs, directed graphs (digraphs), preorders) which could be used to model DSOs in Fig 1.1. In each case, we can define *morphisms* between objects of the same kind, which we think of as preserving certain properties of the structure, given in Fig. 2.1. These morphisms are often called ***graph homomorphisms***, ***directed graph homomorphisms***, and ***order-preserving maps***, respectively. In each case, the class of objects (graphs, digraphs, preorders), together with for each pair of objects A and B, a set $\mathbf{Hom}(A, B)$ of homomorphisms, together with for each triple A, B, and C a composition function $\circ : \mathbf{Hom}(A, B) \times \mathbf{Hom}(B, C) \to \mathbf{Hom}(A, C)$ constitutes a ***category***.

Definition 2.2.1. A ***category*** is a class of objects $\mathcal{C}$, together with a set $\mathcal{C}(A, B)$ of morphisms for each pair of objects (A, B) with $A, B \in \mathcal{C}$, together with a composition function $\circ : \mathcal{C}(A, B) \times \mathcal{C}(B, C) \to \mathcal{C}(A, C)$ for each triple (A, B, C) with

	Morphism	Composite
Graph	Given two graphs (V_G, E_G) and (V_H, E_H), a morphism f is given by a function $f_V : V_G \to V_H$ such that for each $\{a, b\} \in E_G$, we have $\{f_V(a), f_V(b)\} \in E_H$	Given graphs (V_G, E_G), (V_H, E_H), and (V_I, E_I), and morphisms $f_V : V_G \to V_H$ and $g_V : V_H \to V_I$, then the composite $g_V \circ f_V : V_G \to V_I$ gives a morphism, taking a node $a \in V_G$ to $g_V(f_V(a))$.
Directed Graph	Given two digraphs (V_G, E_G) and (V_H, E_H), a morphism f is given by a function $f_V : V_G \to V_H$ such that for each $(a, b) \in E_G$, we have $(f_V(a), f_V(b)) \in E_H$	Given digraphs (V_G, E_G), (V_H, E_H), and (V_I, E_I), and morphisms $f_V : V_G \to V_H$ and $g_V : V_H \to V_I$, then the composite $g_V \circ f_V : V_G \to V_I$ gives a morphism, taking a node $a \in V_G$ to $g_V(f_V(a))$.
Preorder	Given two preorders $(P, \leq_P)$ and $(Q, \leq_Q)$, a morphism f is given by a function $f : P \to Q$ such that if $a \leq_P b$ in P, then $f(a) \leq_Q f(b)$	Given partial orders $(P, \leq_P)$ and $(Q, \leq_Q)$, and $(R, \leq_R)$ and morphisms $f : P \to Q$ and $g : Q \to R$, then the composite $g \circ f : P \to R$ gives a morphism.

Figure 2.1: Homomorphisms between objects

$A, B, C \in \mathcal{C}$. These data are subject to the following axioms:

1. For every object A, there is a morphism $1_A \in \mathcal{C}(A, A)$, called the identity on A, such that given any morphisms $f \in \mathcal{C}(A, B), g \in \mathcal{C}(B, A)$ the following equalities hold: $f \circ 1_A = f, 1_A \circ g = g$

2. Given morphisms $f \in \mathcal{C}(A, B), g \in \mathcal{C}(B, C), h \in \mathcal{C}(C, D)$ the following equality holds: $h \circ (g \circ f) = (h \circ g) \circ f$.

Borceux [23] 1.2.1

It is important to note that in defining a category, both the objects and morphisms must be given. There is a motto in category theory that it is the morphisms which are actually important, while the objects don't matter that much.[2] That is, where in set theory we might define a structure by building up a set and defining certain operations or relations on it, in category theory it is the arrows which determine the relevant structure. The 'structure' of an object is essentially what remains unchanged by the arrows, and most important properties of an object (its elements, substructures, isomorphisms between structures) and constructions on objects (Cartesian products, disjoint unions, intersections, quotients) can all be stated entirely in terms of properties of morphisms, without making any assumptions about the objects. In this way, it is useful to think of giving a class of morphisms as roughly the same as giving an axiomatic description of the relevant properties which determine a DSO.

[2] "The knowledge of the maps is equivalent to the knowledge of the interior structure of an object." Pareigis [24], p. 3

Many basic facts about morphisms can be proven in any category. We give a simple example.

Claim 2.2.1. If $x, y \in \mathcal{C}(A, A)$ are identities on A, then $x = y$

Proof. $x \circ y = x$ since y is an identity, and $x \circ y = y$ since x is an identity. We denote the identity on A by 1_A. □

We denote the above categories as **Grph**, **DGrph**, and **Proset**. We prefix each of these categories with **F** for the subcategory of objects with underlying sets of nodes which are finite. In each case, we could add various kinds of data to the objects and construct an associated notion of morphism and composite to get a category.

1. We could put a 'PF ordering' (precedence relation) on the vertices. Concretely, we define a precedence relation on a graph G, digraph γ, or finite partial order P as a preorder $\preceq$ on its underlying set of nodes, such additionally (1) $\preceq$ is antisymmetric - $a \preceq b$ and $b \preceq a$ imply $a = b$, and (2) $\preceq$ is total - for any a, b, either $a \preceq b$ or $b \preceq a$.[3] We can construct morphisms as graph, digraph, or order-preserving morphisms ϕ, where additionally for any vertices $a \preceq b$, we have $\phi a \preceq \phi b$. In each case, using the obvious function compositions gives a category.

[3] However, we will later want to allow $\preceq$ to be an arbitrary preorder. Part of the reason is so that we can describe 'disjoint unions' of structures with precedence relations, where we do not want to have to introduce ordering relations between the summands. Similarly, the reason we characterize precedence as reflexive is to allow gluings/identification of elements in a structure.

2. For any set $\mathbf{L}$, we can construct categories of (directed) graphs or preorders (partially) labeled by $\mathbf{L}$. Concretely, we add (partial) labelling data to a structure with set of nodes V with a (partial) function $f : V \to \mathbf{L}$ from the underlying set of nodes. A morphism ϕ of (partially) labeled (directed) graphs or preorders is a (directed) graph homomorphism or order-preserving map such that if a is a vertex with label $f(a)$, then the label of $\phi(a)$ is $f(a)$.

3. We can equip a (directed) graph or preorder with set of nodes V with a predicate $\alpha : V \to \{\text{true}, \text{false}\}$ on the underlying nodes. For example, we could say that if $\alpha(a) = \text{true}$, then a is 'inactive'. If (A, α) and (B, β) are (directed) graphs or preorders with predicates, we can define a morphism to be a morphism ϕ of the relevant type, such that if $\alpha(a) = \text{true}$, then $\beta(\phi(a)) = \text{true}$. We could also use a set of such predicates for syntactic typing instead of labels, i.e. use a set of predicates V, N, wh, ... such that V(x) is true if x is verbal, etc.

Adding any combination of the structures above to (directed) graphs or preorders has an associated notion of morphism which gives a category.

In any category, we have a notion of ***isomorphism (iso)***.

Definition 2.2.2. A morphism $f : X \to Y$ in a category is called an ***isomorphism*** (iso) if there exists a morphism $g : Y \to X$, called its ***inverse***, such that $f \circ g = 1_Y$ and $g \circ f = 1_X$.

We give examples of isos.

the
dog
furry
the
cat
hairless

the $\preceq$ furry $\preceq$ dog
$f_X(\text{the}) = D$, $f_X(\text{dog}) = N$,
$f_X(\text{furry}) = Adj$
$\xi(\text{the}) = \text{false}$, $\xi(\text{dog}) = \xi(\text{furry}) = \text{true}$

the $\preceq$ hairless $\preceq$ cat
$f_Y(\text{the}) = D$, $f_Y(\text{cat}) = N$,
$f_Y(\text{hairless}) = Adj$
$\iota(\text{the}) = \text{false}$, $\iota(\text{cat}) = \iota(\text{hairless}) = \text{true}$

Figure 2.2: Two isomorphic DSOs

1. An iso between graphs G and H is a bijection between the underlying sets of nodes $f : V_G \to V_H$ such that there is an edge $\{a, b\}$ in G iff $\{fa, fb\}$ is an edge in H.

2. An iso of digraphs G and H is a bijection $f : V_G \to V_H$ such that there is an edge (a, b) in G iff (fa, fb) is an edge in H.

3. An iso of preorders P and Q is a bijection between their underlying sets, such that $p \leq_P p'$ iff $fp \leq_Q fp'$.

4. An iso between preorders with precedence relation, partially labeled by $\mathbf{L}$, with a unary predicate $(P, \leq_P, \preceq_P, f_P : P \to \mathbf{L}, \pi)$ and $(Q, \leq_Q, \preceq_Q, f_Q : Q \to \mathbf{L}, \kappa)$ is a bijection $\phi : P \to Q$ such that (1) $a \leq_P b$ iff $\phi a \leq_Q \phi b$; (2) $a \preceq_P b$ iff $\phi a \preceq_Q \phi b$; (3) a has label $f_P(a)$ iff $\phi(a)$ has label $f_Q(\phi a) = f_P(a)$; (4) $\pi(a) = \text{true}$ iff $\kappa(\phi(a)) = \text{true}$.

We give an explicit example of an isomorphism from the last category in Fig. 2.2. The bijection associating *the* with *the*, *dog* with *cat*, and *furry* with *hairless* is an isomorphism.

We can similarly prove the following in any category.

Claim 2.2.2. Two objects A and B in some category $\mathcal{C}$ ***are isomorphic*** if there exists some isomorphism $f : A \to B$ between them. We write this relation $A \approx B$. This relation is an equivalence relation, in that $A \approx A$ for any A, $A \approx B$ if and only if $B \approx A$, and $A \approx B$ and $B \approx C$ imply $A \approx C$.

Proof. For any object A, the identity $1_A : A \to A$ gives an isomorphism which is its own inverse. If we have an isomorphism $f : A \to B$, then by definition there is an inverse $g : B \to A$ which is also an isomorphism. Given isomorphisms $f : A \to B$ and $g : B \to C$ with inverses $f' : B \to A$ and $g' : C \to B$, $g \circ f : A \to C$ is an isomorphism with inverse $f' \circ g'$ since $(f' \circ g') \circ (g \circ f) = f' \circ 1_B \circ f = f' \circ f = 1_A$, and conversely $(g \circ f) \circ (f' \circ g') = 1_C$. □

Claim 2.2.3. In any category $\mathcal{C}$, if $f : A \to B$ is an isomorphism, it has exactly one inverse $g : B \to A$.

Proof. By the definition of an isomorphism, f must have some inverse. Let $x : B \to A$ and $y : B \to A$ both be inverses to f. Then, $x \circ f = y \circ f = 1_A$, so $x = x \circ 1_B = x \circ (f \circ x) = y \circ (f \circ x) = y \circ 1_B = y$. □

2.3 Representably Concrete Categories of DSOs

In all of the categories discussed so far, the objects seem to be 'sets with extra structure'. To formalize this, we want to say that each of the categories bears a special relationship to **Set**, the category of sets and set functions given below.

The class of sets together with the set $\mathbf{Set}(A, B)$ of functions from A to B for any sets A and B forms a category **Set**. Composition is given by standard set-function composition: given functions $f : A \to B$ and $g : B \to C$, for any $a \in A$, we have $(g \circ f)(a) = g(f(a))$. The identity morphisms must be the identity functions, e.g. $1_X : X \to X$ is the function such that $1_X(x) = x$ for any $x \in X$.

Relationships between categories are given by *functors*. A functor $F : \mathcal{C} \to \mathcal{D}$ is intuitively a mapping from the objects of $\mathcal{C}$ to the objects of $\mathcal{D}$ together with a mapping from the morphisms of $\mathcal{C}$ to the morphisms of $\mathcal{D}$ in a manner which is compatible with composition.

Definition 2.3.1. A ***functor*** F from a category $\mathcal{A}$ to a category $\mathcal{B}$ consists of the following:

(1) a mapping $|\mathcal{A}| \to |\mathcal{B}|$ between the classes of objects of $\mathcal{A}$ and $\mathcal{B}$; the image of $A \in \mathcal{A}$ is written $F(A)$ or just FA;

(2) for every pair of objects A, A' of $\mathcal{A}$, a mapping $\mathcal{A}(A, A') \to \mathcal{B}(FA, FA')$; the image of $f \in \mathcal{A}(A, A')$ is written $F(f)$ or just Ff.

These data are subject to the following axioms:

(1) for every pair of morphisms $f \in \mathcal{A}(A, A'), g \in \mathcal{A}(A', A'')$: $F(g \circ f) = F(g) \circ F(f)$

(2) for every object $A \in \mathcal{A}$: $F(1_A) = 1_{FA}$

Borceux [23] 1.2.2

It can be checked that the 'obvious' definition for the composition $G \circ F : \mathcal{A} \to \mathcal{C}$ of functors $F : \mathcal{A} \to \mathcal{B}$ and $G : \mathcal{B} \to \mathcal{C}$, taking an object A to $G(F(A))$ and morphism f to $G(F(f))$, is in fact a functor. For any category $\mathcal{A}$, there is an identity functor $1_{\mathcal{A}} : \mathcal{A} \to \mathcal{A}$, fixing every object and morphism of $\mathcal{A}$. It then makes sense to define an *isomorphism of categories*.

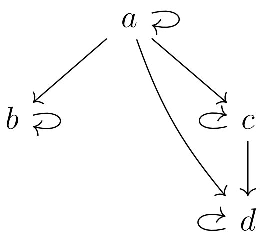
Definition 2.3.2. A functor $F : \mathcal{C} \to \mathcal{D}$ is an ***isomorphism of categories*** if there is a functor $G : \mathcal{D} \to \mathcal{C}$ such that $G \circ F = 1_{\mathcal{C}}$ and $F \circ G = 1_{\mathcal{D}}$.

We can describe these relations between the categories above precisely as functors between them. We have the following examples of functors:

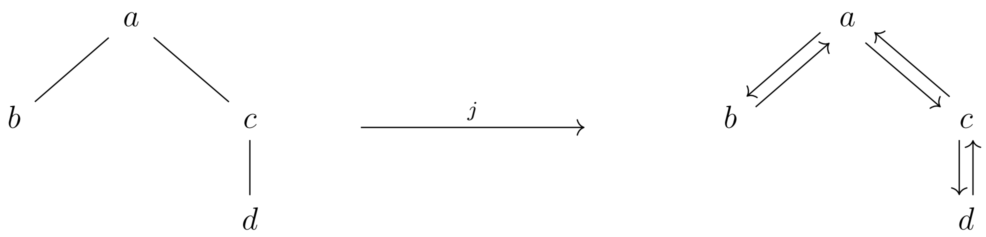
1. i : **FProset** → **DGrph**. It sends a preorder $(P, \leq_P)$ to the graph with vertices P and edge relation $\leq_P \subset P \times P$. It sends an order-preserving map ϕ to the graph homomorphism acting the same way on underlying sets of vertices.

   $\{a, b, c, d\}$
   $a \leq a$, $a \leq b$, $a \leq c$, $a \leq d$, $b \leq b$, $c \leq c$, $c \leq d$, $d \leq d$

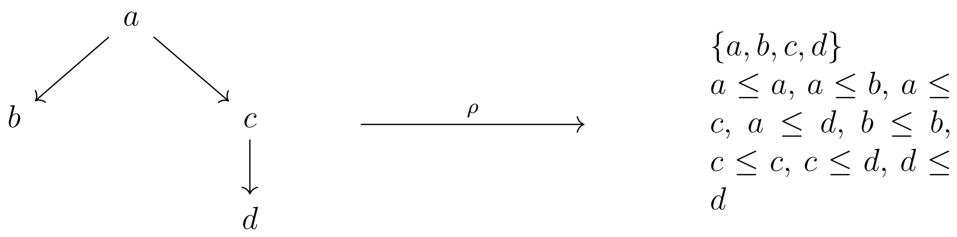
   $\xrightarrow{i}$



2. j : **Grph** → **DGrph**. It sends a graph G to the digraph with vertices V_G, such that for each edge $\{a, b\} \in E_G$, we have (a, b) and (b, a) in E_{jG}. j takes a graph homomorphism $f : G \to H$ to a digraph homomorphism since $\{a, b\}$ implies $\{fa, fb\}$ and hence (fa, fb) and (fb, fa).



3. $\rho : \mathbf{DGrph} \to \mathbf{Proset}$. To each digraph G, we can associate a preorder with underlying set of nodes V_G, and we take the smallest preorder $\leq_G \subset V_G \times V_G$ containing E_G, sometimes called the *reachability relation* generated by E_G. If $f : G \to H$ is a morphism of digraphs, then $\rho(f)$ is order-preserving. That is, if $a \leq_G b$ is in the reachability relation generated by E_G, then $\rho(f)(a) = f(a) \leq_H f(b) = \rho(f)(b)$ is in the in the reachability relation generated by E_H, so $\rho(f)$ is order-preserving from $\rho(G)$ to $\rho(H)$.



Functors can be classified by their behavior on morphism-sets. A functor $F : \mathcal{C} \to \mathcal{D}$ is ***faithful*** if morphisms in $\mathcal{C}$ can be essentially seen as 'special $\mathcal{D}$-morphisms'. F is ***full*** if every $\mathcal{D}$-morphism corresponds to *some* $\mathcal{C}$-morphism.

Definition 2.3.3. A functor $F : \mathcal{C} \to \mathcal{D}$ is ***faithful*** if every induced set-function $\mathcal{C}(C, C') \to \mathcal{D}(FC, FC')$, taking $C \xrightarrow{f} C'$ to $FC \xrightarrow{F(f)} FC'$, is injective - i.e. if $F(f) = F(g)$ iff $f = g$.

Definition 2.3.4. A functor $F : \mathcal{C} \to \mathcal{D}$ is ***full*** if every induced set-function $\mathcal{C}(C, C') \to \mathcal{D}(FC, FC')$, taking $C \xrightarrow{f} C'$ to $FC \xrightarrow{F(f)} FC'$, is surjective, - i.e. if

$FC \xrightarrow{h} FC'$ is any morphism in $\mathcal{D}(FC, FC')$, then there is some morphism $C \xrightarrow{f} C'$ in $\mathcal{C}(C, C')$ such that $F(f) = h$.

We can now make precise what we mean by $\mathcal{C}$ being a category of 'sets with extra structure'.

Definition 2.3.5. A category $\mathcal{C}$ together with a functor $U : \mathcal{C} \to \mathcal{D}$ is said to be ***concrete over*** $\mathcal{D}$ if U is faithful. A category $U : \mathcal{C} \to \mathbf{Set}$ concrete over $\mathbf{Set}$ is called a ***construct***.

Adámek, et al. [25], Ch. 5.

The intuition behind a construct is that U maps each object C of $\mathcal{C}$ to its underlying set UC. Since the behavior on morphism-sets is injective, the collection of morphisms $\mathcal{C}(C, C')$ can be put in correspondence with a subset of all possible functions from UC to UC' - the designated 'special' set-functions which preserve the $\mathcal{C}$-structure. Each of the categories above is a construct using the functor $U : \mathcal{C} \to \mathbf{Set}$ taking a graph, digraph, or preorder to its underlying set of nodes. That $\mathcal{C}(C, C') \hookrightarrow \mathbf{Set}(C, C')$ is an injection for each pair of objects (C, C') says that each morphism is totally determined by a function 'which meets certain conditions', but we need not keep track of any data other than the function Uf to know which morphism we are referring to.

Claim 2.3.1. Let $U : \mathcal{C} \to \mathbf{Set}$ be any construct, and let $f : A \to B$ be any isomorphism in $\mathcal{C}$. Then $U(f)$ is a bijection.

Proof. This is a special case of a more general claim: any functor preserves isomorphisms, in that if f is an isomorphism, then $U(f)$ is an isomorphism. To see this, simply note that if $f : A \to B$ is an isomorphism with inverse f', then $F(f') \circ F(f) = F(f' \circ f) = F(1_A) = 1_{FA}$ and $F(f) \circ F(f') = F(f \circ f') = F(1_B) = 1_{FB}$ by the definition of a functor. It then just remains to show that the isomorphisms of **Set** are exactly the bijections, but this follows straightforwardly from invertability. □

We can then view isomorphisms in a construct as bijections f, such that f and its inverse function are both morphisms. In a construct, it also makes sense to define substructure embeddings. Intuitively, given an object A and subset $S \subset UA$, the substructure on S, if it exists, is an object $\mathcal{S}$ such that $U\mathcal{S} = S$ such that the inclusion $S \hookrightarrow UA$ underlies a morphism $i : \mathcal{S} \to A$ such that $\mathcal{S}$ has 'as much structure as possible' such that the inclusion underlies a morphism.

Definition 2.3.6. A morphism $f : A \to B$ in a category $\mathcal{C}$ is called a ***monomorphism*** when, for every object $C \in \mathcal{C}$ and every pair of morphisms $g, h : C \rightrightarrows A$, the following property holds: $(f \circ g = f \circ h) \Rightarrow (g = h)$. Borceux [23] 1.7.1

Such morphisms are often called *left-cancellable*. We have the following examples:

1. A function $f : A \to B$ in **Set** is a monomorphism iff it is injective.

2. An order-preserving function $f : P \to Q$ in **Proset** is a monomorphism iff it is injective on the underlying sets.

We can now describe embeddings in any concrete category.

Definition 2.3.7. Let $U : \mathbf{A} \to \mathbf{X}$ be a faithful functor.

1. An $\mathbf{A}$-morphism $f : A \to B$ is called ***initial*** provided that for any $\mathbf{A}$-object C, an $\mathbf{X}$-morphism $g : UC \to UA$ is an $\mathbf{A}$-morphism whenever $(Uf) \circ g : UC \to UB$ is an $\mathbf{A}$-morphism.

2. An initial morphism $f : A \to B$ that has monomorphic underlying $\mathbf{X}$-morphism $Uf : UA \to UB$ is called an ***embedding***.

Adámek, et al. [25], 8.6

When $\mathbf{X} = \mathbf{Set}$, that is, when $(\mathbf{A}, U)$ is a construct, this simplifies to the following definition.

Definition 2.3.8. Let $U : \mathbf{A} \to \mathbf{Set}$ be a construct. An $\mathbf{A}$-morphism $f : A \to B$ is called an ***embedding*** provided that $Uf : UA \to UB$ is an injective function with the following property: for any $\mathbf{A}$-morphism $g : C \to B$ such that the set-theoretic image of g is contained in $f(UA) \subset UB$, then the unique function $g^A : UC \to UA$ such that $Uf \circ g^A = Ug$ underlies an $\mathbf{A}$-morphism.

We give an example. Let $\mathbf{A}$ be the category whose objects are sets of nodes X, together with a domination preorder $\leq$, precedence preorder $\preceq$, and partial labeling function $f_X : X \to \mathbf{L} = \{D, N, Adj\}$. Let the morphisms of this category be functions $\phi : (X, \leq_X, \preceq_X, f_X) \to (Y, \leq_Y, \preceq_Y, f_Y)$ from X to Y such that if $a \leq_X b$, then $\phi(a) \leq_Y \phi(b)$, if $a \preceq b$ then $\phi(a) \preceq \phi(b)$, and $f_Y(\phi(a)) = f_X(a)$ whenever $f_X(a)$ is defined. This category forms a construct using the functor U taking $(X, \leq_X, \preceq_X, f_X)$ to the set X, and taking each morphism to the underlying set function. We give an example of an embedding, as well as a monomorphism

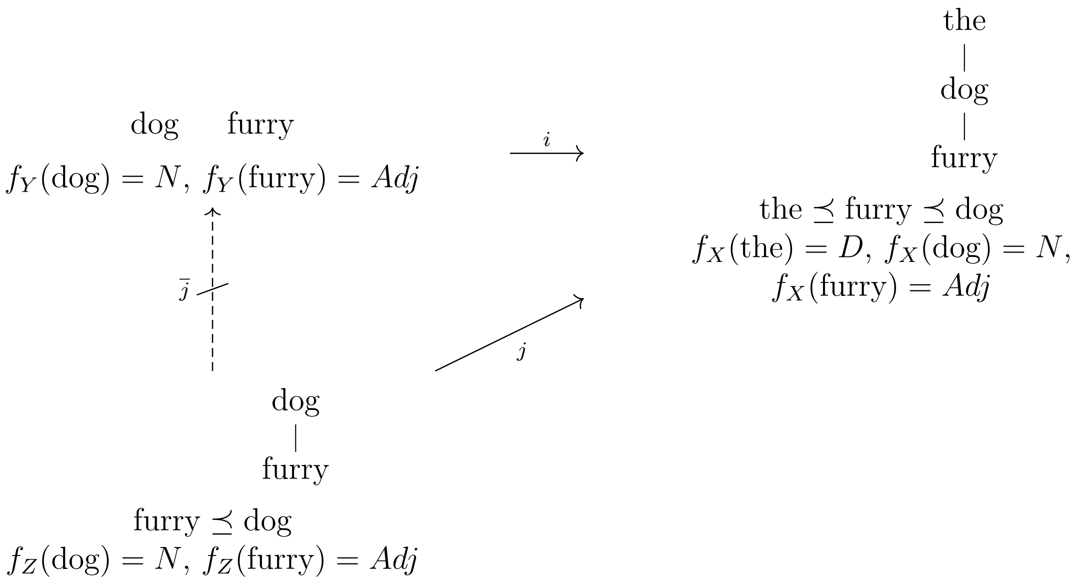

the
dog
furry
the ⪯ furry ⪯ dog
f_X(the) = D, f_X(dog) = N,
f_X(furry) = Adj
dog furry
f_Y(dog) = N, f_Y(furry) = Adj
i
$\bar{j}$
j
dog
furry
furry ⪯ dog
f_Z(dog) = N, f_Z(furry) = Adj

Figure 2.3: An example of two morphisms whose underlying function is a subset inclusion. However, only j is an embedding.

which is *not* an embedding, in Fig. 2.3. Only j is an embedding. That i is not an embedding can be seen since the image of j factors uniquely through the object in the upper left corner via the function $\bar{j}$, but $\bar{j}$ is not a morphism, since it does not preserve precedence and dominance relations from the object in the lower left corner. To see that j is an embedding, note that if $k : A \to X$ is any morphism into X whose image is contained in the subset $\{\text{dog}, \text{furry}\}$, there will be a unique factorization of k through j, call it $\overline{k} : A \to Z$. $\overline{k}$ acts exactly as k does on the elements of A. $\overline{k}$ will be a morphism since k preserves labels, dominance, and precedence.

We give more examples of embeddings below.

1. In (**D**)**Grph**, a (directed) graph homomorphism $i : G \to H$ is an embedding if and only if it is an injective function, such that $a, b \in V_G$ have an edge between them (there is an edge (a, b)) iff ia and ib do (iff (ia, ib) is an edge in H).

2. In **FProset**, an order-preserving function $i : P \to Q$ is an embedding iff it is an injective function, such that $p \leq_P p'$ iff $ip \leq_Q ip'$.

3. Let **A** be the category of finite preorders with a precedence order, partially labeled by **L** with a single unary predicate on the vertices. The functor $U : \mathbf{A} \to \mathbf{Set}$ taking $(P, \leq_P, \preceq_P, f_P : P \to \mathbf{L}, \pi)$ to P turns **A** into a construct. A morphism $i : (P, \leq_P, \preceq_P, f_P : P \to \mathbf{L}, \pi) \to (Q, \leq_Q, \preceq_Q, f_Q : Q \to \mathbf{L}, \kappa)$ is an embedding iff (1) $i : P \to Q$ is an injective function; (2) $a \leq_P b$ iff $ia \leq_Q ib$; (3) $a \preceq_P b$ iff $ia \preceq_Q ib$; (4) $f_P(a)$ is defined iff $f_Q(ia)$ is defined and has value $f_Q(ia) = f_P(a)$; (5) $\pi(a) = \text{true}$ iff $\kappa(ia) = \text{true}$.

2.3.1 Representability

We say that a category $\mathcal{C}$ is *concretizable* if there is some functor $U : \mathcal{C} \to \mathbf{Set}$ which is faithful. However, it turns out that this is not a very strong requirement. Whenever the collection of all morphisms in $\mathcal{C}$ form a set, $\mathcal{C}$ will be concretizable using a method which forms sets indexed over the whole category. We will now look at a more restrictive notion of concretizability which corresponds with an intuitive notion of returning a ‘set of elements’.

We first note that the elements of any set X are in a canonical correspondence with the functions from any fixed one-point set $\{*\}$ into X. We simply associate to each element $x \in X$ the function $x : \{*\} \to X$ taking $*$ to x. Conversely, each function $f : \{*\} \to X$ can be associated to the element $f(*) \in X$. We then have a bijection $X \approx \mathbf{Set}(\{*\}, X)$ for any set X. For any category $\mathcal{C}$ and fixed object $\blacksquare$ of $\mathcal{C}$, we will have an associated functor $\mathcal{C}(\blacksquare, -) : \mathcal{C} \to \mathbf{Set}$. At each object C of $\mathcal{C}$, this functor will have the value $\mathcal{C}(\blacksquare, C)$, which is by definition a set. Moreover, given a morphism $f : C \to C'$ of $\mathcal{C}$, there is a naturally induced function $f \circ - : \mathcal{C}(\blacksquare, C) \to \mathcal{C}(\blacksquare, C')$ taking any element $k : \blacksquare \to C \in \mathcal{C}(\blacksquare, C)$ to $f \circ k : \blacksquare \to C' \in \mathcal{C}(\blacksquare, C')$. For example, let $\blacksquare$ be the directed graph with one node and no edges. These is a canonical correspondence between elements of $\mathbf{DGrph}(\blacksquare, G)$ and the nodes of G: each morphism $k : \blacksquare \to G$ can be associated with the node $k(\blacksquare)$; conversely, each node g of G can be associated with the graph homomorphism $k : \blacksquare \to G$ sending $\blacksquare$ to g. In other words, for each graph G, we have a canonical bijection $UG \approx \mathbf{DGrph}(\blacksquare, G)$. Similarly, let $\mathbf{A_L}$ be the category of finite sets of nodes X together with domination preorder $\leq$, precedence preorder $\preceq$, and partial labeling function $f : X \to \mathbf{L}$. Let $\blacksquare$ be the object of $\mathbf{A_L}$ consisting of one node, with the only possible dominance and precedence preorderings, such that the labeling partial function is undefined everywhere. Morphisms $k : \blacksquare \to A$ will again be in correspondence with nodes of A, since for any element $a \in A$, we may send $\blacksquare$ to a, and this must be a morphism since $a \leq a$, $a \preceq a$, and we do not have any labeling data to preserve.

We now want to make precise what we mean by there being *canonical* bijections between UA and $\mathbf{A}(\blacksquare, A)$ for any object A of $\mathbf{A}$. This is stated in category theory by saying that we have a *natural isomorphism* between the functors U and $\mathbf{A}(\blacksquare, -)$.

Definition 2.3.9. Given functors $F, G : \mathcal{C} \rightrightarrows \mathcal{D}$, a ***natural transformation*** $\eta : F \to G$ from F to G is a collection of $\mathcal{D}$-morphisms $\eta_C : FC \to GC$, one for each object of $\mathcal{C}$, which meet the condition: for any $\mathcal{C}$-morphism $f : C \to C'$, we have $\eta_{C'} \circ Ff = Gf \circ \eta_C$. Composition of natural transformations $\eta : F \to G$ and $\beta : G \to H$ is given componentwise, in that $(\beta \circ \eta)_C = \beta_C \circ \eta_C$.

based on Mac Lane [26], I.4., Borceux [23], 1.3.1

A natural transformation η then compares what F and G do at each object C inside the image category $\mathcal{D}$. For any functor $F : \mathcal{C} \to \mathcal{D}$, there is a natural transformation $1_F : F \to F$ whose component at each object X of $\mathcal{C}$ is just the identity $1_{FX} : FX \to FX$. This is clearly an identity in the sense that for any natural transformation $\eta : F \to G$ we have $\eta = \eta \circ 1_F$, and for any natural transformation $\beta : H \to F$ we have $\beta = 1_F \circ \beta$. It then makes sense to define a natural isomorphism between functors. Two functors $F, G : \mathcal{C} \rightrightarrows \mathcal{D}$ are naturally isomorphic if there are natural transformations $\eta : F \to G$ and $\beta : G \to F$ such that $\beta \circ \eta = 1_F$ and $\eta \circ \beta = 1_G$. We say that a functor $F : \mathcal{C} \to \mathbf{Set}$ is *representable* if it is naturally isomorphic to one of the form $\mathcal{C}(\blacksquare, -)$ for some object $\blacksquare$ of $\mathcal{C}$. In this case, we say that F is *represented* by $\blacksquare$. If F is representable, its representer is determined up to isomorphism, in that if $\blacksquare$ and $\bullet$ both represent F, then $\blacksquare \approx \bullet$.

There are many nice properties which hold of any representable functor. For example, if $f : A \to B$ is any monomorphism in $\mathcal{C}$, then $F(f) : FA \to FB$ is injective. This can be proven simply since each element of FA is in correspondence with a morphism $k : \blacksquare \to A$. Then, if we have two elements of FA corresponding to morphisms k and j, $F(f)(k) = f \circ k = f \circ j = F(f)(j)$ implies $k = j$ by the left-cancellability of f. We say that a category is *representably concrete* if its forgetful functor $U : \mathcal{C} \to \mathbf{Set}$ is representable. We give representations of each of our 'forgetful' functors from categories above.

1. The forgetful functor $U : \mathbf{DGrph} \to \mathbf{Set}$ is represented by $\mathbf{1}$, the digraph with one node and no edges.

2. The forgetful functor $U : \mathbf{Proset} \to \mathbf{Set}$ is represented by $\mathbf{1}$, the preorder with one node $*$ and relation $* \leq *$.

3. The forgetful functor $U : \mathbf{A} \to \mathbf{Set}$ where $\mathbf{A}$ is the category of preorders with precedence order, partially labeled by $\mathbf{L}$, with single unary predicate, and U returns the underlying set of the preorder, is representable. It is represented by the object $(*, \leq, \preceq, \epsilon, \lambda)$, where $*$ is a one-point set, ϵ is undefined everywhere, and $\lambda(*) = \text{false}$.

2.4 Categorifications of Existing Models

We conclude this section with a 'categorifications' of proposed models of syntactic objects from the literature. Barker [19] defines a tree as follows: a tree T is a 5-tuple $T = (N, L, \geq^D, <^P, \textsc{label})$, where

1. N is a finite set, the nodes of T

2. L is a finite set, the labels of T

3. $\geq^D$ is a reflexive, antisymmetric relation on N, the dominance relation of T

4. $<^P$ is an irreflexive, asymmetric, transitive relation on N, the precedence relation of T, and

5. LABEL is a total function from N to L, the labeling function of T

such that for all a, b, c, and d from N and some unique r in N (the ROOT NODE of T), the following hold:

1. The Single Root Condition: $r \geq^D a$
2. The Exclusivity Condition: $(a \geq^D b \vee b \geq^D a) \leftrightarrow \neg(a <^P b \vee b <^P a))$
3. The Nontangling Condition: $(a <^P b \wedge a \geq^D c \wedge b \geq^D d) \rightarrow c <^P d$

There are a number of categories one could define using trees T above as objects. We could take as morphisms pairs of functions $(f_N, f_L) : T \to S$ between trees such that:

1. $f_N : N_T \to N_S$ and $f_L : L_T \to L_S$ are functions
2. If $a \geq^D b$ in T, then $f_N(a) \geq^D f_N(b)$ in S[4]
3. If $a <^P b$ in T, then $f_N(a) <^P f_N(b)$ in S
4. The following diagram commutes:

$$\begin{tikzcd}
N_T \arrow[r, "f_N"] \arrow[d, "\textsc{label}_T"'] & N_S \arrow[d, "\textsc{label}_S"] \\
L_T \arrow[r, "f_L"'] & L_S
\end{tikzcd}$$

That is, $\textsc{label}_S \circ f_N = f_L \circ \textsc{label}_T$.

Composition of morphisms is given by composite of node and label functions. Call this category **A**. Labels which are not used still affect the structure of a tree. For

[4] We do not usually want to preserve the root, or constituent embeddings will not be morphisms.

example, the following two trees are nonisomorphic:

$$(\{*\},\{V\},* \geq^D *,\emptyset,\textsc{label}(*)=V) \not\approx (\{*\},\{V,C\},* \geq^D *,\emptyset,\textsc{label}(*)=V)$$

Both objects seem to correspond to the tree consisting of one node $*$ labeled by V, but since the number of labels is different, the two trees are nonisomorphic. In fact, the functor $U : \mathbf{A} \to \mathbf{Set}$ sending each object $T = (N_T, L_T, \geq^D, <^P, \textsc{label})$ to N_T and each morphism (f_N, f_L) to f_N gives an example of a non-faithful functor. This is because if l is a label of L_T not used by any node in N_T,[5] f_N will not determine where to send l - any assignment is possible.

If we restrict to the objects where $\textsc{label}$ is surjective so that every label in L_T is used, the category is equivalent to the following category $\mathbf{BP}$.[6] Objects are 4-tuples $T = (N, \geq^D, <^P, \sim)$ such that $\geq^D$ and $<^P$ are subject to the axioms above, and $\sim$ is an equivalence relation on N 'has the same label'. In this case, the forgetful functor taking each T to its set N_T of nodes is represented by $(*, \geq^D, <^P, \sim)$, where $*$ is a one-point set with dominance relation $* \geq^D *$, there are no precedence relations, and $* \sim *$.

However, the function of labels is often to distinguish nodes as N, V, C, T, etc. absolutely. We fix a set of labels L and construct $\mathbf{BP}_L$, the category of trees labeled by L. Its objects are trees T such that $L_T = L$, with morphisms above such that $f_L = 1_L$ is the identity on L. For each L, $\mathbf{BP}_L$ can be turned into a construct using the functor $N_{(-)} : \mathbf{BP}_L \to \mathbf{Set}$ taking each T to N_T and each $(f_N, 1_L)$ to f_N.

[5] That is, not in the image of $\textsc{label}$.

[6] Two categories $\mathcal{C}$ and $\mathcal{D}$ are equivalent if there exist functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ and natural isomorphisms $\eta : FG \to 1_{\mathcal{D}}$ and $\beta : 1_{\mathcal{C}} \to GF$

We give another example. In Stabler [27], trees are formalized as *expressions*. We simplify his definitions here. Given vocabulary items V and a set of base syntactic types *base*, Stabler constructs a collection $\mathscr{L}$ of labels. The *syntactic features* are partitioned as $F = (\mathit{base} \cup \mathit{select} \cup \mathit{licensors} \cup \mathit{licensees})$, defined as follows. $\mathit{select} = \{=\mathrm{x} \mid \mathrm{x} \in \mathit{base}\}$; $\mathit{licensees} = \{\text{-x} \mid \mathrm{x} \in \mathit{base}\}$; $\mathit{licensors} = \{+\mathrm{x} \mid \mathrm{x} \in \mathit{base}\}$. The set of labels are the regular expressions $\mathscr{L} = F^* \cdot V^*$. Expressions over $\mathscr{L}$ are defined as trees $\tau = (N_\tau, \triangleleft^*_\tau, \preceq^*, <^*_\tau, \textsc{label}_\tau)$ such that

1. N_τ is a finite set of nodes

2. $\triangleleft^*_\tau$ is a preordering on N_τ 'dominates'

3. $\preceq^*$ is a preordering on N_τ 'precedes'

4. $<^*$ is a preordering on N_τ 'projects over'

5. $\textsc{label}_\tau : L_\tau \to \mathscr{L}$ is a function, where L_τ are the leaves of N_τ (nodes which only dominate themselves).

   Such that

1. For any $x \in N_\tau$, the set of nodes $\{y \in N_\tau \mid y \triangleleft^*_\tau x\}$ is a linear order, and there is a unique element r such that $r \triangleleft^*_\tau a$ for all $a \in N_\tau$.

2. For any distinct nodes $a, b \in N_\tau$, either one dominates the other or one precedes the other, but never both.

3. $(\forall w, x, y, z) : (x \preceq^* y \wedge x \triangleleft^* w \wedge y \triangleleft^* z) \to (w \preceq^* z)$

4. $(\forall x) : ((\exists y)(x \lhd y)) \to ((\exists y)(\forall z \neq y)(x \lhd z \to y < z))$, where $\lhd$ and $<$ are the immediate 'dominates' and 'projects over' relations.

Morphisms $f : \tau \to \sigma$ should be functions $f_N : N_\tau \to N_\sigma$ which at least have the following properties: (1) $a \lhd_\tau^* b$ implies $f_N a \lhd_\sigma^* f_N b$; (2) $a \preceq_\tau^* b$ implies $f_N a \preceq_\sigma^* f_N b$; and (3) $a <_\tau^* b$ implies $f_N a <_\sigma^* f_N b$, which we call a 'map of unlabeled trees'. If we require that f_N commute with labeling, we get a very rigid category: we cannot alter the features at all. Instead, we should like to let the labels on heads vary in certain ways under morphisms.

We look at the structural changes Stabler uses to determine what the relevant morphisms are. We use the notation $[_< \tau, \sigma]$ to indicate an expression with immediate subtrees τ and σ in that precedence order, where the root of τ immediately projects over that of σ, with all of the relevant relations added to meet the axioms. We define a partial binary operation on expressions using Stabler's terminology.[7]

$\textsc{merge}(\tau, \sigma) =$

i. $[_< \tau', \sigma']$ if τ is a head with initial feature $= x$ and σ has feature x. Here, τ' is τ, but where the string of features α on the head of τ has had $= x$ removed from the front, and similarly x is removed from the front of the string of features β

[7] Given a tree τ, $x, y \in N_\tau$, x is a *head* of y iff either (1) y is a leaf and $x = y$; or (2) $(\exists z)(y \lhd z \wedge (\forall w)(y \lhd w \to z <^* w \wedge x$ is a head of $z)$. Every tree τ has a root r, and r has a unique head y, which is a leaf. This leaf is labeled by $\textsc{label}_\tau(y) \in \mathscr{L}$, which has as first part a string of features $\alpha \in F^*$. A tree τ *has feature* f if the first symbol in the string α is f. A *maximal projection* m in N_τ is an element minimal (with respect to $\lhd^*$) amongst the nodes with head equal to the head of m. τ is a *head* if it consists of one node, otherwise it is *complex*.

on the head of σ. (complement merger)

ii. $[_{>}\sigma', \tau']$ if τ is complex and has feature $= x$, and σ has feature x. σ' and τ' are as above. (base specifier merger)

We define a partial unary operation $\textsc{move}(\tau) = [_{>}\tau_0', \tau']$, defined iff (1) τ has feature +x; (2) τ has exactly one proper subtree τ_0 with feature -x; and (3) the root of τ_0 is a maximal projection in τ. τ_0' is like τ_0 with -x removed from the head of τ_0, and τ' is like τ except that +x is deleted from the head of τ and the subtree τ_0 is replaced by a leaf with no features (an unindexed trace).

For $\textsc{merge}$, the inclusions of unlabeled trees $m_1 : \tau \to \textsc{merge}(\tau, \sigma)$ and $m_2 : \sigma \to \textsc{merge}(\tau, \sigma)$ preserve $\lhd^*$, $\preceq^*$, and $<^*$, and in fact their immediate variants, and hence are maps of unlabeled trees. A leaf x of τ with label $\alpha \cdot \pi \in F^* \cdot V^*$ may have its initial feature removed, in that the image $m_1(x)$ will be a leaf of $\textsc{merge}(\tau, \sigma)$ whose feature string looks just like that of x, but without the first feature. With $\textsc{move}$, we have an map of unlabeled trees $\tau \to \textsc{move}(\tau)$, sending all nodes in τ_0 to a dummy 'trace' node; all features in τ_0 are removed under this map. We also have a map $\tau_0 \to \textsc{move}(\tau)$ embedding τ_0 into the moved position, with -x removed.

We would like to define morphisms which include composites of mappings of the above type, so that we can use these morphisms as SCs. We define a morphism $f : \tau \to \sigma$ to be a map of unlabeled trees such that if $x \in L_\tau$ has label $\alpha \cdot \pi$, then $f(x)$ is in L_σ with label $\alpha' \cdot \pi'$, where α' is α with some initial substring removed, and π' is π with some initial substring removed. This will allow composites of structural changes under $\textsc{merge}$ to be morphisms, since each step will remove more and more features from the front. This will also allow the replacement of a subtree of τ with a trace node while deleting features to be a morphism. We call this category (**Strict**)**MGExp**. The functor $N_{(-)} : \mathbf{SMGExp} \to \mathbf{Set}$ turns **SMGExp** into a construct. The 'subtrees' Stabler describes are embeddings in this construct. However, this is again a very 'strict' category, in that isomorphic objects must be identically labeled. Since the phonetic string π and specific features α are part of the label structure, any two sentences with different words will be nonisomorphic. We can weaken this in various ways by allowing more morphisms between objects, though if we are allow multiple comparisons between feature-structures with the same map between underlying unlabeled trees, then the category will not be concrete under the functor taking an MG-object to its set of nodes.

2.5 Constituency and C-Command

We will now restrict to DSOs given by a set X together with a *partial* dominance ordering on X. A partial ordering is a preorder which meets an *antisymmetry* condition: if $x \leq y$ and $y \leq x$, then we have $x = y$. Such objects have an algebraic representation, and there is a close connection between constituency and c-command in this form. We denote the category of partial orders, sets X with a fixed partial ordering $\leq_X$, together with order-preserving functions between them as **Pos**. We call a partial order *finite* if its underlying set of nodes is finite, and denote the category of finite partial orders and order-preserving functions by **FPos**. We first characterize trees and forests, decomposition of forests into trees, and then characterize a relationship between constituency and c-command in trees.

2.5.1 Forests and trees in **FPos**

We define trees and forests in this section, and describe how to systematically decompose forests into trees in a unique way.

Definition 2.5.1. A partial order P is a ***linear order*** iff for every pair of elements $x, y \in P$, either $x \leq y$ or $y \leq x$.

Definition 2.5.2. A finite partial order P is a ***forest of trees*** if for every $x \in P$, $\{y \in P \mid y \leq x\}$ is a linear order.

Definition 2.5.3. A forest of trees P is a ***tree*** if it has a minimum $z \in P$, such that $z \leq x$ for all $x \in P$.

The property of being a forest is *hereditary* under subspaces. That is, if X is a forest and $S \subset X$ is a subset given the subspace ordering, then S is a forest. There is an intuitive sense in which all forests are made up of trees - each 'connected' as a subspace - while 'disconnected' from each other. We call a subset $U \subset X$ *open* if for every $x \in U$ and $x \leq y$ in X, we have $y \in U$. We say that a collection $\{S_i \subset S\}_{i \in I}$ of subsets of S *cover* S if we have $\bigcup_{i \in I} S_i = S$. We say that the collection is an *open cover* of S if each S_i is open in S. We say that the sets in a collection $\{S_i \subset S\}_{i \in I}$ are *pairwise disjoint* if for any S_i and S_j such that $S_i \neq S_j$, we have $S_i \cap S_j = \emptyset$.

Definition 2.5.4. A subset $S \subset X$ of a finite partial order given the subspace ordering is ***connected*** if there is no open cover of S where the open sets in the cover are pairwise disjoint.

Definition 2.5.5. The ***connected components*** of a partial order X are the connected subspaces $S \subset X$ which are maximal with respect to subset inclusion.

For any partial order X, the connected components are closed (the complement of an open set), and are always disjoint. When X is finite, they are also open. When X is a finite partial order, a subspace S is connected iff it is possible to 'zig-zag' up or down between any two elements $s, s' \in S$, where we form a sequence of elements $z_i \in S$, where $s \leq z_1$ or $s \geq z_1$, and similarly $z_i \leq z_{i+1}$ or $z_i \geq z_{i+1}$ for each step in the sequence, such that eventually we get to s'.[8]

Claim 2.5.1. If P is a finite forest, the connected components of P are the open subspaces $T_i \subset P$, each a tree, such that any pair $T_i \cap T_j = \emptyset$ is disjoint if $i \neq j$, and $\bigcup_{i \in I} T_i = P$.

We write a union of disjoint sets as sums, so the above proposition can be stated that every finite forest P factors uniquely as the disjoint union $P = T_1 + \ldots + T_n$, up to rearrangement of the summands. Applying this statement to trees, any open subset $U \subset T$ of a tree is a forest, and hence factors into constituents of T, with $U = K_1 + \ldots K_n$. In this way, the open sets of a tree T can be thought of families of disjoint constituents of T. A *constituent* of a tree T will then be an embedding of a connected open subset of T.

The mapping which takes in a finite partial order X and returns the set of connected components of X is actually a functor $\kappa : \mathbf{FPos} \to \mathbf{FSet}$, where $\mathbf{FSet}$ is

[8] For this proof, and a description of the relationship between topological connectivity and paths, see May [28].

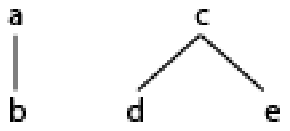


Figure 2.4: A disconnected partial ordering on the set $\{a, b, c, d, e\}$. It has two connected components: the subspaces corresponding to $\{a, b\}$ and $\{c, d, e\}$. X is a forest, and each connected component of X is a tree.

the category of finite sets and set-functions. To see this, note that for any order-preserving function $f : X \to Y$ between partially ordered sets, if a and b in X are any two elements in the same connected component, then $f(a)$ and $f(b)$ are in the same connected component. Denoting the set of connected components of a space X by $\kappa(X)$, the behavior of κ on an order-preserving function $\kappa(f) : \kappa(X) \to \kappa(Y)$ takes any component $K \subset X$ to the component of Y containing $f(a)$, where a is any element of K. This is well-defined since any two elements of K are taken to the same component of Y. We will later see how this functor is induced as an *adjoint* arising from a string a functors originating in the forgetful functor from **FPos** to **FSet** in §3.4.

2.5.2 C-command

We denote the set of open sets of a finite partial order as $\mathcal{O}(X)$. Every $U \in \mathcal{O}(X)$ corresponds to a family of constituents κU. The basic motivating fact is that there is a naturally arising unary operation on $\mathcal{O}(X)$ which takes in a constituent and returns the open set corresponding to the c-commanding constituents when X is a tree.

Definition 2.5.6. Given a finite partial order X and open subset $U, V \in \mathcal{O}(X)$, we define the ***relative pseudo-complement*** operator $U \Rightarrow V$ to be the largest open subset W such that $W \cap U \subset V$. We define the ***pseudo-complement*** of U to be $U \Rightarrow \emptyset$. This is the largest open subset V such that $U \cap V = \emptyset$. We denote this pseudo-complement as $\neg U$.

The relative pseudo-complement operation arises naturally as part of an induced Heyting algebra structure on $\mathcal{O}(X)$. It is related to the *modus ponens* law, in that $U \cap (U \Rightarrow V) \subset V$ by definition, which is formally similar to the relation $A \wedge (A \Rightarrow B) \vdash B$ from logic. Note that when $\mathcal{O}(X) = \mathcal{P}(X)$, then the pseudo-complement is just the usual complement. We show how this operation arises naturally on $\mathcal{O}(X)$ in §3.4. We now recall the traditional definition of c-command.

Definition 2.5.7. Let T be a a tree. We say that x ***c-commands*** y in T if for every node z properly dominating x in T, z dominates y, and neither x or y dominate the other. *Proper domination* means that $z \leq x$ and $z \neq x$. We say that a constituent K c-commands a constituent C if the root node of K c-commands the root node of C.

Claim 2.5.2. If X is a tree and $K \subset X$ is a constituent - that is, a connected open subset of X - then $\kappa(\neg K)$ is exactly the set of constituents c-commanding K.[9]

[9] The claim can also be interpreted as saying that the set of points in a constituent K and the set of points G in the collection of constituents c-commanding K are maximally disjoint amongst the dominance-closed sets of X. That is, $\neg K = G$ and $\neg G = K$. The binary c-command relation can be recovered from 'uncurrying' this operator: for a tree X, denote the set of constituents of X by const_X. We have a negation operator $\mathrm{const}_X \to \mathcal{O}(X)$ taking K to $\neg K$. We have a function $\mathcal{O}(\kappa) : \mathcal{O}(X) \to \mathcal{P}(\mathrm{const}_X)$ mapping each open set U to the set of components $\kappa(U)$,

Proof. Let $K \subset X$ be a constituent, and let V be a connected component of $\neg K$. Then V is itself a constituent since it is open and connected; call its root v. Let $x \in X$ be any element properly dominating v, i.e. $x \leq v$ and $x \neq v$. Suppose that $x \in \neg K$. Denote by U_x its *upset* $\{y \in X \text{ such that } x \leq y\}$. Since $\neg K$ is open, we have $U_x \subset \neg K$. But U_x is connected, contradicting the maximality of V amongst the connected subspaces of $\neg K$. Hence, if $x < v$, then $x \notin \neg K$. In other words, $U_x \cap K \neq \emptyset$. For any two constituents in a tree, either one is contained in the other, or they are disjoint. If $U_x \subset K$, then $V \subset K$, contradicting the fact that $\neg K \cap K = \emptyset$. Hence, $K \subset U_x$, and x dominates all nodes in K. □

As is the general theme in category theory, we would like to express this result in terms of morphisms. We now illustrate that in this format there is a close connection between constituency and c-command. We say that an order-preserving function between partial orders $f : X \to Y$ is an ***open map*** if for every open subset $U \subset X$, the image $f(U) = \{y \in Y \text{ such that } \exists x \in U, f(x) = y\}$ is open in Y. An order-preserving function f between trees is open if and only if f preserves constituents, i.e. for every constituent $K \subset X$, the image $f(K)$ is a constituent of

which is a set of constituents. The powerset $\mathcal{P}(\text{const}_X)$ is in a canonical bijection with the set of characteristic functions $\{\text{true}, \text{false}\}^{\text{const}_X}$. Composing the two functions, we have a map $\text{const}_X \to \{\text{true}, \text{false}\}^{\text{const}_X}$ taking a constituent K to the function sending a constituent C to true if and only if C c-commands K. We can uncurry this to a function $\text{const}_X \times \text{const}_X \to \{\text{true}, \text{false}\}$ sending (C, K) to true if and only if C c-commands K. This can be viewed as a characteristic function of a subset $R \subset \text{const}_X \times \text{const}_X$, the c-command relation on the set of constituents of X.

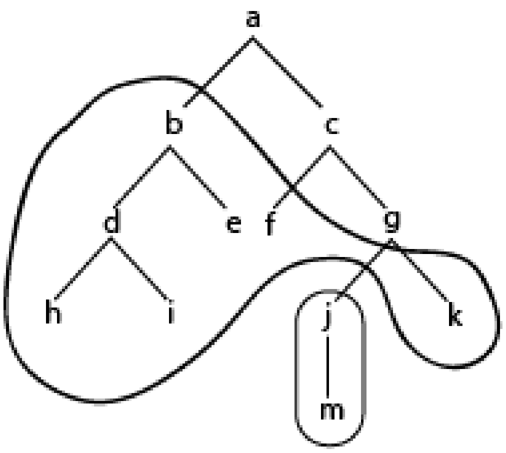


Figure 2.5: The open subset $K = \{j, m\}$ is a constituent. Its negation $\neg K$ is the largest open subset disjoint from K, and is circled. Being an open subset of a tree (i.e. being a forest), implies that this space decomposes uniquely into connected components (each a constituent), the corresponding to $\{b, d, e, h, i\}$, $\{f\}$ and $\{k\}$. These three constituents consisting of these elements are exactly the constituents c-commanding K.

Y.

Claim 2.5.3. A function $f : X \to Y$ is order-preserving if and only if for every $V \in \mathcal{O}(Y)$, the inverse image $f^{-1}(V) = \{x \in X \text{ such that } f(x) \in V\}$ is an element of $\mathcal{O}(X)$. An order-preserving function f is an open map if and only if it preserves relative pseudo-complements, in that $f^{-1}(U \Rightarrow V) = f^{-1}(U) \Rightarrow f^{-1}(V)$.

Proof. This is a specialization of a result from locale theory, see, e.g., Johnstone [29] and Mac Lane & Moerdijk [30]. □

Given a constituent-preserving map $f : X \to Y$ between trees, we then know that for any constituent $K \subset Y$, we have $f^{-1}(\neg K) = \neg f^{-1}(K)$. It is then not surprising that c-command relations which hold in Y are pulled back to c-command relations in X.

Claim 2.5.4. Let $f : X \to Y$ be an open map between finite trees. If $K \subset Y$ is a constituent and V c-commands K in Y, then the connected components of $f^{-1}(V)$ are exactly the constituents of X which map into V which c-command at least one of the connected components of $f^{-1}(K)$.

Proof. Suppose $f : X \to Y$ is an open map and $V, K \subset Y$ are two constituents such that V c-commands K. We write $x < U$ for an open subset U if $x < u$ for all $u \in U$.

$\Rightarrow$) Choose any connected component L of $f^{-1}V$. We show it c-commands some component of $f^{-1}K$. Let x be the greatest element such that $x < L$. Since f is order preserving, $f(x) \leq f(l)$ for all $l \in L$, and $f(l) \in V$. V has a root r and since the points above $f(l)$ form a linear order, r and $f(x)$ must be linearly ordered. If $r \leq f(x)$, then $f(x) \in V$; but in this case, $x < l$ both map into V, and hence the upset of x would be in the preimage of V, contradicting the maximality of L as a connected component of $f^{-1}V$. So $f(x) < r$, hence $f(x) < V$. Since V c-commands K by hypothesis, $f(x) < K$. Let Z be the upset of x in X. By the assumption that f is open, $f(Z)$ is the upset of $f(x)$, which contains K. Now, choose any point $m \in K$. By the assumption of openness, there must be some point $z \in Z$ such that $f(z) = m$. This z belongs to some connected component of $f^{-1}K$, and suppose this component is dominated by p (note that p cannot be smaller than x by monotonicity). Then $x < p < z$, so in particular $x < K_p$, where K_p is the upset of p in X. Then, for every node dominating L, this node will dominate K_p. In particular, we have produced a component K_p of $f^{-1}K$ which L c-commands.

$\Leftarrow$) Choose any $L \subset X$ such that $f(L) \subset V$ and L c-commands some compo-

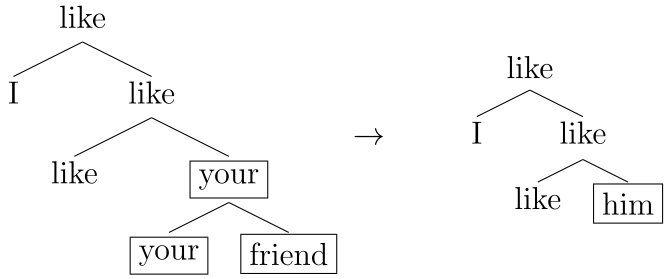


Figure 2.6: A constituent-preserving map loosely preserves c-command up to images in Y. For example, since I c-commands *him* in Y, the preimage I c-commands the preimage *your friend* in X.

nent of $f^{-1}K$. We show that L is a component of $f^{-1}V$. Clearly, $L \subset f^{-1}V$, and L is connected, so it must be contained in some connected component L' of $f^{-1}V$. Suppose that $L \subset L'$ is proper, i.e. $L \neq L'$, and suppose the roots are l and l', respectively. Since L c-commands some component, call it C, l' must dominate C, i.e. $l' < C$. But then $\emptyset \neq f(L' \cap C = C) \subset V \cap K$, contradicting the assumption that V c-commands K. So $L = L'$, and L is a connected component of $f^{-1}V$. □

2.6 Summary

The purpose of this chapter was primarily to introduce standard category-theoretic methods to analyze DSOs. By looking at representable constructs, we showed how we can recover notions of isomorphisms of DSOs and subobjects of DSOs in a general setting which is amenable to many formalizations of DSOs. We then gave an original result relating constituent structure and c-command for order-theoretic DSOs: morphisms $f : X \to Y$ between trees which preserve constituency on the nose pull c-command relations in Y back to c-command relations in X.

Chapter 3: Structural Changes, Grammatical Relations, and Derivations

3.1 Overview

We introduce structural changes, which are the main novel contribution of the research presented in this thesis. By 'adding structural change (SC) data' to a grammar, we mean that instead of simply returning a derived syntactic object Z given a tuple of input DSOs $(A_1, \dots, A_n)$, we will give explicit information relating the structure of each A_i to the structure of Z in terms of morphisms. One of the main reasons to do this is so that we can handle all structural changes in a uniform way, and not as *sui generis* operations. We then demonstrate that many grammatical properties become easy to state. In particular, we can describe projection, selection, agreement, and many other dependencies very easily, even when the DSOs have rich structure such as feature geometry. Isomorphisms of DSOs will scale up to isomorphisms between SCs. We can then extract information about how connected two DSOs become *over* a particular SC, such that we can recover grammatical relations from a typology of these connectivity properties. We then introduce a naïve model of derivations as families of DSOs connected by SCs. We then describe grammars, isomorphisms of derivations and subderivations, and isomorphisms and equivalences of languages using this richer structure. We finally revise our model of derivations to a more restrictive one which is a well-behaved representable construct.

3.2 Structural Changes and Grammatical Relations

Structural changes (SCs) can be represented as tuples of functions (defined on the sets of nodes) from a tuple of DSOs into a new DSO. In Fig. 1.5, we have a pair of DSOs (‘her parents’, ‘pet the furry dog’) mapping into the DSO ‘her parents pet the furry dog’. However, it is not simply the case that we have assigned this output DSO to this pair: the pair of functions f and g between sets of nodes explicitly map nodes of the input trees to nodes of the output tree, e.g. the node for ‘dog’ in the input DSO is mapped to a corresponding node for ‘dog’ in the output DSO by g. Moreover, f and g ‘preserve the structure’ of the input DSOs; that is, they are morphisms.

Fix a category $\mathbf{A}$ of DSOs. We tentatively formalize a SC on an n-tuple of $\mathbf{A}$ objects $(A_1, \dots, A_n)$ as an output DSO Z together with an n-tuple of $\mathbf{A}$-morphisms $f_i : A_i \to Z$. Isomorphisms of DSOs naturally induce isomorphisms of SCs on a fixed tuple: given two SCs on an n-tuple $(f_i : A_i \to Z : 1 \leq i \leq n)$ and $(g_i : A_i \to Y : 1 \leq i \leq n)$, we say that they are *isomorphic* if there is an $\mathbf{A}$-isomorphism $k : Z \to Y$ such that for each f_i, we have $k \circ f_i = g_i$. We tentatively formalize an n-ary *rule* G as assignment which takes in an n-tuple of $\mathbf{A}$-objects $(A_1, \dots, A_n)$ and returns a set $G(A_1, \dots, A_n)$ of SCs $(f_i : A_i \to Z : 1 \leq i \leq n)$, such that no two elements of the set are isomorphic. We do not require that G is defined for all n-tuples, and we allow it to return a set such that the result does not have to be deterministic. We now look at some examples which can be used in a linguistic context. For simplicity, we will first just consider $\mathbf{A} = \mathbf{FPos}$. We will consider very unrestricted rules for now so that we do not have to keep track of as many details about the DSOs involved.

Phrasal attachment. We can construct a binary phrasal attachment rule G. Let $G(A, B)$ be defined whenever A and B both have a least element (root). We are going to define this rule such that it attaches the left operand to the right one. We define $G(A, B)$ to be a singleton. The output DSO Z consists of the disjoint union of points of A and B, together with all of the order relations in A or B. Additionally, we add a relation $r \leq a$ from the root r of B to each element $a \in A$. We then define the SC to be the two order-preserving inclusions $f : A \to Z$ and $g : B \to Z$. An example is given in Fig. 3.1.

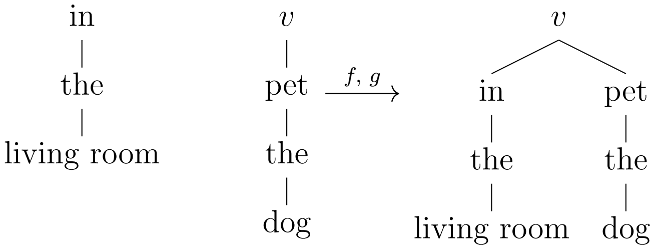


Figure 3.1: A pair of order-preserving functions attaching the root of the first operand to the root of the other.

Specifier/agreeing adjunct attachment. We construct a rule which attaches two phrases, but also attaches two ‘features’ within those phrases. Let A and B be any two orders with least elements r_A and r_B, along with any other selection of elements $k_A \in A$ and $k_B \in B$ such that $k_A \neq r_A$ and $k_B \neq r_B$. We intend to again attach A to B, but we will also introduce a dependency $k_A \leq k_B$, indicating a dependence of the element k_A on k_B. For example, A could be an adjunct and k_A and k_B gender features, such that the gender feature of the adjunct becomes dependent on the gender feature of the head of the phrase B it attaches to. Or, k_A and k_B could be case features or category and EPP features. In this unrestricted version of the rule, we will let $G(A, B)$ consist of many SCs, one for each pair (k_A, k_B) of non-root elements of A and B. For each such pair, we will have a DSO Z_{k_A,k_B} which again consists of the disjoint union of elements of A and B. It will have all the order relations from both A and B, but additionally have relations of the form $r_B \leq a$ for all $a \in A$, and relations $k_A \leq b$ for all $b \in B$ such that $k_B \leq b$. Associated to this DSO will be the order-preserving inclusions $f : A \to Z_{k_A,k_B}$ and $g : B \to Z_{k_A,k_B}$. An example is given in Fig. 3.2.

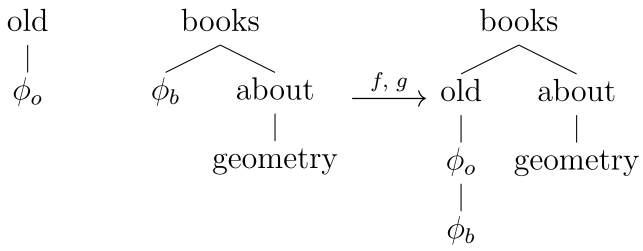


Figure 3.2: A pair of order-preserving functions attaching the root of the first operand to the root of the other, while also attaching the gender feature of the head noun to the gender feature of the adjunct.

Selection. Consider objects A and B just as in the previous example. We will model *selection* distinctly from agreement/licensing by identifying k_A and k_B instead of creating a dependency between them. For each such (k_A, k_B) pair, we construct Z_{k_A,k_B} which is like the disjoint union of A and B, except k_A and k_B are identified. It has the minimum order relations necessary such that the inclusions $f : A \to Z_{k_A,k_B}$ and $g : B \to Z_{k_A,k_B}$ are order-preserving. That is, it is the transitive closure of the order relations $f(a) \le f(a')$ and $g(b) \le g(b')$ whenever $a \le a'$ in A or $b \le b'$ in B. An example is given in Fig. 3.3.

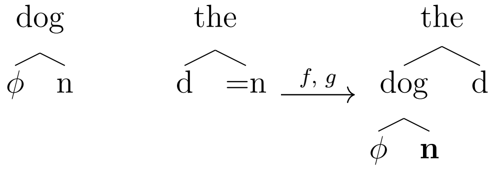


Figure 3.3: A pair of order-preserving functions attaching the root of the first operand to the root of the other, while also identifying a selector feature and category feature.

More structure. We also give one example with more structure to illustrate how if DSOs are more richly structured, the SCs will be as well. Let $\mathbf{A_L}$ be the category of sets X together with a dominance preorder $\le_X$, precedence preorder $\preceq_X$, with one unary predicate λ_X on X for each $\lambda \in \mathbf{L}$. The morphisms $f : X \to Y$ of $\mathbf{A_L}$ are again functions between underlying sets which preserve $\le$, $\preceq$, such that if $\lambda_X(x) = \text{true}$, then $\lambda_Y(f(x)) = \text{true}$. We define a specifier-merge rule which will attach one phrase to another, and linearize the attacher before the attachee in terms of precedence.[1] Let A and B be any objects which have a root with respect to $\le$, such that elements are totally ordered with respect to $\preceq$ in both. For any such pair of objects, we define $G(A, B)$ to be a singleton. The output DSO Z will consist of the disjoint union of elements of A and B, with $\lambda_Z(x) = \text{true}$ if and only if $\lambda_A(x) = \text{true}$ or $\lambda_B(x) = \text{true}$. It will have all the dominance relations in A and B, but additionally relations $r \le a$ from the root r of B to each element $a \in A$. It will

[1] This will differ from the way we describe unselected specifiers in §3.2.1 which uses features.

have all the precedence relations of A and B, but additionally precedence relations $a \preceq b$ for each $a \in A$ and $b \in B$. The SC maps into Z are given by the inclusions $f : A \to Z$ and $g : B \to Z$. An example is given in Fig. 3.4.

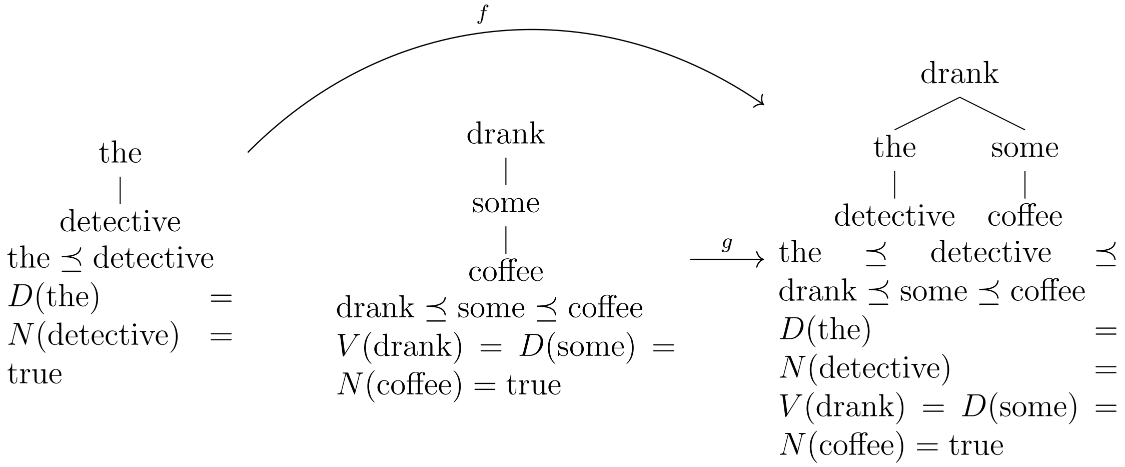

f
the
detective
the $\preceq$ detective
D(the) = N(detective) = true
drank
some
coffee
drank $\preceq$ some $\preceq$ coffee
V(drank) = D(some) = N(coffee) = true
g
drank
the
some
detective
coffee
the $\preceq$ detective $\preceq$ drank $\preceq$ some $\preceq$ coffee
D(the) = N(detective) = V(drank) = D(some) = N(coffee) = true

Figure 3.4: Specifier-merge with precedence and syntactic type data. Assume that all $\lambda_X(x) =$ false unless indicated otherwise.

Head-adjunction. We can also give a (non-copying) SC associated to head adjunction. Given items X and Y with roots x and y, instead of introducing a relation $x < y$ or $y < x$, we can use the other option of the trichotomy, $x = y$. This will have the automatic effect of unioning the features of the heads. An example is given in Fig. 3.5. This is similar to the Distributed Morphology operation of Fusion, while keeping track of the feature structure of each.[2] This is depicted in Fig. 3.5. However, such an operation might be 'too symmetric' for use in linguistics. For example, the technology for extending minimal domains outlined in Chomsky [6] and reviewed in Hornstein [32] allows head-adjunction to extend a domain related to movement. However, in the case of successive head-adjunction, the extension only applies as far

[2] "Fusion combines two sister nodes into a single X^0, with features of both input nodes" Bobaljik [31], p. 14

as the last head adjoined. That is, "Importantly, each successive adjunction doesn't extend the previous chain." Hornstein [32], p. 156. The problem with the model of head adjunction just proposed can easily be made asymmetric. If f and g are the category features of the heads, we simply introduce an asymmetry $f \leq g$ between the category features. Then, we get the correct result that feature structures are automatically unioned, the labels are 'flattened', such that they become inseparable, yet we have an asymmetry between the categories involved.

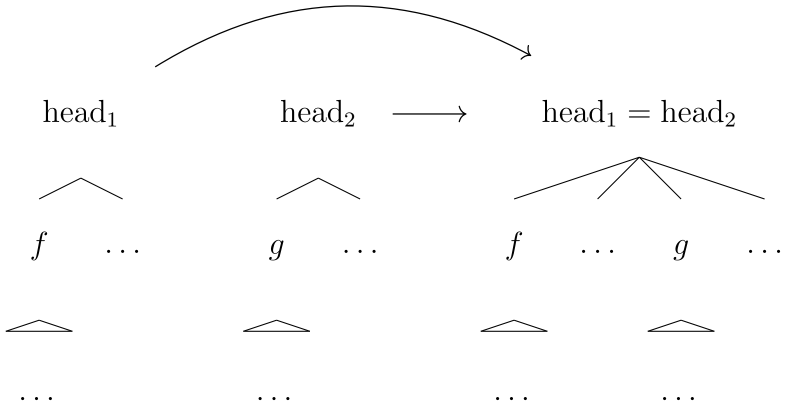


Figure 3.5: The SC induced by identifying the labels of two heads.

3.2.1 Grammatical Relations

Many syntactic properties can be recovered by analyzing the images of nodes under SC morphisms. Most primitive among them is *projection*. For example, in Fig. 3.4, we can tell that it is the right operand which projects, as it is the root of that DSO which maps to the root of the output DSO. We do not have to make any assumptions about whether the rule G projects the left or right operand - this can be recovered just by looking at the SC itself. More general syntactic relations can be recovered in a similar manner by tracking the images of nodes and what dependencies are introduced.

Each rule above can be seen as a sort of *attaching* or *gluing* of two structures. This will be made precise in Chapter 4. We will now use a category theoretic generalization of intersections to measure *how much* DSOs were glued together over a SC. Consider the 'classical' case of a tree T with constituents K and C, viewed as subsets of T. Intersecting K and C gives one of three results: K, C, or $\emptyset$. K results exactly when $K \subset C$ is a subconstituent, and conversely C when $C \subset K$. $\emptyset$ results when there is no dependency between the two phrases. So, intersecting two phrases tells us about whether there is a dependency between them or not. We generalize this analysis to when K and C are related to T by arbitrary morphisms, such that we can measure dependencies introduced over SCs.

We first generalize intersections to *pullbacks.*

Definition 3.2.1. In any category $\mathcal{C}$, given a pair of morphisms $f : A \to C$ and $g : B \to C$, a *pullback* of f and g, if it exists, is an object $A \times_C B$ together with morphisms $\pi_A : A \times_C B \to A$ and $\pi_B : A \times_C B \to B$ such that (1) $f \circ \pi_A = g \circ \pi_B$; and (2) if $\pi'_A : Z \to A$ and $\pi'_B : Z \to B$ are any morphisms such that $f \circ \pi'_A = g \circ \pi'_B$, then there is a *unique* morphism $u : Z \to A \times_C B$ such that $\pi'_A = \pi_A \circ u$ and $\pi'_B = \pi_B \circ u$. See Fig. 3.6.

based on Borceux [23] 2.5.1

The second condition is often called a *universal* requirement, in that all solutions to the problem "find morphisms $k_A : Z \to A$ and $k_B : Z \to B$ such that

$$\begin{tikzcd}
Z \arrow[drr, bend left, "\pi'_B"] \arrow[dr, dotted, "u"] \arrow[ddr, bend right, "\pi'_A"'] & & \\
& A \times_C B \arrow[r, dashed, "\pi_B"] \arrow[d, dashed, "\pi_A"'] & B \arrow[d, "g"] \\
& A \arrow[r, "f"'] & C
\end{tikzcd}$$

Figure 3.6: A pullback diagram. $(A \times_C B, \pi_A, \pi_B)$ is a pullback of f and g.

$f \circ k_A = g \circ k_B$" factor through a pullback. While the solution to a universal problem may not be unique - that is, there may be multiple pullbacks $(\pi_A : A \times_C B \to A, \pi_B : A \times_C B \to B)$ and $(\pi'_A : (A \times_C B)' \to A, \pi'_B : (A \times_C B)' \to B)$, the universality requirement guarantees that there is a *unique* isomorphism $u : A \times_C B \to (A \times_C B)'$ between any two pullbacks such that $\pi'_A \circ u = \pi_A$ and $\pi'_B \circ u = \pi_B$.

We show how to compute the pullback of two morphisms $f : A \to C$ and $g : B \to C$ in various categories. The pullback of functions f and g in **Set** can be computed as $A \times_C B = \{(a, b) \in A \times B \mid f(a) = g(b)\}$ together with the coordinate projections. Note that when f and g are subset inclusions, the intersection $A \cap B$ together with the inclusions $A \cap B \hookrightarrow A$ and $A \cap B \hookrightarrow B$ is a pullback of f and g. In **Proset**, the pullback of two order-preserving functions can be computed on the underlying set, then giving $A \times_C B$ the order relations $(a, b) \leq (a', b')$ if and only if $a \leq a'$ and $b \leq b'$. Pullbacks in **Pos** and **FPos** are computed identically. If $\mathbf{A}$ is a category of partial orders with predicates on it, then for a predicate λ, $\lambda(a, b)$ will be true in the pullback if and only if both $\lambda(a)$ and $\lambda(b)$ are true.

We will be interested in the case when C is an output DSO, and each of A

and B is either a head, with f or g the associated SC, or a constituent subset of C, with f or g the associated subset inclusion function. Consider the derivation of 'the dog' given in Fig. 1.7, with A the DSO corresponding to the lexical item 'the', B the DSO corresponding to the lexical item 'dog', and C the DSO corresponding to the derived object 'the dog', with f and g the SCs. The pullback of f and g is the singleton $\{(=n, n)\}$, indicating that it is this pair of features which becomes identified in C.

Now consider the derivation of 'big old tree' given in Fig. 1.6. In particular, let C be the DSO corresponding to 'old tree', A the lexical item 'old', B the lexical item 'tree', and f and g the respective SC functions corresponding to the first application of adjunction in the derivation. The pullback of f and g is empty, indicating that no features of the two lexical items have identified under this adjunction operation. However, setting A and B equal to the subsets of C corresponding to the constituents dominated by the images of 'old' and 'tree' (that is, $A = \{old, \phi_o, \phi_t\}$ and $B = C$), the pullback (intersection) of the two inclusions is A, indicating that 'tree' dominates 'old' in C. That is, by the time the derivation produces C there is a dependency between the two lexical items. This indicates that at least 'phrasal-attachment' has occurred. Furthermore, let A be the subset $\{\text{old}, \phi_o, \phi_t\} \subset C$, corresponding to the constituent dominated by the image of 'old' with f the substructure embedding, and B the lexical item 'tree' together with g, the SC mapping it into C. The pullback of these two functions is essentially the singleton $\{\phi_t\}$, indicating that this feature of the head 'tree' is a dependent of 'old' by the stage C. The nonemptiness of this pullback indicates that agreement or licensing has occurred, in that the phrase

$$\begin{tikzcd}
\mathbf{min}(x) \times_C \mathbf{min}(y) \arrow[r, dotted] \arrow[d, dotted] & \mathbf{max}(x) \times_C \mathbf{min}(y) \arrow[r, dotted] \arrow[d, dotted] & \mathbf{min}(y) \arrow[d] \\
\mathbf{min}(x) \times_C \mathbf{max}(y) \arrow[r, dotted] \arrow[d, dotted] & \mathbf{max}(x) \times_C \mathbf{max}(y) \arrow[r, dotted] \arrow[d, dotted] & \mathbf{max}(y) \arrow[d] \\
\mathbf{min}(x) \arrow[r] & \mathbf{max}(x) \arrow[r] & C
\end{tikzcd}$$

Figure 3.7: The 2-by-2 pullback comparison of the head-level and phrasal-level constituents associated to points x and y at a DSO C containing them, considered as pullback diagrams in **FPos**.

projected by *old* depends on some feature originating from the head *tree* whose projected phrase it is attached to.

These comparisons can be given a general treatment. Let $\mathbf{min}(x)$ and $\mathbf{min}(y)$ be two lexical items which both map into a common stage C under SCs f and g. Denote by $\mathbf{max}(x)$ the subset of C of elements dominated by some element in the image of f, and similarly denote by $\mathbf{max}(y)$ the subset of elements dominated by some element in the image of g. Note that $f : \mathbf{min}(x) \to C$ must factor through $\mathbf{max}(x) \hookrightarrow C$, and similarly for g. We can collect up all cross-comparisons of 'overlap' of minimal and maximal projections of the two lexical items at C, given in Fig. 3.7.

If any one of the pullbacks in Fig. 3.7 is nonempty, then all sets it maps into must be nonempty. In particular, if $\overline{x}$ and $\overline{y}$ are the images of the roots of the lexical items in C, and we have an immediate domination relation $\overline{y} \leq \overline{x}$,[3] then at least $\mathbf{min}(x) \times_C \mathbf{max}(y)$ is nonempty (and hence so is $\mathbf{max}(x) \times_C \mathbf{max}(y)$). We think of the immediate domination relation $\overline{y} \leq \overline{x}$ as indicating that xP is in the

[3] We say that a *immediately dominates* b if $a \leq b$, $a \neq b$, and if $a \leq z \leq b$, then $z = a$ or $z = b$.

minimal domain of yP [6]. There are then only three 'degrees of connectivity' that x and y can have according to this metric - either (1) all of the other pullbacks are empty, in which case we can think of the xP as a *pure adjunct* of the yP, undergoing no other feature connectivity; (2) $\mathbf{min}(x) \times_C \mathbf{min}(y)$ is nonempty (and hence so is $\mathbf{max}(x) \times_C \mathbf{min}(y)$), in which case two features of the heads have identified, which we may think of as indicating *selection*; or (3) $\mathbf{max}(x) \times_C \mathbf{min}(y)$ is nonemtpy while $\mathbf{min}(x) \times_C \mathbf{min}(y)$ is empty, indicating that the xP is dependent on some feature of the head y, but has not identified with it, which we may think of as indicating an (unselected) *specifier* which undergoes licensing or agreement.

It is also a nice consequence that using these definitions, we have a natural ordering of grammatical relations by connectivity: selected arguments are the most connected, unselected licensed or agreeing phrases less so, and pure adjuncts the least connected. To state this formally, we first note that we are, at this level of granularity, only interested in whether each pullback is nonempty. Recall that if any one of the pullbacks is nonempty, then every pullback it maps into must also be nonempty, since a function out of a nonempty set must take values in a nonempty set. We can then view the 4 pullbacks as being elements in a partial order, ordered by implication of nonemptiness. We are then interested in 6 possibilities, corresponding to the upsets of this partial order: (1) all pullbacks are empty; (2) all but $\mathbf{max}(x) \times_C \mathbf{max}(y)$ are empty; (3) only $\mathbf{min}(x) \times_C \mathbf{min}(y)$ and $\mathbf{max}(x) \times_C \mathbf{min}(y)$ are empty; (4) only $\mathbf{min}(x) \times_C \mathbf{min}(y)$ and $\mathbf{min}(x) \times_C \mathbf{max}(y)$ are empty; (5) only $\mathbf{min}(x) \times_C \mathbf{min}(y)$ is empty; or (6) all are nonempty. These are naturally ordered by subset inclusion, indicating increasing degree of connectivity. Further

$$\begin{tikzcd}
 & \text{Selected Argument} & \\
 & \text{Unselected Spec} \arrow[u] & \\
\text{Adjunct} \arrow[ur] & & -\text{Adjunct} \arrow[ul] \\
 & -\text{Spec} \arrow[ul] \arrow[ur] & \\
 & \text{Disconnected} \arrow[u] &
\end{tikzcd}$$

Figure 3.8: The 6 upsets of the 'lattice of impliciation of nonemptiness of the 2-by-2 pullback diagram of the head and phrasal projections of two points'. We give each element of the lattice a name corresponding to its meaning in the case of one being in the minimal domain of the other.

restricting attention to when $\mathbf{min}(x)$ and $\mathbf{min}(y)$ have roots x and y such that these elements map to elements $\overline{x}$ and $\overline{y}$ in C where an immediate domination relation $\overline{y} \leq \overline{x}$ holds, we will only be in half of the situations since $\mathbf{min}(x) \times_C \mathbf{max}(y)$, and hence $\mathbf{max}(x) \times_C \mathbf{max}(y)$, will both be nonempty. We name the elements of this partial order accordingly in Fig. 3.8. In the case where $\overline{y} \leq \overline{x}$ is an immediate domination relation in C, we say that x is an *Adjunct* of y if only $\mathbf{min}(x) \times_C \mathbf{max}(y)$ and $\mathbf{max}(x) \times_C \mathbf{max}(y)$ are nonempty. Similarly, we say that x is an *Unselected Spec* of y if only $\mathbf{min}(x) \times_C \mathbf{min}(y)$ is empty when $\overline{y} \leq \overline{x}$ is an immediate domination relation. Finally, we say x is a *Selected Argument* of y if none of the pullbacks are nonempty when $\overline{y} \leq \overline{x}$ is an immediate domination relation.

It is natural to then ask what the 'extraneous' connectivity relations mean. These can notably only occur when it is not the case that the xP is contained in yP, i.e. when $\overline{y} \not\leq \overline{x}$. We will more generally say that x is R-related to y when R is one of the elements in Fig. 3.8 and the relevant pullbacks are empty, regardless of whether there is a dominance relationship between $\overline{x}$ and $\overline{y}$. For example, if x is an Adjunct of y, it is Adjunct related to it, and so on. *Disconnected* has the obvious meaning: the intersection of the two phrases in C is empty - they have no elements in common. *-Adjunct* is the adjunction relation in reverse. x is *-Adjunct related* to y if some element of the head $\mathbf{min}(y)$ enters the phrase $\mathbf{max}(x)$ but does not identify with it, and no element of the head $\mathbf{min}(x)$ enters the phrase $\mathbf{max}(y)$. In particular, if y is an Adjunct of x, then x is -Adjunct related to y. The only 'new' relation then is -Spec. Such a relation can only occur when there is some element in the intersection of $\mathbf{max}(x)$ and $\mathbf{max}(y)$, but this element is *not* the image of a feature from either head. This relation is also symmetric, in that x is -Spec related to y if and only if y is -Spec related to x. Such a configuration could arise if both the xP and the yP are licensed by a common feature originating outside of their projecting heads. One such case might be when there is a *wh* feature on a complementizer C which licenses multiple *wh*-phrases in the case of multiple *wh*-movement. An example is given in Fig. 3.9. It can also arise in other feature-sharing contexts, such as multiple adjuncts engaging in concord with a gender feature from a head noun. The idea is that these phrases are not totally disconnected, but neither depends directly on the other. This relation gets ordered as weaker than an adjunction relation.

More subtle grammatical relations can be defined based on similar notions by requiring specific kinds of features to be in the pullbacks, such as when we are working in category when features are typed, or by describing more specific configurations which must hold between elements of heads and their images. Summarizing,

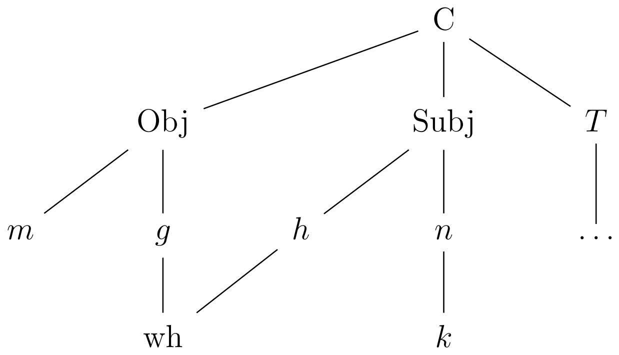


Figure 3.9: Moving multiple NPs - $\{\mathrm{Obj}, m, g\}$ and $\{\mathrm{Subj}, n, k\}$ - to the same wh feature. This leads to a -Spec relation between the Obj and Subj, since their phrases overlap on a wh element, but this element did not arise from the heads projecting either phrase.

the basic relations, using the SC data, can be recovered as a typology of connectivity properties of labels and features and are naturally ordered as above.

3.3 Derivations

The previous section motivated using tuples of morphisms to model SCs induced by a grammatical rule. Ehrig, et al. [5] describes a *direct derivation* as a special kind of morphism $f : A \to B$ between graphs with extra structure. A general *derivation* is a composite of such maps, which can be thought of as the *net structural change*. However, we want to allow operations which take in tuples of objects and give structural change morphisms from each input to the output object. Additionally, we want to study the structure of the whole sequence of structural changes parameterized over a sequence of steps, and not just the net structural change. We will describe a general naïve model of derivations in this section, which we will revise for technical reasons in §3.5. To use the SCs as we have introduced them, we should think of a derivation as a partial order, where each node p in the partial order corresponds to a DSO A_p, and each order relation $q \leq p$ corresponds to a morphism $f_{q,p} : A_p \to A_q$ arising from a SC. In this sense, a derivation of $\mathbf{A}$-objects is a diagram of objects from $\mathbf{A}$ linked together by morphisms of $\mathbf{A}$. We could formalize a derivation using the following definitions.[4]

Note 1. Any partial order $(P, \leq)$ can be turned into a category whose objects are elements $p \in P$, such that there is exactly one morphism $p \to q$ if and only if $q \leq p$.

Definition 3.3.1. A ***diagram of shape*** P is a functor $F : P \to \mathbf{A}$. We designate $F(p) \equiv F_p$ and $F(p \leq q) = !_{p,q} : A_q \to A_p$. The functorial condition says that $!_{p,p} = 1_{F_p}$ is the identity on F_p and $!_{q,p} \circ !_{p,r} = !_{q,r}$.

Definition 3.3.2. Given a category $\mathbf{A}$, an $\mathbf{A}$***-derivation*** is a functor $F : P \to \mathbf{A}$, where P is a finite partial order.

Usually, we are interested in cases where P is a tree. If $p_1, \ldots, p_n$ are 'sisters' in P, with $x < p_i$ an immediate ordering relation for all p_i, then we can think of the maps $F_{p_i} \to F_x$ as the structural changes from the n-tuple of inputs to F_x.[5] Intuitively, a morphism of derivations $\mu : (P, F) \to (Q, G)$ is a map of underlying states $p \mapsto \overline{p}$ together with an $\mathbf{A}$-morphism $\mu_p : F_p \to G_{\overline{p}}$ for each state $p \in P$ which is 'compatible' with net structural changes.

[4] The definitions for functors and natural transformations were given in Defn. **??** and Defn. 2.3.9.

[5] We do not view the sisters as 'ordered', though we do view them as 'distinct' operands. That is, we think of this structural change as modeling an operation with n distinguished argument slots, but the slots are not linearly ordered with respect to each other.

Claim 3.3.1. Fix any category $\mathbf{A}$. Let $\mathscr{D}(\mathbf{A})$ be the collection of diagrams $F : P \to \mathbf{A}$ for any finite partial order P. We define a morphism $(m, \mu) : (P, F) \to (Q, G)$ to be an order-preserving function $m : P \to Q$ and natural transformation $\mu : F \to G{\circ}m$ of functors from P to $\mathbf{A}$. Composition of morphisms $(m, \mu) : (P, F) \to (Q, G)$ and $(n, \nu) : (Q, G) \to (R, H)$ is given by an order-preserving function nm, where $(nm)(p) = n(m(p))$, and natural transformation $\nu \star_m \mu : F \to H \circ (nm)$ with coordinates $\nu_{m(p)} \circ \mu_p : F_p \to G_{m(p)} \to H_{n(m(p))}$. These data give a category.

A morphism of $\mathbf{A}$-derivations is a correspondence between states, together with an $\mathbf{A}$-morphism for each DSO in correspondence. The naturality condition says that if $!_{q,p} : F_p \to F_q$ is a net structural change in F corresponding to $!_{mq,mp} : G_{mp} \to G_{mq}$ in G, then $\mu_q{\circ}!_{q,p} = !_{mq,mp} \circ \mu_p$. We describe the isos in any category of derivations.

Claim 3.3.2. If $(m, \mu) : (P, F) \to (Q, G)$ is an isomorphism, then (1) m is an isomorphism of partial orders, such that $\mu : F \to G \circ m$ is a natural isomorphism of functors. Hence, each $\mu_p : F_p \to G_{mp}$ is an $\mathbf{A}$-iso. This induces an isomorphism between each structural change in correspondence.

$\mathbf{A}$ is isomorphic to a subcategory of $\mathscr{D}(\mathbf{A})$, taking each DSO a in $\mathbf{A}$ to the derivation consisting of just one step - the object a itself. Intuitively, DSOs are simply single-state derivations, and we can include these single-state derivations into the category of all derivations. We construct this functor explicitly below.

Claim 3.3.3. The map $i : \mathbf{A} \to \mathscr{D}(\mathbf{A})$ sending $a \mapsto (*, !_a : * \to \mathbf{A})$, where $*$ is a one-point partial order, and $!_a$ is the assignment sending the single object of $*$ to a, and sending a morphism $f : a \to b$ to the derivation morphism $(1_*, !_f)$, where $!_f$ has the single coordinate $f : a \to b$, is a functor. i is isomorphic to a full subcategory embedding. We often denote $(*, !_a)$ as a.

We will now assume that $\mathbf{A}$ is a representable construct, such that we can describe *points* of a derivation. Let $\blacksquare$ be an object of $\mathbf{A}$ representing a faithful functor $U : \mathbf{A} \to \mathbf{Set}$. We can use the inclusion above to turn $\blacksquare$ into a derivation. Whenever $U : \mathbf{A} \to \mathbf{Set}$ is concrete and represented by $\blacksquare$, we call a map $x : \blacksquare \to (P, F)$ a *point*. A point of a derivation is given by a selection of state p, along with an $\mathbf{A}$-morphism $x_p : \blacksquare \to F_p$. In other words, it is simply a selection of state F_p together with an element in $U(F_p)$. $\blacksquare$ as a derivation represents a functor $\mathscr{D}(\mathbf{A})(\blacksquare, -) : \mathscr{D}(\mathbf{A}) \to \mathbf{Set}$. $\mathscr{D}(\mathbf{A})(\blacksquare, (P, F))$ is the set of points of (P, F), and consists of the disjoint union of all $U(F_p)$. Whenever $U : \mathbf{A} \to \mathbf{Set}$ is concretely representable, it makes sense to define a *projection relation* on the points of (P, F).

Definition 3.3.3. Let $U : \mathbf{A} \to \mathbf{Set}$ be a construct represented by $\blacksquare$. Given two points x and y of (P, F), living in DSOs F_p and F_q such that $q \leq p$, we call y a *projection* of x if $!_{q,p}(x) = y$.

While it is obvious that all properties of derived objects are invariant under iso of derivations, this gives an example of a relation between points living in different objects in a derivation preserved under iso (and in fact arbitrary morphisms). We saw in §3.2.1 that we can describe grammatical relations as a typology of facts about dependencies introduced over the course of derivational steps. For example, we might naïvely think of two points $x \in F_p$ and $y \in F_q$ as undergoing *selection* if there is an element $z \in F_r$ such that both x and y project to z. In the case of digraphs, we might think of x as *becoming dependent on* y if there is an edge (a, b) in some DSO F_r such that x projects to a and y to b. In each case, the relevant relationships between points of the derivation are preserved under isomorphisms of derivations, capturing the intuition that isomorphic derivations have isomorphic grammatical relations between corresponding parts.

More formally, we reconstruct grammatical relations within any derivation (P, F), relativized to some step of the derivation. We want to relativize *grammatical relations at* C, where $C = F_c$ for some $c \in P$. We will call a stage $A = F_a$ in (P, F) a *head* if a is a *maximum*, in that $a \leq p$ for any $p \in P$ implies $a = p$. We want $\mathbf{A}$ to be a construct over $\mathbf{FPos}$, in that we are provided with a faithful functor $U : \mathbf{A} \to \mathbf{FPos}$ taking each $\mathbf{A}$ object to its underlying dominance partial order. We will also usually be interested in the case where composition of this functor with the forgetful functor $V : \mathbf{FPos} \to \mathbf{Set}$ is representable. Suppose that A and B are heads of (P, F) in any category $\mathbf{A}$ of DSOs which have underlying domination finite partial orders, such that A and B have roots r_a and r_b with respect to those orderings. Suppose that C is any stage such that $c \leq a$ and $c \leq b$, so that we have maps $!_{c,a} : A \to C$ and $!_{c,b} : B \to C$. We will also have associated constituent inclusions $\mathbf{max}(a) \hookrightarrow U(C)$ and $\mathbf{max}(b) \hookrightarrow U(C)$, where $\mathbf{max}(a)$ is the collection of points of $U(C)$ dominated by some element in the image of $U(!_{c,a})$ and $\mathbf{max}(b)$ is the collection of points of $U(C)$ dominated by some element in the image of $U(!_{c,b})$, such that we have factorizations $U(A) \to \mathbf{max}(a) \hookrightarrow U(C)$ and $U(B) \to \mathbf{max}(b) \hookrightarrow U(C)$. We can then carry out the pullback constructions of §3.2.1 between these functions to find the grammatical relations from A to B *at the stage* C.[6] We will again be especially interested in the case when $U(!_{c,b})(r_b) < U(!_{c,a})(r_a)$ is an immediate domination relation in $U(C)$.

3.3.1 Sums

Much like in §2.5.1, where we formed forests from trees and decomposed forests into trees, there are similar constructions we can perform on derivations. Being able to assemble many derivations into a larger one will be useful for constructing a language recursively. We will need a notion of 'disjoint union' of derivations, but which keeps track of all of the derivational structure. Such 'structured sums' of objects can be defined in any category.

Definition 3.3.4. Let $\mathcal{C}$ be any category, and A and B any two objects of that category. A ***coproduct*** or ***sum*** of A and B in $\mathcal{C}$, if it exists, is an object $A+B$ together with *coprojection* morphisms $\kappa_A : A \to A+B$ and $\kappa_B : B \to A+B$, such that for any pair of morphisms $f : A \to Z$ and $g : B \to Z$, there is a unique morphism $u : A+B \to Z$ such that $f = u \circ \kappa_A$ and $g = u \circ \kappa_B$. See the diagram in Fig. 3.10. We say that a category $\mathcal{C}$ ***has sums*** if there exists a sum for any pair (A, B) of objects from $\mathcal{C}$.

see, e.g., Borceux [23], 2.2

[6] This is similar to the way that other grammatical domains and relations relative to some point in the derivation. "Recall that *domain* and *minimal domain* are understood derivationally, not representationally." Chomsky [9], p. 299.

$$\begin{tikzcd}
A \arrow[r, "\kappa_A"] \arrow[dr, "f"'] & A+B \arrow[d, dashed, "u"] & B \arrow[l, "\kappa_B"'] \arrow[dl, "g"] \\
 & Z &
\end{tikzcd}$$

Figure 3.10: A coproduct diagram.

Like all universal constructions, the sum $A + B$ of two objects is given up to unique isomorphism. We give examples of categories which have sums and the construction of sums in those categories.[7]

- In **Set**, the sum of two sets $A+B$ is given by their disjoint union together with the two inclusions $\kappa_A : A \hookrightarrow A + B$ and $\kappa_B : B \hookrightarrow A + B$. Explicitly, this can be constructed by fixing two indexing singletons $\{1\}$ and $\{2\}$, and constructing $(A \times \{1\}) \cup (B \times \{2\}) = \{(a, 1) \text{ or } (b, 2) \text{ such that } a \in A \text{ or } b \in B\}$ together with the functions mapping $a \mapsto (a, 1)$ and $b \mapsto (b, 2)$.[8] To see that this has the desired universal property, take any two functions $f : A \to Z$ and $g : B \to Z$. We describe the *sum* of these functions as the universal map $u : A + B \to Z$, which must be the uniquely defined function such that $u(a) = f(a)$ and $u(b) = g(b)$.

- In **Proset**, the sum of two preorders A and B can be computed as having as

[7] It is worth noting that all of these sums are *concrete*, in that $U(A+B) = U(A)+U(B)$, where $U : \mathcal{C} \to$ **Set** is the associated forgetful functor turning them into constructs. This need *not* be the case in general: that is, the sum of two objects in a construct need not have underlying set which is the disjoint union of their underlying sets.

[8] Like the Cartesian product × itself, there are many ways to construct a disjoint union. However, described by the universal property, all methods of constructing the disjoint union will be in a canonical bijection with each other.

underlying set the disjoint union of the underlying sets of A and B. We turn this set into a preorder using the relations $x \leq y$ if and only if $x \leq y$ in A or $x \leq y$ in B, depending on which set the elements originated from. The sums in **Pos** and **FPos** can be computed identically.

- For a more complex example, take our category $\mathbf{A_L}$ of sets A together with a dominance preorder $\leq_A$, precedence preordering $\preceq_A$, and collection of unary predicates λ_A on A, one for each $\lambda \in \mathbf{L}$. The sum of two objects $(A, \leq_A, \preceq_A, \{\lambda_A\}_{\{\lambda \in \mathbf{L}\}})$ and $(B, \leq_B, \preceq_B, \{\lambda_B\}_{\{\lambda \in \mathbf{L}\}})$ has underlying set which is the disjoint union of A and B. It has dominance relations $x \leq y$ if and only if $x \leq_A y$ or $x \leq_B y$. It has precedence relations $x \preceq y$ if and only if $x \preceq_A y$ or $x \preceq_B y$. $\lambda_{A+B}(x) =$ true if and only if $\lambda_A(x) =$ true or $\lambda_B(x) =$ true for each $\lambda \in \mathbf{L}$. To see that this has the universal property, take any two $\mathbf{A_L}$ morphisms $f : (A, \leq_A, \preceq_A, \{\lambda_A\}_{\{\lambda \in \mathbf{L}\}}) \to (Z, \leq_Z, \preceq_Z, \{\lambda_Z\}_{\{\lambda \in \mathbf{L}\}})$ and $g : (B, \leq_B, \preceq_B, \{\lambda_B\}_{\{\lambda \in \mathbf{L}\}}) \to (Z, \leq_Z, \preceq_Z, \{\lambda_Z\}_{\{\lambda \in \mathbf{L}\}})$. Note that the *function* $u : A + B \to Z$ sending $a \mapsto f(a)$ and $b \mapsto g(b)$ is the unique function such that $u \circ \kappa_A = f$ and $u \circ \kappa_B = g$, so if we can show that it is a *morphism*, then we will have constructed the required universal map. To see that it is, note that if $x \leq y$ in $A + B$, then either $x \leq y$ in A, in which case we already know $u(x) \leq u(y)$, since $f(x) \leq f(y)$ and f is a morphism, or $x \leq y$ in B, in which case $u(x) \leq u(y)$ is still true since g is a morphism. We can argue similarly for the $\preceq$ relation on $A + B$. Finally, note that $\lambda_{A+B}(x) =$ true only if $\lambda_A(x) =$ true or $\lambda_B(x) =$ true. If x is from A, then we know that

$u(a) = f(a)$, and since f is a morphism, we must have $\lambda_Z(f(a)) = \text{true}$. We argue identically for B, and so u is in fact an $\mathbf{A_L}$-morphism.

Claim 3.3.4. Any category $\mathscr{D}(\mathbf{A})$ has a coproduct for each pair of derivations (P, F) and (Q, G). It is given by a state space with underlying set $P + Q$, with the coproduct preordering on it. We construct $F + G$ as the functor $(F + G)(x) = F_x$ or G_x, depending on whether $x \in P$ or Q, and $(F + G)(x \leq y)$ is $F(x \leq y)$ or $G(x \leq y)$. The coprojections are given by the inclusions $P, Q \hookrightarrow P + Q$, such that the coordinates of the natural transformations are isomorphisms.

3.3.2 Yields

When a derivation (P, F) has a state diagram with root r such that $r \leq p$ for all $p \in P$, then for all objects in (P, F), we have a unique map $!_{r,p} : F_p \to F_r$. In this case, it makes sense to think of F_r as the *yield* of the derivation, that is, the final output object. When the category of DSOs $\mathbf{A}$ has sums, then intuitively if derivations-with-roots (P, F) and (Q, G) have yields F_r and G_s, their sum should yield $F_r + G_s$. We now want to see when we can define a yield functor in general, such that this functor takes any derivation in $\mathscr{D}(\mathbf{A})$ to its yield in $\mathbf{A}$.

Recall that for any $\mathbf{A}$, we have an inclusion $i : \mathbf{A} \hookrightarrow \mathscr{D}(\mathbf{A})$ taking each DSO a to $i(a) = (*, !_a : * \to \mathbf{A})$, which we usually just write as a. For any derivation (P, F) and DSO a, a morphism $(!, \mu) : (P, F) \to a$ is given by a morphism $\mu_p : F_p \to a$ for each $p \in P$. If a derivation (P, F) has a final state (root) r, then a morphism from that derivation to any derived object is determined totally by its value on the final

DSO F_r. Similarly, if (P,F) and (Q,G) are two derivations with roots r and s, a morphism $(P+Q,F+G) \to a$ is determined totally by a pair of maps $F_r \to a$ and $G_s \to a$, and hence, if $\mathbf{A}$ has sums, a single map $F_r + G_s \to a$. We want a general method which assigns to each derivation its 'yield' as a functor $\top : \mathscr{D}(\mathbf{A}) \to \mathbf{A}$ which has the property that derivation morphisms from (P,F) to any DSO a are determined totally by $\mathbf{A}$-morphisms $\top(P,F) \to a$. Moreover, we actually want natural transformations $\epsilon : \top \circ i \to 1_{\mathbf{A}}$ and $\eta : \mathbf{1}_{\mathscr{D}(\mathbf{A})} \to i \circ \top$ such that ϵ is a natural isomorphism and for any derivation (P,F), DSO a, and derivation morphism $(!,\mu) : (P,F) \to i(a)$, we have a factorization $(!,\mu) = i(\epsilon_a) \circ i(\top(!,\mu)) \circ \eta_{(P,F)} : (P,F) \to i(\top(P,F)) \to i(\top(i(a))) \approx i(a)$. This first derivation homomorphism takes the derivation to its yield, the second is the induced morphism from its yield to a, and the last is the isomorphism $i(\top(i(a))) \approx i(a)$ given by ϵ.

We already know that for this to be possible, $\mathbf{A}$ must have all sums, since we can always take sums of derivations. Consider now the fact that every category of derivations will have an empty derivation $(\emptyset, E)$ such that $\emptyset$ is the empty set, and $E : \emptyset \to \mathbf{A}$ is the only possible functor with empty image. This is the derivation with no stages at all. For any DSO a, we will have a unique derivation morphism $(!,\mu) : (\emptyset, E) \to i(a)$ which is given by the inclusion $m : \emptyset \hookrightarrow *$, such that μ has no coordinate morphisms. If this derivation is to have a yield $\top(\emptyset, E)$, it must be an object $\mathbf{0}$ of $\mathbf{A}$ with the property that there is a unique $\mathbf{A}$-morphism $! : \mathbf{0} \to a$ for any DSO a. Such an object in any category is called an ***initial object***.

Definition 3.3.5. In any category $\mathcal{C}$, an object $\mathbf{0}$ is called an ***initial object*** if for any object C of $\mathcal{C}$, there is a unique $\mathcal{C}$-morphisms $! : \mathbf{0} \to C$.

see, e.g. Borceux [23], 2.3

Such objects can often be thought of as *empty objects*. Consider the following examples.

- **Set** has an initial object $\emptyset$. For any set X, there is a unique function to it from $\emptyset$, given by the inclusion $! : \emptyset \hookrightarrow X$.
- **Proset**, **Pos**, **FPos**, and $\mathbf{A_L}$ all have initial object $\emptyset$ with the only possible orderings and type-determinations on it.
- $\mathscr{D}(\mathbf{A})$ has an initial object given by the empty derivation $(\emptyset, E)$.

Finally, to guarantee the existence of a yield functor, we need one more construction to be possible in $\mathbf{A}$ known as a *pushout*.

Definition 3.3.6. Let $\mathcal{C}$ be any category, and let $f : A \to B$ and $g : A \to C$ be any two morphisms. A ***pushout*** of f and g is an object $B +_A C$ together with morphisms $\kappa_B : B \to B +_A C$ and $\kappa_C : C \to B +_A C$ such that $\kappa_B \circ f = \kappa_C \circ g$, and for any $\kappa'_B : B \to Z$ and $\kappa'_C : B \to Z$ such that $\kappa'_B \circ f = \kappa'_C \circ g$, there is a unique morphism $u : B +_A C \to Z$ such that $u \circ \kappa_B = \kappa'_B$ and $u \circ \kappa_C = \kappa'_C$. See Fig. 3.11. We again say that $\mathcal{C}$ ***has pushouts*** if it has a pushout for every pair of morphisms $(f : A \to B, g : A \to C)$.

Pushouts are again determined up to a unique isomorphism. We give constructions of them in some categories. We will need the notion of an *equivalence*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "\kappa_B"] \arrow[ddr, bend left, "\kappa'_B"] & \\
C \arrow[r, "\kappa_C"'] \arrow[drr, bend right, "\kappa'_C"'] & B +_A C \arrow[dr, dashed, "u"] & \\
 & & Z
\end{tikzcd}$$

Figure 3.11: A pushout diagram.

relation on a set X. A relation $E \subset X \times X$ is an *equivalence relation* if the following properties hold: (reflexivity) $(x, x) \in E$ for each $x \in X$; (symmetry) $(x, y) \in E$ implies $(y, x) \in E$; and (transitivity) if $(x, y) \in E$ and $(y, z) \in E$, then $(x, z) \in E$. We write $x \sim y$ if $(x, y) \in E$. We will also need the notion of a *quotient* by an equivalence relation. Let X be any set and E an equivalence relation on it. Then, for any element $x \in X$ there is a set $\{y \in X \text{ such that } x \sim y\}$ which we denote $[x]$, called the *equivalence class* of x. Note that for any points $x, y \in X$, either $[x] = [y]$ (if and only if $x \sim y$) or $[x] \cap [y] = \emptyset$ (if and only if $x \not\sim y$). We write the set $X/E = \{[x] \text{ such that } x \in X\}$, and call this set the *quotient* of X by E. There is a canonical function $q : X \to X/E$ called the *quotient map* which sends $x \mapsto [x]$. This is well-defined, since each element belongs to exactly one equivalence class. For any relation $R \subset X \times X$, there is a unique smallest equivalence relation on X containing R. It can be computed by taking the intersection of all equivalence relations E on X such that $R \subset E$. Note that if R is an equivalence relation, this relation is just R itself. We call this the equivalence relation *generated* by R.

- Consider two functions $f : A \to B$ and $g : A \to C$ in **Set**. We construct $B +_A C$ as follows. First construct the disjoint union $B + C$. We know that if

$a \in A$ is any element, then we must have $\kappa_A(f(a)) = \kappa_B(g(a))$ in $B +_A C$. We construct an equivalence relation on $B + C$. We start by building a relation R on $B + C$ consisting of the relations $(f(a), g(a))$ for each element $a \in A$. We then take the equivalence relation E generated by R, and construct the quotient map $q : B + C \to (B + C)/E$. We define $(B + C)/E \equiv B +_A C$ and we give the two coprojections to it as $\kappa_B \equiv q \circ i_B : B \hookrightarrow B + C \to B +_A C$ and $\kappa_C \equiv q \circ i_C : B \hookrightarrow B + C \to B +_A C$, where i_B and i_C are the inclusions into the disjoint union.

To see that $(B +_A C, \kappa_B, \kappa_C)$ is actually a pushout, we must check two things. First, we must check that $\kappa_B \circ f = \kappa_C \circ g$. To see this, take any element $a \in A$. By definition, $\kappa_B(f(a)) = [f(a)]$ and $\kappa_C(g(a)) = [g(a)]$. But since $(f(a), g(a)) \in R$ by definition, these elements are equivalent in the equivalence relation generated by R, hence $[f(a)] = [g(a)]$.

Now we must check that for any functions $\kappa'_B : B \to Z$ and $\kappa'_C : C \to Z$ such that $\kappa'_B \circ f = \kappa'_C \circ g$, we have a unique function $u : B +_A C \to Z$ such that $u \circ \kappa_B = \kappa'_B$ and $u \circ \kappa_C = \kappa'_C$. For any $x \in B +_A C$, x must come from some $b \in B$ or $c \in C$ - i.e. there exists a $b \in B$ or $c \in C$ such that $\kappa_B(b) = x$ or $\kappa_C(c) = x$. $u(x)$ must be equal to $\kappa'_B(b)$ or $\kappa'_C(c)$ if it is to meet the conditions $u \circ \kappa_B = \kappa'_B$ and $u \circ \kappa_C = \kappa'_C$. We define $u(x)$ to be the element $\kappa'_B(b)$, where b is any $b \in B$ such that $q(b) = x$, or $\kappa'_C(c)$, where $c \in C$ is any element such that $q(c) = x$. We must make sure that this is well-defined. First note that for any $s, t \in B + C$, $q(s) = q(t)$ if and only if there is some $a \in A$ such that

a has image s under f or g and image t under f or g. If $s, t \in B$, this is only possible only if $s = t$ since f is a function, and similarly if $s, t \in C$, this is possible only if $x = y$ since g is a function. So u might only be ill-defined if s and t are from distinct sets. Let $b \in B$ be any element such that $q(b) = x$ and $c \in C$ any element such that $q(c) = x$. This implies that there is some $a \in A$ such that $f(a) = b$ and $g(a) = c$. To make sure that u is well-defined, we must check that $\kappa'_B(b) = \kappa'_C(c)$, since u as we have defined it wants to send x to both of these elements of Z. But $\kappa'_B(b) = \kappa'_B(f(a)) = \kappa'_C(g(a)) = \kappa'_C(c)$, so this map is well-defined. We have $u \circ \kappa_B = \kappa'_B$ and $u \circ \kappa_C = \kappa'_C$ by the construction of u, and so u gives the unique required function.

- Consider any two order-preserving functions $f : A \to B$ and $g : A \to C$ in **Proset**. We construct their pushout as follows. Let $B +_A C$ have underlying set $(B + C)/E$ as above, with inclusion functions also as above. We turn $B +_A C$ into a preorder by taking the smallest preorder containing the all the relations of the form $\kappa_B(b) \leq \kappa_B(b')$ whenever $b \leq b'$ in B and $\kappa_C(c) \leq \kappa_C(c')$ whenever $c \leq c'$ in C.

- We construct pushouts in **Pos**. First note that if $(P, \leq)$ is any preordered set, we can construct an equivalence relation on it $p \sim p'$ whenever $p \leq p'$ and $p' \leq p$. Denote this equivalence relation as E and construct the quotient map $q : P \to P/E$. P/E inherits a preordering from P by taking the smallest preorder containing all relations of the form $q(p) \leq q(p')$ whenever $p \leq p'$ in P. This preordering on P/E is actually a partial order, and we call it the

soberification of P, and the quotient q is order-preserving. Now, given two order-preserving functions between partial orders $f : A \to B$ and $g : A \to C$, we construct a pushout. First, construct the pushout of preorders with coprojections $B \hookrightarrow B + C \to (B + C)/E$ and $C \hookrightarrow B + C \to (B + C)/E$. To complete a computation of the pushout in **Pos**, compose each of these order-preserving functions with the soberification quotient. Note that this gives an example of a pushout which is not *concrete*, in that the underlying set of the pushout of f and g is not necessarily the pushout of the underlying sets.

We now define yields in general and characterize when they exist.

Claim 3.3.5. Let $\mathbf{A}$ be any category and let $\mathscr{D}(\mathbf{A})$ be the category of derivations over it, where we write $i : \mathbf{A} \to \mathscr{D}(\mathbf{A})$ for the canonical inclusion of DSOs into derivations. We say that a functor $\top : \mathscr{D}(\mathbf{A}) \to \mathbf{A}$ together with a natural transformation $\eta : \mathbf{1}_{\mathscr{D}(\mathbf{A})} \to i \circ \top$ and natural isomorphism $\epsilon : \top \circ i \to \mathbf{1}_{\mathbf{A}}$, if it exists, is a ***yield functor*** if every morphism of the form $(!, \mu) : (P, F) \to i(a)$ for any (P, F) in $\mathscr{D}(\mathbf{A})$ and a in $\mathbf{A}$ factors as $i(\epsilon_a) \circ i(\top(!, \mu)) \circ \eta_{(P,F)} : (P, F) \to i(\top(P, F)) \to i(\top(i(a))) \approx i(a)$.

This functor is unique up to natural isomorphism, in that if $\top$ and $\top'$ are any two yield functors, then they are naturally isomorphic. Furthermore, this functor exists if and only if $\mathbf{A}$ has all sums, pushouts, and an initial object.

Notably, if (P, F) has a root r, then $\top(P, F)$ is isomorphic to F_r, and if $(P + Q, F + G)$ is the sum of rooted derivations, then $\top(P + Q, F + G)$ is isomorphic to $F_r + G_s$. Also, the yield of the empty derivation is the initial object of $\mathbf{A}$. The proof of these facts, as well as a construction of the yield functor, is actually very straightforward, but only after introducing an extremely important piece of category theoretic machinery called an *adjunction*, which we will develop in §3.4. However, we delay adjunctions so that we can immediately apply sums and yields to the recursive construction of languages.

3.3.3 Grammars, Languages, and Equivalences

We can define a language to be a subclass $\mathscr{L} \subset \mathscr{D}(\mathbf{A})$, considered as a full subcategory. Given languages $i : \mathscr{L} \hookrightarrow \mathscr{D}(\mathbf{A})$ and $j : \mathscr{M} \hookrightarrow \mathscr{D}(\mathbf{A})$, we can define ***strong extensional equivalences*** between languages. Two languages are equivalent iff for every derivation Δ in $\mathscr{L}$ there is an isomorphic derivation in $\mathscr{M}$ and conversely.

Definition 3.3.7. A ***strong extensional equivalence*** between languages is a pair of functors $F : \mathscr{L} \leftrightarrows \mathscr{M} : G$ together with natural isos $j \circ F \approx i$ and $i \circ G \approx j$. This can be strengthened to a ***strong extensional isomorphism*** between languages by requiring $j \circ F = i$ and $i \circ G = j$.

However, we are usually interested in languages which are recursively constructed in some simple way. In these cases, strong extensional equivalences of languages also become more meaningful. We define a *grammar* over $\mathbf{A}$ to be a pair $\mathscr{G}$ consisting of a set LEX of objects of $\mathbf{A}$ together with a set RULES of rules of any finite arity over $\mathbf{A}$. We are usually only interested in cases where both sets are finite, though nothing essential changes if they are not. Intuitively, given an n-ary rule $G \in$ RULES, an n-tuple of derivations $((P_1, F_1), \ldots, (P_n, F_n))$ with yields

$$\begin{tikzcd}
 & & & & Z & & & & \\
 & A_1 \arrow[urrr, "f_1"] & & & \cdots & & & A_n \arrow[ulll, "f_n"'] & \\
W \arrow[ur] & \cdots & X \arrow[ul] & & & & Y \arrow[ur] & \cdots & Z \arrow[ul]
\end{tikzcd}$$

Figure 3.12: Informal picture of the derivation constructed by extending a family of rooted derivations with yields A_i along a SC $(f_i : A_i \to Z : 1 \leq i \leq n)$.

$\top(P_i, F_i) = A_i$, and SC $(f_i : A_i \to Z : 1 \leq i \leq n) \in G(A_1, \ldots, A_n)$, there is an *extension* of the tuple (P_i, F_i) along $(f_i : A_i \to Z : 1 \leq i \leq n)$. Its states should be the disjoint union of all states in the (P_i, F_i) together with Z. Since each component of the SC $f_i : A_i \to Z$ corresponds to a morphism of derivations $(!, \mu^i) : (P_i, F_i) \to Z$, which in turn corresponds to a family of morphisms $\mu^i_p : F_{i,p} \to Z$, one for each state $F_{i,p}$ in (P_i, F_i), we will have an **A**-morphism from each state $F_{i,p}$ in one of the derivations to Z. The connecting SCs in this extension should then consist of all the SCs from each of the derivations in the tuple, together with these 'new' maps μ^i_p from each $F_{i,p}$ to Z. This is especially intuitive when each of the input derivations has a root, in which case this root will be A_i. A figure representing such an extension is given in Fig. 3.12.

By the definition of coproducts and yields, the tuple of morphisms given by an SC corresponds to a single morphism $\coprod_{1\leq i\leq n}(P_i, F_i) \equiv (P_1, F_1) + \ldots + (P_n, F_n) \to i(Z)$. Viewing SCs this way is useful, as we can then describe extensions in a uniform way, regardless of the arity of the SC. We want to functorialize extensions, such that given a morphism from a derivation to a derived object $(!, \mu) : (P, F) \to Z$, we have an extended derivation consisting of (P, F) stages with Z on top, together with all the SCs in (P, F) plus SCs of the form $\mu_p : F_p \to Z$. We can then use this functor to construct derivations recursively.

3.3.3.1 Extensions

We will construct a functor which takes in maps from derivations to DSOs and returns a derivation containing all of those derivations with the new DSO 'on top', connected by the new SCs. Given a map $\mu : (P, F) \to Z$ from a derivation to a DSO, we will formalize the extension of (P, F) along μ as the smallest derivation containing (P, F), Z, and the morphisms $\mu_p : F_p \to Z$. We first construct a category of operations on derivations.

Definition 3.3.8. Given categories and functors

$$E \xrightarrow{T} C \xleftarrow{S} D$$

the ***comma category*** $(T \downarrow S)$, also written (T, S), has as objects all triples $\langle e, d, f\rangle$ with $d \in \mathit{Obj}\ D$, $e \in \mathit{Obj}\ E$ and $f : Te \to Sd$, and as arrows $\langle e, d, f\rangle \to \langle e', d', f'\rangle$ all pairs $\langle k, h\rangle$ of arrows $k : e \to e'$, $h : d \to d'$ such that $f' \circ Tk = Sh \circ f$. In pictures,

$$\begin{tikzcd}
\text{Objects} & Te \arrow[d, "f"] & \text{Arrows} & Te \arrow[r, "Tk"] \arrow[d, "f"] & Te' \arrow[d, "f'"] \\
\langle e, d, f\rangle & Sd \text{ ;} & \langle k, h\rangle & Sd \arrow[r, "Sh"] & Sd' \text{ ,}
\end{tikzcd}$$

with the square commutative. The composite $\langle k', h'\rangle \circ \langle k, h\rangle$ is $\langle k' \circ k, h' \circ h\rangle$, when defined.

Mac Lane [26], II.6

$$\begin{tikzcd}(P,F) \arrow[r, "\mu"] \arrow[d, "(f,\phi)"'] & Z \arrow[d, "k"] \\ (Q,G) \arrow[r, "\nu"'] & Y\end{tikzcd}$$

Figure 3.13: A morphism of operations on derivations.

We then define the *category of operations on* $\mathscr{D}(\mathbf{A})$ to be the comma category given by $\mathbf{1}_{\mathscr{D}(\mathbf{A})} : \mathscr{D}(\mathbf{A}) \to \mathscr{D}(\mathbf{A}) \leftarrow \mathbf{A} : i$, which we write $\mathscr{D}(\mathbf{A})/\mathbf{A}$. Its objects are morphisms $(!, \mu) : (P, F) \to Z$, which we will just write as μ, from any derivation (P, F) to a DSO Z. A morphism of this category from $\mu : (P, F) \to Z$ to $\nu : (Q, G) \to Y$ is a pair $((f, \phi), k)$ where $(f, \phi) : (P, F) \to (Q, G)$ is a derivation morphism, and $k : Z \to Y$ is an $\mathbf{A}$-morphism, such that $\nu \circ (f, \phi) = i(k) \circ \mu$ as derivation morphisms. See Fig. 3.13. We want to give a characterization of extensions as a functor $\mathrm{ext} : \mathscr{D}(\mathbf{A})/\mathbf{A} \to \mathbf{A}$ up to natural isomorphism. We are going to characterize it using a universal property. Given an operation $\mu : (P, F) \to Z$, we will construct the *extension of* (P, F) *along* μ as the 'simplest' derivation $\mathrm{ext}(\mu)$ which both (P, F) and Z map into, such that the image of each state F_p has a SC to the image of Z in $\mathrm{ext}(\mu)$, such that $\mu_p : F_p \to Z$ is carried to this SC.

We first formalize the property of morphisms out of (P, F) and Z given above, and then describe the universal such one. Given an operation $\mu : (P, F) \to Z$, we will be interested in derivations (Q, G) which both (P, F) and Z map into, and hence we can just consider maps out of the sum $(P, F) + i(Z)$. The underlying finite partial order of this sum is the sum of P and a single index for the stage Z, which we call z. We say that a derivation morphism $(n, \nu) : (P, F) + i(Z) \to (Q, G)$ *takes* μ*-images to SCs* if it has the following properties: (1) $n(z) \leq n(p)$ for all $p \in P$, and (2) we

$$\begin{tikzcd}
F_p \arrow[r, "\mu_p"] \arrow[d, "\nu_p"'] & Z \arrow[d, "\nu_z"] \\
G_{n(p)} \arrow[r, "!_{n(z),n(p)}"'] & G_{n(z)}
\end{tikzcd}$$

Figure 3.14: $(n, \nu) : (P, F)+Z \to (Q, G)$ takes μ-images to SCs if the above diagram commutes for each $p \in P$.

have $\nu_z \circ \mu_p = !_{n(z),n(p)} \circ \nu_p$ for every $p \in P$. See Fig. 3.14. We say that a morphism $\mathrm{ext}_\mu : (P, F) + Z \to \mathrm{ext}(\mu)$ is the *universal* such morphism if it takes μ-images to SCs, and for any derivation morphism $(n, \nu) : (P, F) + i(Z) \to (Q, G)$ which takes μ-images to SCs, there is a unique derivation morphism $(x, \chi) : \mathrm{ext}(\mu) \to (Q, G)$ such that $(n, \nu) = (x, \chi) \circ \mathrm{ext}_\mu$. This formalizes the notion that the extension is the 'simplest derivation' that (P, F), Z, and the SCs in μ map into.

Suppose that $((f, \phi), k) : (\mu : (P, F) \to Z) \to (\nu : (Q, G) \to Y)$ is a morphism of operations. We will show that there is a morphism of derivations $\mathrm{ext}((f, \phi), k) : \mathrm{ext}(\mu) \to \mathrm{ext}(\nu)$ which maps $p \mapsto f(p)$ and $z \mapsto y$ with coordinates $\phi_p : F_p \to G_{f(p)}$ and $k : Z \to Y$ which is induced by the universal property of extensions, and hence leads to a functor $\mathrm{ext} : \mathscr{D}(\mathbf{A})/\mathbf{A} \to \mathbf{A}$. Write the extensions $\mathrm{ext}(\mu) = (M, S)$ and $\mathrm{ext}(\nu) = (N, T)$, and the extension $\mathrm{ext}_\nu \equiv (b, \beta) : (Q, G) + Y \to (N, T)$. Suppose we have a morphism of operations given by the pair of morphisms $(f, \phi) : (P, F) \to (Q, G)$ and $k : Z \to Y$. We can take the sum of derivations (Q, G) and Y to get a derivation $(Q, G) + Y$ and compose (f, ϕ) and k with the coprojection inclusions to get a pair of maps $\kappa_{(Q,G)} \circ (f, \phi) : (P, F) \to (Q, G) \to (Q, G) + Y$ and $\kappa_Y \circ k : Z \to Y \to (Q, G) + Y$. We can then take the sum of (P, F) and Z, and use the coproduct property to get a single map $(\kappa_{(Q,G)} \circ (f, \phi)) + (\kappa_Y \circ k) : (P, F) + Z \to (Q, G) + Y$.

$$\begin{tikzcd}
F_p \arrow[r, "\mu_p"] \arrow[d, "\gamma_p=\phi_p"'] & Z \arrow[d, "k=\gamma_z"] \\
G_{f(p)} \arrow[r, "\nu_{f(p)}"'] \arrow[d, "\beta_{f(p)}"'] & Y \arrow[d, "\beta_y"] \\
N_{b(f(p))} \arrow[r, "!_{b(y),b(f(p))}"'] & N_{b(y)}
\end{tikzcd}$$

Figure 3.15: The composite of $\text{ext}_\nu \equiv (b, \beta)$ and the sum $(\kappa_{(Q,G)} \circ (f, \phi)) + (\kappa_Y \circ k) \equiv (g, \gamma)$ is a map which takes μ-images to SCs.

We will write this derivation morphism as (g, γ). For any $p \in P$, g maps $p \mapsto f(p)$, and it maps $z \mapsto y$. For each $p \in P$, $\gamma_p : F_p \to G_{f(p)}$ will simply be the component ϕ_p, while $\gamma_z : Z \to Y$ will just be k. We now show that the composite $(b, \beta) \circ (g, \gamma)$ takes μ-images to SCs, in which case, by definition, there will be a uniquely induced $(x, \chi) : (M, S) \to (N, T)$, such that $(b, \beta) \circ (g, \gamma) = (x, \chi) \circ \text{ext}_\mu$. This morphism will give the behavior of ext on morphisms of $\mathscr{D}(\mathbf{A})/\mathbf{A}$. First, note that for all $p \in P$, we have $b(g(z)) = b(y) \leq b(g(p))$, since (b, β) takes ν-images to SCs. Now note that $k \circ \mu_p = \nu_{f(p)} \circ \phi_p$, since $((f, \phi), k)$ is a morphism of operations. However, the maps ϕ_p and k are just the components of (g, γ); that is, $k = \gamma_z$ and $\phi_p = \gamma_p$. Also, $\beta_y \circ \nu_{f(p)} = !_{b(y),b(f(p))} \circ \beta_{f(p)}$ since (b, β) takes ν-images to SCs. We can compose these squares to obtain the equality $(\beta_y \circ \gamma_z) \circ \mu_p = !_{b(y),b(f(p))} \circ (\beta_{f(p)} \circ \gamma_p)$, i.e. $(b, \beta) \circ (g, \gamma)$ takes μ-images to SCs. See Fig. 3.15.

To finally show that ext actually gives a functor, we must construct $\text{ext}_\mu : (P, F) + Z \to \text{ext}(\mu)$ for any operation $\mu : (P, F) \to Z$. We construct this derivation as follows, writing $\text{ext}(\mu) \equiv (M, S)$. The underlying partial order of $\text{ext}(\mu)$ is the sum of P and the singleton $\{z\}$ as partial orders, and we add relations $z \leq p$ for all $p \in P$. We let $!_{p,p'} : S_{p'} \to S_p$ just be the SC $!_{p,p'} : F_{p'} \to F_p$ from (P, F) whenever $p, p' \in P$. We define $!_{z,p} : S_p \to S_z$ be $\mu_p : F_p \to Z$ otherwise. These data give a derivation since μ was a morphism of derivations. Furthermore, to show that ext_μ is a derivation morphism, we must just show that the inclusions $(P, F) \hookrightarrow \mathrm{ext}(\mu)$ and $Z \to \mathrm{ext}(\mu)$ are derivation morphisms. But this is obvious, since the first is simply a subderivation inclusion, and the second is just the map sending Z to S_z isomorphically. To show this has the universal property, take any derivation morphism $(f, \phi) : (P, F) + Z \to (Q, G)$ taking μ-images to SCs. We define $(x, \chi) : (M, S) \to (Q, G)$ to be the derivation morphism taking $p \mapsto f(p)$ and $z \mapsto f(z)$, together with the components $\chi_p = \phi_p$ and $\chi_z = \phi_z$. x is evidently order-preserving on the points $P \subset M$, since f is. For the only 'new' relations $z \leq p$, we know that $f(z) = x(z) \leq x(p) = f(p)$ is order-preserving since (f, ϕ) maps μ-images to projection. Similarly, we already know that components χ_p are all compatible since (f, ϕ) restricted to (P, F) is a derivation morphism, which describes the behavior of χ on $P \subset M$. For the SCs $!_{z,p} : S_p \to S_z$, we again know that $\chi_z \circ !_{z,p} \equiv \phi_z \circ \mu_p = !_{fz,fp} \circ \phi_p \equiv !_{fz,fp} \circ \chi_p$ since (f, ϕ) takes μ-images to SCs. We have finally proven the following claim.

Claim 3.3.6. Let $\mu : (P, F) \to Z$ be any morphism from a derivation (P, F) to a DSO Z. Then there is a universal derivation morphism $\mathrm{ext}_\mu : (P, F) \to \mathrm{ext}(\mu) \equiv (M, S)$ taking μ-images to SCs given by the derivation with states $P + \{z\}$, together with the order relations $z \leq p$ for all $p \in P$ and $p \leq p'$ whenever the relation holds in P. $S_p = F_p$ whenever $p \in P$ and $S_z = Z$. Finally, $!_{p,p'} : S_{p'} \to S_p$ is equal to $!_{p,p'} : F_{p'} \to F_p$ in (P, F) whenever $p, p' \in P$, and $!_{z,p} : S_p \to S_z$ is equal to $\mu_p : F_p \to Z$ whenever $p \in P$.

This determines a functor $\mathrm{ext} : \mathscr{D}(\mathbf{A})/\mathbf{A} \to \mathbf{A}$ up to natural isomorphism, which maps μ to $\mathrm{ext}(\mu)$. Given a morphism of operations $((f, \phi), k) : (\mu : (P, F) \to Z) \to (\nu : (Q, G) \to Y)$, ext returns the morphism of derivations $\mathrm{ext}((f, \phi), k) : \mathrm{ext}(\mu) \to \mathrm{ext}(\nu)$ which maps $p \mapsto f(p)$ and $z \mapsto y$ and has coordinate morphisms $\phi_p : F_p \to G_{f(p)}$ and $k : Z \to Y$. It arises as the unique morphism $(x, \chi) : \mathrm{ext}(\mu) \to \mathrm{ext}(\nu)$ such that $\mathrm{ext}_\nu \circ ((\kappa_{(Q,G)} \circ (f, \phi)) + (\kappa_Y \circ k)) = (x, \chi) \circ \mathrm{ext}_\mu$. This factorization exists and is unique since $\mathrm{ext}_\nu \circ ((\kappa_{(Q,G)} \circ (f, \phi)) + (\kappa_Y \circ k))$ takes μ-images to SCs and ext_μ is universal with respect to this property.

3.3.3.2 Languages from Grammars

We can finally give a recursive construction of languages given a grammar $\mathscr{G} = (\textsc{lex}, \textsc{rules})$.

Definition 3.3.9. Let $\mathscr{G} = (\textsc{lex}, \textsc{rules})$ be a grammar. Then we define the ***language generated by*** $\mathscr{G}$, $\mathscr{L}(\mathscr{G})$, to be the following class of derivations:

- If $A \in \textsc{lex}$, then $i(A)$ is in $\mathscr{L}(\mathscr{G})$.
- If (P_i, F_i) are derivations in $\mathscr{L}(\mathscr{G})$ with yields A_i, $G \in \textsc{rules}$ any rule, and $(f_i : A_i \to Z : 1 \leq i \leq n)$ any element of $G(A_1, \ldots, A_n)$, then we obtain an n-tuple of derivation morphisms $i(f_i) \circ \eta_{(P_i,F_i)} : (P_i, F_i) \to i(\top(P_i, F_i)) = i(A_i) \to i(Z)$ by composing the component of the natural transformation taking each derivation to its yield with the component SC. We can then take

the sum of derivation morphisms to obtain a single morphism which we write $\mu : \coprod_{1 \leq i \leq n}(P_i, F_i) \to Z$. In this case, $\mathrm{ext}(\mu)$ is in $\mathscr{L}(\mathscr{G})$.

Such grammars only generate derivations whose underlying partial orders P are trees. Such grammars are constructed from rules which are 'Markovian', in that they only rely on the yield - in this case, the final state - of each of the input derivations. However, we can also generalize rules to ones which care about the structure of the whole derivation.[9] We now re-define an n-ary rule on *derivations* G to be an assignment which takes in an n-tuple of derivations $((P_1, F_1), \ldots, (P_n, F_n))$ and returns a set $G((P_1, F_1), \ldots, (P_n, F_n))$ of n-tuples of derivation morphisms $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n)$. We again require that $G((P_1, F_1), \ldots, (P_n, F_n))$ contain no isomorphic SCs, again defining an isomorphism of SCs $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n)$ and $(\nu^i : (P_i, F_i) \to Y : 1 \leq i \leq n)$ as an isomorphism $u : Z \to Y$ of $\mathbf{A}$-objects such that $u \circ \mu^i = \nu^i$ for each i. We now give a revised definition.

Definition 3.3.10. Let $\mathscr{G} = (\textsc{lex}, \textsc{rules})$ be a grammar. Then we define the ***language generated by*** $\mathscr{G}$, $\mathscr{L}(\mathscr{G})$, to be the following class of derivations:

- If $A \in \textsc{lex}$, then $i(A)$ is in $\mathscr{L}(\mathscr{G})$.
- If (P_i, F_i) are derivations in $\mathscr{L}(\mathscr{G})$, $G \in \textsc{rules}$ any rule, and $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n)$ any element of $G((P_1, F_1), \ldots, (P_n, F_n))$, we can take the sum of derivation morphisms to obtain a single morphism which we write $\mu : \coprod_{1 \leq i \leq n}(P_i, F_i) \to Z$. In this case, $\mathrm{ext}(\mu)$ is in $\mathscr{L}(\mathscr{G})$.

[9] This will make the statement of rules and extensions much more natural, and it is always straightforward to restrict to the Markovian case.

Grammars of the first sort are clearly special grammars of the second sort, where each rule is defined for any tuple of derivations whenever it is defined for their yields, and these SCs can be obtained by composing with the coordinates of the natural transformation taking each derivation to its yield.

3.3.3.3 Equivalences of Languages

We now want to consider two grammar $\mathscr{G}$ and $\mathscr{H}$ which produce equivalent languages $\mathscr{L}(\mathscr{G})$ and $\mathscr{L}(\mathscr{H})$. Intuitively, two grammars lead to extensionally equivalent languages if and only if for each lexical item in one, there is an isomorphic lexical item in the other and conversely, and for each tuple of generated derivations $((P_1, F_1), \ldots, (P_n, F_n))$ and structural change $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n)$ in one, there is a tuple of isomorphic derivations $((Q_1, R_1), \ldots, (Q_n, R_n))$ in the other with an isomorphic structural change on them, and conversely. However, we must be careful, as it does not have to be the case that the i^{th} derivation in the first tuple is isomorphic to the i^{th} one in the second: rather, we may rearrange which derivations in the tuple are in correspondence, and the SCs will be isomorphic once the input derivations have been appropriately aligned.

Claim 3.3.7. Let $\mathscr{G}$ and $\mathscr{H}$ be two grammars. The following are equivalent:

- For every $A \in \textsc{lex}_{\mathscr{G}}$, there is an isomorphic $B \in \textsc{lex}_{\mathscr{H}}$, and conversely. Also, for every SC $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n) \in G((P_1, F_1), \ldots, (P_n, F_n))$ for some rule $G \in \textsc{rules}_{\mathscr{G}}$ such that each (P_i, F_i) is in $\mathscr{L}(\mathscr{G})$, there is some SC $(\nu^i : (Q_i, R_i) \to Y : 1 \leq i \leq n) \in H((Q_1, R_1), \ldots, (Q_n, R_n))$ for some rule $H \in \textsc{rules}_{\mathscr{H}}$ such that each (Q_i, R_i) is in $\mathscr{L}(\mathscr{H})$, such that there is a bijection $\alpha : \{1, \ldots, n\} \to \{1, \ldots, n\}$, a collection of derivation isomorphisms $(f_i, \phi_i) : (P_i, F_i) \to (Q_{\alpha(i)}, R_{\alpha(i)})$, and isomorphism $k : Z \to Y$, such that $\nu^{\alpha(i)} \circ (f_i, \phi_i) = k \circ \mu^i$ for each i.

- $\mathscr{L}(\mathscr{G})$ and $\mathscr{L}(\mathscr{H})$ are equivalent.

Proof. $\Rightarrow$) Construct isomorphisms between the derivations by induction on the generation of the languages.

$\Leftarrow$) The objects with one state in each language are exactly the lexicons. Since for every derivation in one, there must be an isomorphic derivation in the other, then *a fortiori* for each lexical item in one there is an isomorphic one in the other. Now suppose that $((P_1, F_1), \ldots, (P_n, F_n))$ is a tuple of derivations in $\mathscr{L}(\mathscr{G})$, and that there is some SC $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n) \in G((P_1, F_1), \ldots, (P_n, F_n))$ for some rule $G \in \textsc{rules}_{\mathscr{G}}$. Then, $\mathrm{ext}(\mu)$ is in $\mathscr{L}(\mathscr{G})$, and by assumption of equivalence, there is some isomorphic derivation (Q, R) in $\mathscr{L}(\mathscr{H})$. By the inductive construction of $\mathscr{L}(\mathscr{H})$, (Q, R) must have been constructed from derivations isomorphic to the trees (Q_i, R_i) which are components of the forest obtained by removing the final stage of (Q, R). The isomorphism between $\mathrm{ext}(\mu)$ and (Q, R) leads to a bijection between the (P_i, F_i) and (Q_i, R_i) such that the derivations in correspondence under this bijection are isomorphic. Moreover, there must be some SC $(\nu^i : (Q_i, R_i) \to Y : 1 \leq i \leq n) \in H((Q_1, R_1), \ldots, (Q_n, R_n))$ for some rule $H \in \textsc{rules}_{\mathscr{H}}$ such that (Q_i, R_i) extended along ν produce (Q, R), giving the requisite isomorphisms between SCs. We could argue conversely starting with a SC in $\mathscr{L}(\mathscr{H})$. □

Notice at the level of an entire *language*, we do not care which rule an SC originated from nor which position each operand goes into in the tuple of operands. Rather, we just care that given a SC used in one language, there is an isomorphic tuple, possibly rearranged, such that an isomorphic SC is possible on them in the other language. This does not mean that at the level of *grammars* or *rules* we cannot tell these apart. We could of course describe more rigid relationships requiring the rules to be in a bijection such that corresponding rules have a more strict relationship between them. For example, one way to describe isomorphism at the level of grammar would not only require a bijection putting in correspondence isomorphic lexical items, but also a bijection between rule-sets, such that for each pair of rules G and H in correspondence, it must be that for each SC $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n) \in G((P_1, F_1), \ldots, (P_n, F_n))$ there is a SC $(\nu^i : (Q_i, R_i) \to Y : 1 \leq i \leq n) \in H((Q_1, R_1), \ldots, (Q_n, R_n))$ such that there exist isomorphisms $(f_i, \phi_i) : (P_i, F_i) \to (Q_i, R_i)$ and $k : Z \to Y$ such that $\nu^i \circ (f_i, \phi_i) = k \circ \mu^i$ for each i.[10] Such a rigid notion of isomorphism would not be extensional, since it could in principle require that even SCs which are not used are in correspondence.

However, such a weak notion has many uses. For example, suppose we are given an n-ary rule G, and a partitioning of $\mathscr{D}(\mathbf{A})^n$, the category of n-tuples of derivations, into i equivalence classes. We could split the rule G into i different rules, where G_i is only defined on the i^{th} equivalence class, but takes exactly the same values as G where defined. Two grammars on the same lexical items, one

[10] We return to a more subtle notion of an equivalence between rules in §4.2.

containing the rule G and the other the rules G_i instead, should be viewed as equivalent, since overall they can produce the same derivations. Additionally, if a grammar contains 'redundant' rules G and H, such that each SC arising from H can already be produced using G, the language generated shouldn't be viewed as significantly different from the language generated by a grammar containing only G.

More simply, an equivalence of languages cares only about *structure* and not *cardinality*, even when languages are finite. Consider the following two languages presented in Fig. 3.16 (where the objects may be typed to only lead to the combinations in the figure). These two languages cannot be isomorphic since they don't even contain the same number of derivations. However, they are equivalent - for each derivation in one, there is an isomorphic derivation in the other, and conversely. This has practical benefits. Suppose for a moment that all grammatical rules are *replete*, in that for any rule G and SC $(\mu^i : (P_i, F_i) \to Z : 1 \leq i \leq n) \in G((P_1, F_1), \ldots, (P_n, F_n))$, if there are isomorphisms $(f_i, \phi_i) : (P_i, F_i) \to (Q_i, R_i)$, then there is a structural change $(\nu^i : (Q_i, R_i) \to Y : 1 \leq i \leq n) \in G((Q_1, R_1), \ldots, (Q_n, R_n))$ which is isomorphic, in that there is an isomorphism $k : Z \to Y$ such that $\nu^i \circ (f_i, \phi_i) = k \circ \mu^i$. Suppose we have a grammar $\mathscr{G}$, and we add a lexical item to $\mathscr{G}$ which fits in the same frames as an existing lexical item in $\mathscr{G}$ to construct $\mathscr{G}'$. This again should not essentially change the structure of $\mathscr{G}$, and of course the languages generated by the two grammars will be equivalent, since every derivation in $\mathscr{L}(\mathscr{G}')$ will be isomorphic to one in $\mathscr{L}(\mathscr{G})$ under the isomorphisms replacing each occurrence of the new lexical item and its images under SCs with

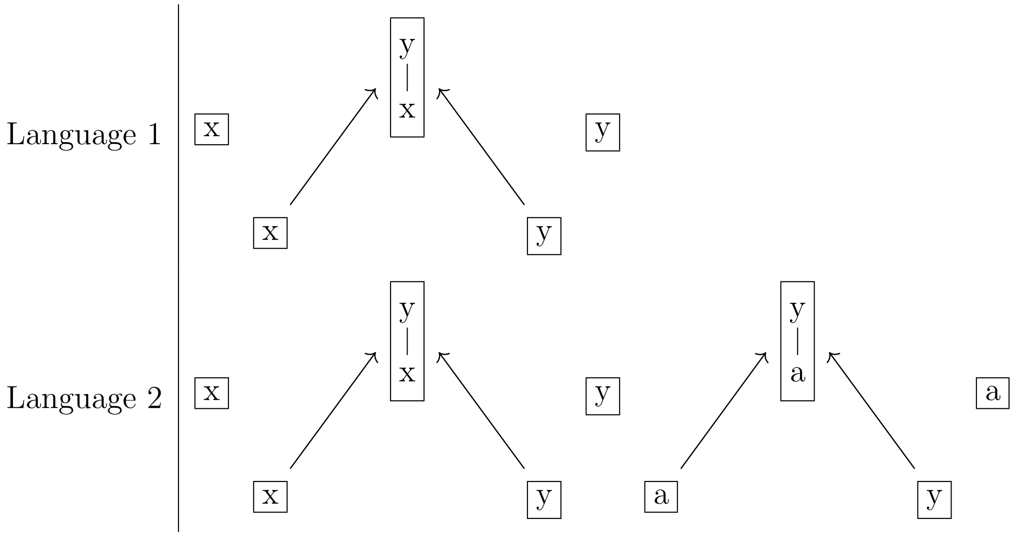


Figure 3.16: Two nonisomorphic equivalent languages.

the lexical item it was already isomorphic to in $\mathscr{G}$ and its images under SCs. If we add a different lexical item (or even multiple lexical items), but (each) isomorphic to the one we added to obtain $\mathscr{G}'$, to obtain $\mathscr{G}''$, we should not only have another equivalence between the languages generated by $\mathscr{G}''$ and $\mathscr{G}$, but also between those generated by $\mathscr{G}'$ and $\mathscr{G}''$. Since equivalences are only about the existence of isomorphic derivations and lexical items, it is again the case that simply adding new words to a grammar which are isomorphic to existing ones does not change the essential structure of the language generated.

3.4 Aside: Adjunctions and (Co)limits

Adjoint functors are one of the central unifying notions in category theory. A functor may have a left or right adjoint, which can be thought of as a 'best approximate inverse' from the left or right side. Adjoints will unify many constructions in this thesis, and make proving statements about certain constructions, such as yields, more straightforward.

Definition 3.4.1. An *adjunction* between two functors $L : \mathcal{C} \to \mathcal{D}$ and $R : \mathcal{D} \to \mathcal{C}$ is a pair of natural transformations $\eta : \mathbf{1}_{\mathcal{C}} \to RL$ and $\epsilon : LR \to \mathbf{1}_{\mathcal{D}}$, such that

- For any object C in $\mathcal{C}$, $\epsilon_{LC} \circ L(\eta_C) : LC \to LRLC \to LC$ is the identity

- For any object D in $\mathcal{D}$, $R(\epsilon_D) \circ \eta_{RD} : RD \to RLRD \to RD$ is the identity

These equalities are often represented by following diagrams:

$$\begin{tikzcd} L \arrow[d, "L\eta"'] \arrow[dr, "1_L"] & \\ LRL \arrow[r, "\epsilon L"'] & L \end{tikzcd} \qquad \begin{tikzcd} R \arrow[d, "\eta R"'] \arrow[dr, "1_R"] & \\ RLR \arrow[r, "R\epsilon"'] & R \end{tikzcd}$$

The natural transformations η and ϵ are called the *unit* and *counit*. When we have such natural transformations relating L and R, we say that L is *left adjoint* to R and that R is *right adjoint* to L, written $L \dashv R$. Given any morphism $f : LC \to D$ in $\mathcal{D}$, we can take its adjoint $R(f) \circ \eta_C : C \to RLC \to RD$; conversely, given any morphism $g : C \to RD$ in $\mathcal{C}$, we can take its adjoint $\epsilon_D \circ L(g) : LC \to LRD \to D$. These correspondences give a bijection $\mathcal{D}(LC, D) \approx \mathcal{C}(C, RD)$ for any C in $\mathcal{C}$ and D in $\mathcal{D}$. based on Mac Lane [26], IV and Borceux [23], Ch. 3

An important property of adjoints is that they determine each other uniquely up to natural isomorphism, if they exist. That is, if $F, F' \dashv G$ are both left adjoint to G, then F and F' are naturally isomorphic. Conversely, if $F \dashv G, G'$ are both right adjoint to F, then G and G' are naturally isomorphic. Also, the bijections $\mathcal{D}(LC, D) \approx \mathcal{C}(C, RD)$ between hom-sets determine the unit and counit: η_C can be recovered by taking the adjoint of $1_{LC} : LC \to LC$ for each C, and ϵ_D can be recovered by taking the adjoint of $1_{RD} : RD \to RD$ for each D.

We give examples of adjoint functors which have occurred already in this thesis.

- $s \dashv i$, where $i : \mathbf{Pos} \to \mathbf{Proset}$ is the inclusion of the category of partial orders into the category of preorders and $s : \mathbf{Proset} \to \mathbf{Pos}$ is the functor sending each preorder to its soberification. The unit $\eta : 1_{\mathbf{Proset}} \to is$ is the natural transformation with coordinates $\eta_P : P \to i(s(P))$ such that η_P is the order-preserving quotient function sending a preorder P to its soberification. The counit $\epsilon : si \to \mathbf{Pos}$ is a natural isomorphism, since the soberification of any partial order, considered as a preorder, is already a partial order.

- $\delta \dashv U$, where $U : \mathbf{FPos} \to \mathbf{FSet}$ is the forgetful functor mapping a finite partial order $(X, \leq_X)$ to its underlying set X and $\delta : \mathbf{FSet} \to \mathbf{FPos}$ is the 'discrete' functor sending any finite set X to the partial order with only the reflexive relations such that $x \leq y$ if and only if $x = y$. The unit $\eta : 1_{\mathbf{FSet}} \to U\delta$ is a natural isomorphism. The counit $\epsilon : \delta U \to 1_{\mathbf{FPos}}$ has coordinates $\epsilon_P : \delta(UP) \to P$ such that ϵ_P is underlying the identity function. It is order-preserving since any partial order must contain the relations of the form $p \leq p$.

- $\kappa \dashv \delta$, where $\delta : \mathbf{FSet} \to \mathbf{FPos}$ is the discrete functor and $\kappa : \mathbf{FPos} \to \mathbf{FSet}$ is the connected components functor. The unit $\eta : 1_{\mathbf{FPos}} \to \delta\kappa$ is the natural transformation with coordinates $\eta_P : P \to \delta(\kappa(P))$ mapping each element of x to the connected component it is contained in. That is, it is the quotient function associated to the equivalence relation 'is in the same component as'. The counit $\epsilon : \kappa\delta \to \mathbf{1_{FSet}}$ is a natural isomorphism.

- Let $\mathbf{Set} \times \mathbf{Set}$ be the category whose objects are pairs of sets (A, B) and whose morphisms are pairs of set functions $(f : A \to A', g : B \to B')$. Let $i : \mathbf{Set} \to \mathbf{Set} \times \mathbf{Set}$ be the functor which maps a set to its 'diagonal' $A \mapsto (A, A)$ and a function to its diagonal $f : A \to B \mapsto (f : A \to B, f : A \to B)$. This functor has a left adjoint $+ : \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ taking a pair (A, B) to its coproduct $A + B$ and a pair of morphism $(f : A \to A', g : B \to B')$ to $(f + g) : A + B \to A' + B'$ which maps $a \mapsto f(a)$ and $b \mapsto g(b)$. The unit $\eta : 1_{\mathbf{Set}\times\mathbf{Set}} \to i \circ +$ is the natural transformation with coordinates $\eta_{(A,B)} : (A, B) \to (A + B, A + B)$ which is the pair of coprojections (κ_A, κ_B). The counit $\epsilon : + \circ i \to 1_{\mathbf{Set}}$ has coordinates $\epsilon_A : A + A \to A$, such that ϵ_A maps each copy of A back to A.

This last example shows that coproducts can be computed as a kind of adjoint. This is in fact true in any category $\mathcal{C}$ with coproducts: the functor taking in a pair of objects (A, B) of $\mathcal{C}$ and returning $A+B$ can be viewed as a functor $+ : \mathcal{C}\times\mathcal{C} \to \mathcal{C}$ and it arises as the left adjoint to the inclusion $\mathcal{C} \to \mathcal{C}\times\mathcal{C}$ mapping $C \mapsto (C, C)$. The category $\mathcal{C}\times\mathcal{C}$ is isomorphic to the category $\mathbf{C}^{\mathbf{2}}$ whose objects are functors $F : \mathbf{2} \to \mathbf{C}$ and whose morphisms are natural transformations, where $\mathbf{2}$ is a category consisting of two objects $\{1, 2\}$ with only the identity morphisms.

A pushout can also be seen as arising as the left adjoint of a functor category. Let $\mathbf{J}$ be the category consisting of three objects $\{1, 2, 3\}$ together with the identity morphisms and two morphisms $x : 1 \to 2$ and $y : 1 \to 3$. A functor $F : \mathbf{J} \to \mathcal{C}$ is essentially a selection of three objects A, B, C together with morphisms $f : A \to B$ and $g : A \to C$. We have an inclusion functor $i : \mathcal{C} \to \mathcal{C}^{\mathbf{J}}$ mapping each object C to the diagram all of whose objects are C and whose morphisms are the identity on C.

An element of the set $\mathcal{C}^{\mathbf{J}}(F, i(X))$ is determined by a pair of maps $k : B \to X$ and $h : C \to X$ such that $kf = hg$. If i has a left adjoint L, then such pairs of maps are in a correspondence with maps $\mathcal{C}^{\mathbf{J}}(LF, X)$. But, we already know that if $\mathcal{C}$ has pushouts, then the functor taking a diagram F to its pushout LF has this property. So $\mathcal{C}^{\mathbf{J}}$ can be viewed as the category of diagrams of shape $(x : 1 \to 2,\ y : 1 \to 3)$ and the functor L takes each such diagram to its pushout.

Similarly, consider the empty category $\mathbf{E}$ with no objects or morphisms. $\mathcal{C}^{\mathbf{E}}$ is the category consisting of just one object, the functor $F : \mathbf{E} \to \mathcal{C}$, and it can be thought of as the empty diagram in $\mathcal{C}$. There is again a functor $i : \mathcal{C} \to \mathcal{C}^{\mathbf{E}}$ taking every object to this element. This has a left adjoint if and only if $\mathcal{C}$ has an initial object, and $L : \mathcal{C}^{\mathbf{E}} \to \mathcal{C}$ is the functor sending the single object of $\mathcal{C}^{\mathbf{E}}$ to this initial element.

More generally, for any category $\mathbf{J}$ whose total collection of morphisms is a set, we can describe the category $\mathcal{C}^{\mathbf{J}}$ whose objects are functors $F : \mathbf{J} \to \mathcal{C}$ and whose morphisms are natural transformations between them. For every such category we have a diagonal inclusion functor $i : \mathcal{C} \to \mathcal{C}^{\mathbf{J}}$ which sends each object C to the functor $i(C) : \mathbf{J} \to \mathcal{C}$ which returns C for each object j in $\mathbf{J}$ and returns the identity on C for each morphism of $\mathbf{J}$. We say that $\mathcal{C}$ *has colimits of type* $\mathbf{J}$ if i has a left adjoint. We say that $\mathcal{C}$ *has all finite colimits* if it has colimits of type $\mathbf{J}$ for any category $\mathbf{J}$ whose collection of morphisms is finite. While it may sound like it is difficult to check if a category $\mathcal{C}$ has all finite colimits, this condition can actually be reduced to simple ones.

Claim 3.4.1. The following are equivalent.

- $\mathcal{C}$ has all finite colimits.
- $\mathcal{C}$ has pushouts and an initial object.

There is another very important basic colimit construction called a *coequalizer*. Coequalizers are colimits of diagrams of the shape $1 \rightrightarrows 2$. We compute coequalizers in **Set**. Given two functions $f, g : A \rightrightarrows B$ in **Set**, the coequalizer is given by the quotient $q : B \to B/E$, where E is the equivalence relation generated by relations of the form $(f(a), g(a))$ for any $a \in A$. The equivalent statements in Claim. 3.4.1 are additionally equivalent to the following statement.

- $\mathcal{C}$ has all binary coproducts, coequalizers, and an initial object.

Summarizing, in **Set**, the initial object is the empty set, the coproduct is the disjoint union, and coequalizers are essentially quotients by the transitive closure of a relation. Colimits in many concrete categories can be thought of as abstractions of these. Another important property of left adjoints is that they *preserve colimits*. This means that given a diagram $F : \mathbf{J} \to \mathcal{C}$ and functor $L : \mathcal{C} \to \mathcal{D}$ which has a right adjoint, if F has a colimit K, then LK is a colimit of the diagram $LF : \mathbf{J} \to \mathcal{D}$ in $\mathcal{D}$. As an application of this fact, recall how we constructed pushouts in **Pos**. We can first compute the pushout in **Proset** and then apply s. This works since s is a left adjoint, and hence carries the pushout in **Proset** to the pushout in **Pos**.

We now rephrase yield functors as adjoints.

Claim 3.4.2. $i : \mathbf{A} \to \mathscr{D}(\mathbf{A})$ has a left adjoint $\top$ if and only if $\mathbf{A}$ has all finite colimits. If it exists, we call $\top$ the ***yield functor*** of $\mathscr{D}(\mathbf{A})$.

Proof. First note that the original definition of a yield functor is equivalent to the statement that $\top$ is left adjoint to i. Given a derivation $F : P \to \mathbf{A}$, the yield functor returns the colimit of this diagram. This is since if K is a colimit of F, then maps from $K \to A$ to any DSO A are in correspondence with morphisms $(P, F) \to A$ by the construction of a colimit. Then notice that we can draw all pushout diagrams and the empty diagram in $\mathbf{A}$ using derivations, hence $\mathbf{A}$ must have all pushouts and an initial object. We know that $\top$ carries sums to sums and that it is determined unique up to isomorphism since it is a left adjoint. □

Colimits have a natural dual called *limits*. Given a category $\mathbf{J}$ whose collection of morphisms is a set, we say that $\mathcal{C}$ *has all limits of type* $\mathbf{J}$ if the inclusion $i : \mathcal{C} \to \mathcal{C}^{\mathbf{J}}$ has a *right* adjoint. We give a simple example. A limit of type $\mathbf{2}$ is called a *product*. A product of a diagram selecting the pair (A, B) characterizes an object P together with maps $\pi_A : P \to A$ and $\pi_B : P \to B$, such that for any object Z with maps $f : Z \to A$ and $g : Z \to B$, there is a unique $u : Z \to P$ such that $\pi_A \circ u = f$ and $\pi_B \circ u = g$. In $\mathbf{Set}$, the Cartesian product $A \times B$ together with the projections $\pi_A : A \times B \to A$ mapping $(a, b) \mapsto a$ and $\pi_B : A \times B \to B$ mapping $(a, b) \mapsto b$ gives such a product. For any functions $f : Z \to A$ and $g : Z \to B$, there is a unique function $u : Z \to A \times B$ mapping $z \mapsto (f(z), g(z))$ such that $\pi_A \circ u = f$ and $\pi_B \circ u = g$. Summarizing, the left adjoint of $i : \mathbf{Set} \to \mathbf{Set^2}$ gives disjoint unions, while the right adjoint gives Cartesian products.

Consider the category $\mathbf{J}$ with morphisms $x : 1 \to 2$ and $y : 1 \to 3$. The left adjoint to $i : \mathcal{C} \to \mathcal{C}^{\mathbf{J}}$ gives pushouts. The right adjoint gives pullbacks, as used in §3.2.1 on grammatical relations.

Consider again any one-object category $*$, and the unique functor $i : \mathcal{C} \to *$ mapping all objects of $\mathbf{C}$ to the single object of $*$. This functor has a left adjoint only if $\mathcal{C}$ has an initial object. It has a right adjoint $R : * \to \mathbf{C}$ only if $\mathcal{C}$ has a *terminal object*. A terminal object $\mathbf{1}$ of $\mathcal{C}$ is an object such that every object X of $\mathcal{C}$ has exactly one morphism to $\mathbf{1}$. In **Set**, any singleton is a terminal object.

Consider again the inclusion $i : \mathcal{C} \to \mathcal{C}^{1 \rightrightarrows 2}$. This has a left adjoint only if $\mathcal{C}$ has all coequalizers. We say that it has all equalizers if it has a right adjoint. We compute equalizers in **Set**. Given a pair of functions $f, g : A \rightrightarrows B$, the equalizer can be computed as the subset inclusions $\{a \in A \text{ such that } f(a) = g(a)\} \hookrightarrow A$.

We again say that $\mathcal{C}$ *has all finite limits* if it has all limits of type $\mathbf{J}$ for every category $\mathbf{J}$ with a finite number of morphisms. We have the following result, dual to the one for finite colimits.

Claim 3.4.3. The following are equivalent.

- $\mathcal{C}$ has all finite limits.
- $\mathcal{C}$ has pullbacks and a terminal object.
- $\mathcal{C}$ has all binary products, equalizers, and a terminal object.

A dual property of right adjoints is that they *preserve limits*. This means that given a diagram $F : \mathbf{J} \to \mathcal{C}$ and functor $R : \mathcal{C} \to \mathcal{D}$ which has a left adjoint, if F has a limit K, then RK is a limit of the diagram $RF : \mathbf{J} \to \mathcal{D}$ in $\mathcal{D}$.

See, e.g., Awodey [33] or Mac Lane & Moerdijk [30] for proofs of these facts, as well as explicit constructions of general finite limits and colimits from simpler ones.

It is also useful to be able to define (co)limits in a more general setting without reference to their existence everywhere in the category. Given a diagram $F : \mathbf{J} \to \mathcal{C}$, we define a *cocone* on F to be a natural transformation $\mu : F \to i(C)$ for some object C of $\mathcal{C}$. This is essentially a map $\mu_j : F(j) \to C$ for each $j \in \mathbf{J}$ such that 'everything commutes'. We say that a cocone $\kappa : F \to i(C)$ is a *colimit* of F if for every cocone $\mu : F \to i(D)$, there is a unique $\mathcal{C}$-morphism $u : C \to D$ such that $\mu = i(u) \circ \kappa$. We call the maps $\kappa_j : F(j) \to C$ the *coprojections* to the colimit. The inclusion $i : \mathcal{C} \to \mathcal{C}^{\mathbf{J}}$ has a left adjoint if and only if every diagram F in $\mathcal{C}^{\mathbf{J}}$ has a colimiting cocone. Conversely, we call a natural transformation $\mu : i(C) \to F$ a *cone* on F. A cone $\pi : i(C) \to F$ on F is a *limit* of F if for every cone $\mu : i(D) \to F$ on F, there is a unique morphism $u : D \to C$ such that $\mu = \pi \circ i(u)$. We call the components $\pi_j : C \to F(j)$ the *projections*. i has a right adjoint if and only if for every diagram F it has a limiting cone. We can now state our definitions and results for preservation of (co)limits in this more general setting. A functor $U : \mathcal{C} \to \mathcal{D}$ is said to *preserve* (co)limits if for any diagram $F : \mathbf{J} \to \mathcal{C}$, if F has a (co)limit K, then UK is a (co)limit of UF. Left adjoints preserve all colimits that exist in $\mathcal{C}$, while right adjoints preserve all limits that exist in $\mathcal{C}$, even if $\mathcal{C}$ and $\mathcal{C}$ do not have all limits or colimits.

We give another example of an adjunction which occurred in the description of c-command. Let X be any finite partial order, and let $\mathcal{O}(X)$ be the set of open subsets of X. We turn $\mathcal{O}(X)$ into a category. For any open subsets $U, V \in \mathcal{O}(X)$, let $\mathcal{O}(X)(V, U)$ consist of exactly one morphism if and only if $V \subset U$. This category has binary products: for any two objects V and U, their product is $V \cap U$. We can construct a functor $U \cap - : \mathcal{O}(X) \to \mathcal{O}(X)$ for any object U, which maps $V \mapsto U \cap V$ and $W \subset V$ to $U \cap W \subset U \cap V$. This functor has a right adjoint $U \Rightarrow -$ which takes V to the largest open set W such that $U \cap W \subset V$. The counit of this adjunction is given by $U \cap (U \Rightarrow V) \subset V$. We can use this construction to define $U \Rightarrow V$ for any $U, V \in \mathcal{O}(X)$ to give the relative pseudo-complement operation of §2.5.2. Note that we can carry out an identical construction in **Set**. The product in this category takes a pair (A, B) to the set $A \times B$. For any set A, we can construct the functor $A \times - : \mathbf{Set} \to \mathbf{Set}$ mapping $B \mapsto A \times B$ and $f : B \to C$ to $1_A \times f : A \times B \to A \times C$. This functor has a right adjoint $(-)^A$ mapping $B \mapsto B^A = \{f : A \to B \text{ such that } f \text{ is a function}\}$ and $g : B \to C$ to $g^A : B^A \to C^A$ taking a function $f : A \to B$ to $g \circ f : A \to C$. The counit of this adjunction is given by function application $\epsilon_B : A \times B^A \to B$ mapping (a, f) to $f(a)$.

There is finally a very special case of adjunctions known as an *equivalence of categories*. An equivalence of categories between $\mathcal{C}$ and $\mathcal{D}$ is a pair of functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ together with an adjunction $\eta : \mathbf{1}_{\mathcal{C}} \to GF$ and $\epsilon : FG \to \mathbf{1}_{\mathcal{D}}$ such that each of these natural transformations is a natural isomorphism. In this case, we have $F \dashv G$ and $G \dashv F$. Given a functor $F : \mathcal{C} \to \mathcal{D}$, it is one half of an equivalence of categories if and only if: (1) F is full; (2) F is faithful; and (3) for each object D of $\mathcal{D}$, there is some C in $\mathcal{C}$ such that FC is isomorphic to D. If a functor F meets this last condition, we often say that it is *isomorphism-dense*. By construction, equivalent languages are equivalent categories.

3.5 Representably Concrete Derivations

Chapter 2 showed that categories (representable constructs) $\mathbf{A}$ make for good models of syntactic objects and structural changes between them. In §3.3, we constructed a category $\mathscr{D}(\mathbf{A})$ of derivations over $\mathbf{A}$ which are diagrams of objects from $\mathbf{A}$ connected by structural changes from $\mathbf{A}$. However, even when $\mathbf{A}$ is a representable construct, $\mathscr{D}(\mathbf{A})$ need not be. In this section, we will construct a much more restrictive category of derivations which forms a construct, and is much better behaved mathematically. We first illustrate how despite their intuitive construction, categories $\mathscr{D}(\mathbf{A})$ do not have many desirable properties. We then move onto an alternative but closely related model of derivations which does have many good concrete properties.

3.5.1 $\mathscr{D}(\mathbf{A})$ Need Not Be a Representable Construct

$\mathscr{D}(\mathbf{A})$ is rarely representably concrete, even when $\mathbf{A}$ is. For example, when $\mathbf{A} = \mathbf{Grph}$, there is an empty graph $\emptyset$, corresponding to a derivation with one state, and also a derivation $\emptyset \to \emptyset$ with two states. There are two distinct morphisms from the first to the second. If $\blacksquare$ is any derivation with all empty DSOs, then it will not even be able to tell apart nodes of a DSO G in a derivation with just one state, and hence it cannot represent a faithful functor.[11] However, if $\blacksquare$ has any nonempty DSO, then there are no maps from $\blacksquare$ to $\emptyset$ or $\emptyset \to \emptyset$, so both have empty set of points. But there is only one function from the empty set to itself, so this cannot represent a faithful functor either. Hence, there is no object representing a faithful functor to **Set**, so $\mathscr{D}(\mathbf{Grph})$ is not a representable construct. We can make similar arguments for many categories of derivations over representable constructs, such as $\mathscr{D}(\mathbf{FPos})$.

One of the main problems causing this is that totally empty objects are part of the structure of a derivation. Recall that when $\mathbf{A}$ is concretely representable by $\blacksquare$, the functor $\mathscr{D}(i(\blacksquare), -) : \mathscr{D}(\mathbf{A}) \to \mathbf{Set}$ takes a derivation (F, P) to its set of 'points' - the disjoint union of the points in each DSO F_p occurring in it. In this sense, empty stages aren't 'visible' from the set of points of the derivation.

We can consider the subcategory $\mathcal{C}$ of derivations with no empty stages and the functor $U : \mathcal{C} \to \mathbf{Set}$ taking each derivation to its set of points. This functor is faithful, so we can describe subderivations with respect to it. However, it is not possible to define a subderivation on an arbitrary subset. Consider the derivation in Fig. 3.17, call it (P, F). $\{\mathrm{the}', \mathrm{the}, \mathrm{dog}\}$ is a subset on this derivation. It is not possible to construct a subderivation on this subset. We cannot put *the′* and *dog* in the same step in the subderivation, since we will then be forced to map this stage

[11] Any maps $f, g : H \to G$ precomposed with the empty map $! : \emptyset \to H$ are the empty map $! : \emptyset \to G$. Since there are graphs with more than one vertex G, and hence multiple homomorphisms $x, y : \mathbf{1} \to G$, the functor represented by the empty graph is not faithful. We can embed **Grph** in $\mathscr{D}(\mathbf{Grph})$ and recreate the problem.

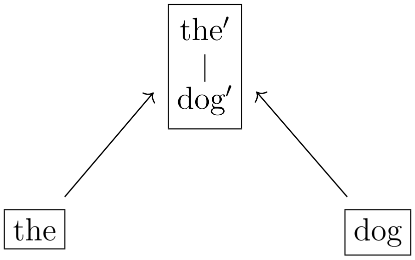


Figure 3.17: A derivation with no subderivation on $\{\text{the}', \text{the}, \text{dog}\}$.

both the to top stage of (P, F) and the lexical item *dog*. If they are to be in different stages in the subderivation, then we will be forced to construct a SC mapping *dog* to *the*′, but then the inclusion will not be a morphism since the component maps do not give a natural transformation. We can argue similarly to show that there is no possible subderivation structure on this subset no matter how we divide the points into stages. However, it is still possible to describe the derivational relationships between these points. For example, we can tell that *the* projects to *the*′, and that *the*′ depends derivationally on *dog*, but *dog* does not project to *the*′.

We will construct a category very closely related to $\mathscr{D}(\mathbf{FPos})$ which is a representable construct (which will imply it has no 'empty' stages), such that many concrete constructions on them are possible, such as subderivations on arbitrary subsets.

3.5.2 Representably Concrete Derivations of Finite Partial Orders

We construct a category very similar to $\mathscr{D}(\mathbf{FPos})$, except it is concretely representable and has many other good concrete properties. For any partial order $(X, \leq)$ and $x \in X$, let $U_x = \{y \in X \mid x \leq y\}$.

Definition 3.5.1. A ***derivation*** Δ consists of: (1) a set $|\Delta|$ of points; (2) a partial ordering $\leq$ on $|\Delta|$; and (3) for each point $x \in \Delta$, a partial order $\top_x$ and an order-preserving surjection $!_x : U_x \to \top_x$. The assignments in (3) must meet the condition: if $y \leq x$, then there exists an order-preserving function $f_{y,x} : \top_x \to \top_y$ such that $!_y \circ i_{y,x} = f_{y,x} \circ !_x$, where $i_{y,x} : U_x \hookrightarrow U_y$ is the subspace inclusion. If such a function exists, it is unique.

Claim 3.5.1. Define a ***morphism of derivations*** $\phi : \Delta \to \Gamma$ to be (1) a function $|\phi| : |\Delta| \to |\Gamma|$; (2) such that $a \leq_\Delta b$ implies that $\phi(a) \leq_\Gamma \phi(b)$; and (3) for every $x \in \Delta$, there must exist a (necessarily unique) order-preserving function $\phi_x : \top_x \to \top_{\phi(x)}$ such that $!_{\phi(x)} \circ \phi = \phi_x \circ !_x : U_x \to \top_{\phi(x)}$. The class of derivations with these morphisms forms a category **Der**.

There is an obvious faithful functor $\mathbf{Der} \to \mathscr{D}(\mathbf{FPos})$ mapping a derivation Δ to its underlying fposet $(|\Delta|, \leq_\Delta)$ with diagram taking x to $\top_x$, and an ordering $x \leq y$ to the map $f_{x,y} : \top_y \to \top_x$. However, the advantage of this category over $\mathscr{D}(\mathbf{FPos})$ is that the underlying function $|\phi|$ of a morphism ϕ by definition induces all of the relevant comparisons between DSOs, hence we have the following result.

Claim 3.5.2. The functor $|\cdot| : \mathbf{Der} \to \mathbf{Set}$ is faithful (hence **Der** is a construct). It is represented by $\mathbf{1}$, the derivation consisting of a singleton with the only possible derivation structure.

One natural question is what the embeddings of derivations are. In fact, for any subset $S \subset |\Delta|$, there is a unique derivation structure with underlying set S whose inclusion is an embedding, and all embeddings are isomorphic to one of this form. Even better, the ordering between states in the subderivation is just the ordering between them in Δ, and each derived object is associated to an embedding of finite partial orders.

Claim 3.5.3. Let $S \subset |\Delta|$ be any subset. For any $a, b \in S$, define $a \leq_S b$ iff $a \leq_\Delta b$. For any $a \in S$, define $\top^S_a = \{k \in \top^\Delta_a \mid (\exists x \in S) : (a \leq x) \text{ and } (!_a(x) = k)\}$, with the ordering $k \leq c$ in $\top^S_a$ iff the relation holds in $\top^\Delta_a$. Construct the map $!^S_a : \{x \in S \mid a \leq x\} = S_a \to \top^S_a$ sending each $x \in S_a$ to $!_a(x)$. This defines a derivation $\mathcal{S}$ with underlying set S, and the function $S \hookrightarrow |\Delta|$ underlies a morphism $i : \mathcal{S} \hookrightarrow \Delta$, and this morphism is an embedding. Furthermore, any embedding $j : \sigma \hookrightarrow \Delta$ is isomorphic to one of this form.

Notice one difference between **Der** and $\mathscr{D}$(**FPos**) is that every point of a **Der**-derivation corresponds to a 'stage'. We should then actually think of the DSOs $\top_x$ as (possibly total) fragments of DSOs or derivational steps. We will again have an inclusion of DSOs into derivations given by a functor i : **FPos** $\to$ **Der**. However, this map will associate a finite partial order A to a derivation with one point for every point of A.

Claim 3.5.4. The map i : **FPos** $\hookrightarrow$ **Der** (1) sending $(P, \leq) \mapsto (P, \leq_P, id)$ where for each $x \in P$, $id_x : U_x \to \top_x$ is an iso; and (2) sending $f : P \to Q$ to a derivation morphism acting as f on underlying points is a functor, and it is isomorphic to a full subcategory inclusion. This functor has a left adjoint $\top$: **Der** $\to$ **FPos**, and we denote $\top(\Delta)$ as $\top_\Delta$. When Δ has root r, then $\top_\Delta \cong \top_r$. Similarly, if Δ is the sum of rooted derivations $\Delta_1 + \ldots + \Delta_n$ with roots $r_1, \ldots r_n$, then $\top_\Delta \cong \top_{r_1} + \ldots \top_{r_n}$. We call this left adjoint the ***yield functor***. Furthermore, there is a forgetful functor $F : \mathbf{Der} \to \mathbf{FPos}$ sending a derivation $(\Delta, \leq_\Delta, \top_{(-)})$ to the underlying partial order $(\Delta, \leq_\Delta)$. F is right adjoint to i.

Summarizing, we have a string of adjoints $\top \dashv i \dashv F$. This string of adjoints is very similar to the string of adjoints $\kappa \dashv \delta \dashv U$, where $U : \mathbf{FPos} \to \mathbf{FSet}$ is the forgetful functor from finite partial orders to their underlying sets. In this way we can think of **Der** as putting 'order relations on partial orders', or more precisely, projection relations on partial orders. We make this precise.

Claim 3.5.5. Define ***projection*** as a relation on the points of Δ: $x \sqsubset y$ if and only if $f_{x,y} : \top_y \to \top_x$ maps the root of $\top_y$ to the root of $\top_x$. Any morphism $\phi : \Delta \to \Gamma$ preserves projection, in the sense that $x \sqsubset_\Delta y$ implies $\phi x \sqsubset_\Gamma \phi y$. For a subderivation $\mathcal{S} \hookrightarrow \Delta$, and points $x, y \in S$, $x \sqsubset_\Delta y$ additionally implies $x \sqsubset_{\mathcal{S}} y$.

The two functors $U : \mathbf{FPos} \to \mathbf{FSet}$ and $F : \mathbf{Der} \to \mathbf{FPos}$ are similar in that they both forget structure. i is similar to the discrete functor, in that for any DSO A, $i(A)$ is 'discrete' with respect to projection, as $x \sqsubset_\Delta y$ implies $x = y$. Finally, $\top$ is similar to κ, in that each point $\top_\Delta$ represents a 'component' with respect to projection - it is essentially the quotient of the underlying partial order of Δ by the relation 'is in the same projection component as'.

3.5.3 Concrete Properties of **Der**

We will describe some more concrete properties of morphisms of **Der**. We have a dual notion to that of monomorphisms (Defn. 2.3.6) called an *epimorphism*. A map $e : A \to B$ is an *epimorphism* if for any maps $f, g : B \rightrightarrows C$ such that $fe = ge$, then $f = g$. In other words, e is an epimorphism if it is *right-cancellable.* We have the following results.

Claim 3.5.6. Let $\phi : \Delta \to \Gamma$ be a morphism of derivations.

1. ϕ is an isomorphism iff it is (1) a bijection; such that (2) $a \leq_\Delta b$ iff $\phi a \leq_\Gamma \phi b$; (3) each induced $\phi_x : \top_x \to \top_{\phi x}$ is an isomorphism; (4) inducing isomorphisms of structural changes as in the diagram:
$$\begin{tikzcd}
\top_x \arrow[r, leftrightarrow, "\sim"] & \top_{\phi(x)} \\
\top_y \arrow[u, "!_{x,y}"] \arrow[r, leftrightarrow, "\sim"] & \top_{\phi(y)} \arrow[u, "!_{\phi x,\phi y}"']
\end{tikzcd}$$
2. ϕ is a monomorphism iff its underlying set function is an injection
3. ϕ is an epimorphism iff its underlying set function is a surjection

Claim 3.5.7. **Der** has all finite limits and colimits and they are preserved by the functor U : **Der** $\to$ **FPos** taking each derivation to its underlying partial order. Monomorphisms arising as equalizers coincide with subderivations, so the notion of subderivation can actually be stated internal to **Der**, without any reference to underlying sets. In particular, having all finite limits means that **Der** has

1. A ***terminal object* 1**

   The terminal objects are singletons $*$ together with the only possible derivation structure. Any derivation has exactly one morphism $! : \Delta \to \mathbf{1}$ sending all points to the single point of **1**.

2. For any pair of derivations (Δ, Γ), a ***product*** derivation $\Delta \times \Gamma$

   A product of Δ and Γ is given by the set $|\Delta| \times |\Gamma|$ together with the ordering $(d, g) \leq (d', g')$ iff $d \leq_\Delta d'$ and $g \leq_\Gamma g'$. For each point (d, g), $\top_{(d,g)} = \top_d \times \top_g = \{(k, c) \mid k \in \top_d, c \in \top_g\}$ with the ordering $(k, c) \leq (k', c')$ iff $k \leq k'$ in $\top_d$ and $c \leq c'$ in $\top_g$. Note that $U_{(d,g)} = U_d \times U_g$, and we give the map $!_{(d,g)} : U_d \times U_g \to \top_d \times \top_g$ by $!_{(d,g)}(a, b) = (!_d(a), !_g(b))$. This gives a derivation $\Delta \times \Gamma$. The projection functions $\pi_\Delta : \Delta \times \Gamma \to \Delta$ and $\pi_\Gamma : \Delta \times \Gamma \to \Gamma$ sending $(d, g) \mapsto d$ and $(d, g) \mapsto g$, respectively, are morphisms, and turn $\Delta \times \Gamma$ into a product.

3. For any pair of morphisms $a, b : \Delta \rightrightarrows \Gamma$, an equalizer $s : \mathcal{S} \to \Delta$

   The equalizer $\mathcal{S}$ is constructed as the subderivation on $S = \{x \in \Delta \mid a(x) = b(x)\}$, with s the associated embedding. Conversely, every subderivation embedding arises as an equalizer.

while having all finite colimits means that **Der** has

1. An ***initial object*** $\mathbf{0}$

   The initial derivation is given by the empty set $\emptyset$ together with the only possible derivation structure. For any derivation Δ, there is exactly one morphism $! : \mathbf{0} \to \Delta$ given by the empty function.

2. For any pair of derivations Δ, Γ a ***coproduct*** or ***sum*** derivation $\Delta + \Gamma$

   The sum is given by the disjoint union $|\Delta| + |\Gamma|$ together with the coproduct ordering. A point $x \in |\Delta| + |\Gamma|$ corresponds to exactly one element of either

Δ or Γ, and we associate $\top_x$ with the derived object in Δ or Γ. The subset inclusions into the disjoint union $\Delta, \Gamma \hookrightarrow \Delta + \Gamma$ give coproduct inclusions.

3. For any pair of morphisms $a, b : \Delta \rightrightarrows \Gamma$ a ***coequalizer*** $c : \Gamma \to \tilde{\Gamma}$. The underlying set can be computed by first taking the underlying coequalizer of finite partial orders: construct the relation R on $|\Gamma|$ of pairs $(a(d), b(d))$ for each $d \in |\Delta|$, and construct the quotient $q : |\Gamma| \to |\Gamma|/E$ by the transitive closure of this relation. $|\Gamma|/E$ inherits order relations from $(|\Gamma|, \leq_\Gamma)$ of the form $q(g) \leq q(g')$ whenever $g \leq g'$ in Γ. We take the smallest preorder containing these relations to turn $|\Gamma|/E$ into a preorder, such that q is an order-preserving function. We then take the soberification $s(|\Gamma|/E)$ of this preorder, obtaining an order-preserving function $s(q) : |\Gamma| \to s(|\Gamma|/E)$ between partial orders. We define the underlying partial order of $\tilde{\Gamma}$ to be $s(|\Gamma|/E)$. We turn this partial order into a derivation. We first construct a diagram of finite partial orders as follows: for each $x \in |\tilde{\Gamma}|$, we take the partial order $P = \{g \in \Gamma$ such that $x \leq s(q)(g)\}$ and the associated diagram $F : P \to \mathbf{FPos}$ mapping $F(g) = \top_g$ and $F(g \leq g') = f_{g,g'}$. We first define $\top_x$ to be the colimit of F. We hence have obtained a collection of partial orders $\top_x$ for each point $x \in \tilde{\Gamma}$. We want to simultaneously quotient every object in this collection by the relation '$g \sim g'$ if $s(q)(g) = s(q)(g')$'. On each $\top_x$, for each $g, g' \in \Gamma$ such that $s(q)(g) = s(q)(g')$ we add the relation $((\kappa_g \circ !_g)(g), (\kappa_{g'} \circ !_{g'})(g'))$, where $\kappa_g : \top_g \to \top_x$ and $\kappa_{g'} : \top_{g'} \to \top_x$ are the coprojections from the diagram F associated to x to the colimit. We then take the quotient of $\top_x$ by the equivalence relation generated

by this relation followed by soberification to obtain order-preserving functions $q_x : \top_x \to \tilde{\top}_x$. For each point $x \in \tilde{\Gamma}$, we define the function $!_x : U_x \to \tilde{\top}_x$. Given a point $y \in U_x$, choose any $g \in \Gamma$ mapping to it under $s(q)$, and construct the map $q_x \circ \kappa_g \circ !_g : U_g \to \top_g \to \top_x \to \tilde{\top}_x$. We define $!_x(y)$ to be the image of g under this map.

Factorization systems. We call a morphism $\phi : \Delta \to \Gamma$ a ***regular monomorphism*** if it arises as the equalizer of some pair of maps. We simplify the definitions in Adámek, et al. [25] for a $(\mathbf{Epi}, \mathbf{RegMon})$ factorization system.

Definition 3.5.2. Let $\mathcal{C}$ be any category in which the composite of any two regular monomorphisms is again a regular monomorphism. We say that $(\mathbf{Epi}, \mathbf{RegMon})$ is a ***factorization system*** for $\mathcal{C}$ if:

1. For any morphism $f : c \to d$ in $\mathcal{C}$, there is a factorization of $f = m \circ e$

$$\begin{tikzcd}
c \arrow[r, "f"] \arrow[d, two heads, "e"'] & d \\
k \arrow[ur, hook, "m"'] &
\end{tikzcd}$$

   as an epimorphism followed by a regular monomorphism.

2. For any epimorphism $e : a \to b$ and regular monomorphism $m : c \to d$ and pair of arbitrary morphisms $f : a \to c$ and $g : b \to d$ such that $ge = mf$, there is a unique *diagonal* $u : b \to c$ such that $ue = f$ and $mu = g$.

$$\begin{tikzcd}
a \arrow[r, "e"] \arrow[d, "f"'] & b \arrow[d, "g"] \arrow[dl, dotted, "u"'] \\
c \arrow[r, "m"'] & d
\end{tikzcd}$$

The above axioms will guarantee that a factorization is essentially unique. If we have any two epi-regular mono factorizations $a \xrightarrow{e} k \xrightarrow{m} b$ and $a \xrightarrow{e'} k' \xrightarrow{m'}$

b, then there is a unique isomorphism $u : k \xrightarrow{\sim} k'$ which is the diagonal of the associated square. We claim that $(\mathbf{Epi}, \mathbf{RegMon})$ is a factorization system for **Der**. Concretely, this will mean that every morphism factors as a surjection onto its image given the subderivation structure, followed by an embedding.

Claim 3.5.8. The classes **Epi** (equivalently, surjections) and **Emb** (equivalently, embeddings) form a factorization system for **Der**. That is, every morphism $\phi : \Delta \to \Gamma$ factors as an epimorphism followed by an embedding $\Delta \to \mathrm{im}(\phi) \hookrightarrow \Gamma$ up to unique isomorphism of embeddings, where $\mathrm{im}(\phi) = \{g \in \Gamma \mid (\exists d \in \Delta) : \phi(d) = g\}$.

We will prove all of these claims together.

Proof. That each of the constructions above produces the appropriate (co)limit can be checked by direct computation. Hence, **Der** has all finite (co)limits. That U preserves limits is automatic since U is a right adjoint; that it preserves colimits is given by the constructions.

An isomorphism $\phi : \Delta \to \Gamma$ must be a bijection since **Der** is a construct. To see that order-relations hold in Δ if and only if they hold in Γ is ensured by the fact that ϕ and its inverse must be order-preserving. That an isomorphism extends to an isomorphism between each DSO can be checked just be constructing the inverses at each point.

We now show that ϕ is a monomorphism if and only if it is injective. That a monomorphism ϕ has underlying injective function follows immediately from the fact that the forgetful functor is representable: suppose that $\phi : \Delta \to \Gamma$ is a monomorphism. The underlying set function is isomorphic to $\phi \circ - : \mathbf{Der}(\mathbf{1}, \Delta) \to \mathbf{Der}(\mathbf{1}, \Gamma)$.

If $x, y : \mathbf{1} \to \Delta$ are any two points of Δ and $\phi \circ x = \phi \circ y$, then $x = y$ since ϕ is a monomorphism. Now suppose that ϕ is injective. Let $f, g : \Xi \rightrightarrows \Delta$ be any two morphisms such that $\phi \circ f = \phi \circ g$. Suppose $f \neq g$. Then there is some point $x \in \Xi$ such that $f(x) \neq g(x)$. By assumption, $\phi(f(x)) = \phi(g(x))$. But since ϕ is injective, $f(x) = g(x)$, a contradiction. So $f(x) = g(x)$ for all points of Ξ, and hence $f = g$ since **Der** is a construct. So ϕ is a monomorphism.

To prove our claim about epimorphisms and image-factorizations, we need the following lemma, whose proof is in Appendix A.

Lemma 3.5.8.1. Let X be any finite partial order and $i : S \to X$ any injective order-preserving function considered as a subset, and take the pushout *of preorders* of i with itself to obtain $X +_S X$. Then, $X +_S X$ is already antisymmetric, and hence the forgetful functor from **FPos** to **Set** preserves this pushout, and $s(x) = t(x)$ if and only if $x \in S$.

We now prove that a morphism $\phi : \Delta \to \Gamma$ is epimorphic if and only if it is surjective. (Surj $\Rightarrow$ Epi) Suppose that ϕ is surjective. Let $a, b : \Gamma \rightrightarrows \Xi$ be any pair of morphisms such that $a\phi = b\phi$. For any point $g \in \Gamma$, we can find some $d \in \Delta$ such that $\phi(d) = g$ by surjectivity. Then $a\phi(d) = b\phi(d)$ implying that $a(g) = b(g)$ for all $g \in \Gamma$, so $a = b$. ($\neg$ Surj $\Rightarrow$ $\neg$ Epi) Suppose that ϕ is not surjective. Then there is some point $g \in \Gamma$ such that there is no $d \in \Delta$ where $\phi(d) = g$. Construct the subset $|\Gamma| - \{g\} \equiv S$. S can be turned into a subderivation with inclusion $i : S \to \Gamma$. We can take the pushout of this morphism with itself leading to morphisms $s, t : \Gamma \rightrightarrows \Gamma +_S \Gamma$. Since U preserves colimits, and by Lemma 3.5.8.1, these have the property that $s(x) = t(x)$ iff $x \in S$, i.e. iff $x \neq g$. Then $s\phi = t\phi$, since for all $d \in \Delta$, $\phi(d) \neq g$. But by construction, $s \neq t$, since $s(g) \neq t(g)$.

We now show that a morphism is an equalizer of some pair of morphisms if and only if it is a subderivation embedding. (Reg $\Rightarrow$ Emb) If m is regular, then there exist morphisms $a, b : \Delta \rightrightarrows \Gamma$ such that m is their equalizer. We apply $|\cdot|$ to get a diagram of sets. Now suppose we have a derivation Σ and function $f : |\Sigma| \to |\mathcal{S}|$ such that $|m|f$ underlies a morphism.

$$\begin{tikzcd}
|\mathcal{S}| \arrow[r, "|m|"] & |\Delta| \arrow[r, shift left, "|a|"] \arrow[r, shift right, "|b|"'] & |\Gamma| \\
|\Sigma| \arrow[u, "f"] \arrow[ur, "|m|f"'] & &
\end{tikzcd}$$

Now, $|a|(|m|f) = (|a||m|)f$ and $|b|(|m|f) = (|b||m|)f$. But $|a||m| = |b||m|$, so $|a|(|m|f) = |b|(|m|f)$, and $|m|f$ is a morphism equalizing a and b. So there must be a unique morphism $u : \Sigma \to \mathcal{S}$ such that $mu = |m|f$. We apply the $|\cdot|$ functor to obtain the equality $|m||u| = |m|f$ of set functions. But $|m|$ is injective, so $|u| = f$, showing that f underlies a morphism. (Emb $\Rightarrow$ Reg) Let $m : \mathcal{S} \to \Delta$ be an embedding. We construct the pushout of m along itself to obtain morphisms $s, t : \Delta \rightrightarrows \Delta +_{\mathcal{S}} \Delta$. By Lemma 3.5.8.1, $s(d) = t(d)$ iff $d \in \mathcal{S}$. Let Σ be any derivation and $h : \Sigma \to \Delta$ any morphism whose set-theoretic image fits in $\mathcal{S}$. h has image inside $\mathcal{S}$ iff $sh = th$, i.e. if it equalizes s and t. Since $\mathcal{S}$ is an embedding, h factors as a morphism $g : \Sigma \to \mathcal{S}$ such that $mg = h$. This shows that m equalizes s and t, and that it is universal with respect to equalizing s and t.

We finally prove factorization. Let $\phi : \Delta \to \Gamma$ be any morphism. Denote by $\mathrm{im}(\phi)$ the set of points $g \in \Gamma$ such that there is some $d \in \Delta$ where $\phi(d) = g$. As a subset of Γ, we can give this the unique subderivation structure. By the universal property of embeddings, ϕ will factor as a set-function through $\mathrm{im}(\phi)$, and hence must factor through it as a morphism. This morphism is clearly surjective. So, every morphism factors as an epimorphism followed by a regular monomorphism. Now, let $e : \Delta \to \Gamma$ be any epimorphism (surjection), and let $m : \Xi \to \Pi$ be any regular monomorphism, and let $f : \Delta \to \Xi$ and $g : \Gamma \to \Pi$ be any morphisms such that $ge = mf$. We construct the requisite unique diagonal $d : \Gamma \to \Xi$ by a diagram chase. Choose any element $b \in \Gamma$. Since e is epimorphic, it is surjective, so we can find some element $a \in \Delta$ such that $e(a) = b$. We tentatively assign b to $f(a)$, and we must make sure that this is well-defined. Fixing $b \in \Gamma$, choose any two elements $a, a' \in \Delta$ such that $e(a) = e(a') = b$. Clearly, $g(e(a)) = g(e(a'))$ and hence $m(f(a)) = m(f(a'))$. But since m is injective, $f(a) = f(a')$. So choice of a mapping to b does not matter, and the function $d : |\Gamma| \to |\Xi|$ mapping b to $f(a)$ for any a such that $e(a) = b$ is well-defined. By construction, for any element $a \in \Delta$, $d(e(a)) = f(a)$. For any $b \in \Gamma$, $d(b)$ is equal to $f(a)$ for any a mapping to b. Choose any such a, and notice that $g(b) = g(e(a)) = m(f(a)) = m(d(b))$. So $de = f$ and $g = md$. We must now only prove that d is a morphism. But $g = md$, and g is a morphism. Since m is an embedding, d must be a morphism. □

The constructions are all well-behaved with respect to projection. That is, $(a, b) \sqsubset (a', b')$ in $\Delta \times \Gamma$ if and only if $a \sqsubset a'$ and $b \sqsubset b'$; $a \sqsubset b$ in a subderivation $\mathcal{S} \hookrightarrow \Delta$ if and only if the relation holds in Δ; $x \sqsubset b$ in $\Delta + \Gamma$ if and only if the projection relation holds in one of the summands.

3.5.4 Constituency in **Der**

We have seen many nice properties of **Der** as a representably concrete category: isomorphisms are special bijections; embeddings special injections; coproducts 'structured disjoint unions' and products 'structured cartesian products' of underlying sets; surjections and embeddings give a factorization system for **Der**-morphisms compatible with a set-function's image-factorization.

The method for 'concretizing' derivations was giving mappings from the underlying state-space to the DSOs which have to be surjective (every point in a DSOs must correspond to some point in the underlying state-space). However, the cost of this is that the partial orders underlying derivations often have many more points than usually associated with 'derivation trees', and are rarely trees. We remarked that this is one major difference between **Der** and $\mathscr{D}(\mathbf{FPos})$ - every point $x \in \Delta$ of a derivation corresponds to a DSO $\top_x$. We might then not want to think of every associated DSO $\top_x$ as a step of a derivation, but possibly only a fragment of one. We would then like to group the points of a derivation together such that we can think of points in one cluster as being in the same 'step'.

Consider the derivation given in Fig. 3.18. Let $X = \{1, 2, 3, 4\}$ together with the partial ordering on the left in Fig. 3.18. The maps $!_1 : U_1 \to \top_1$ and $!_2 : U_2 \to \top_2$ must be isomorphisms since they must be surjective. We let $!_3 : U_3 \to \top_3$ be the map sending both 3 and 2 to a single point. Finally, $!_4 : U_4 \to \top_4$ is the map identifying 4 with 1 and 3 with 2, such that $[4 = 1] \leq [3 = 2]$. We can think of this as associated to the object of $\mathscr{D}(\mathbf{FPos})$ on the right in Fig. 3.18. We intend to

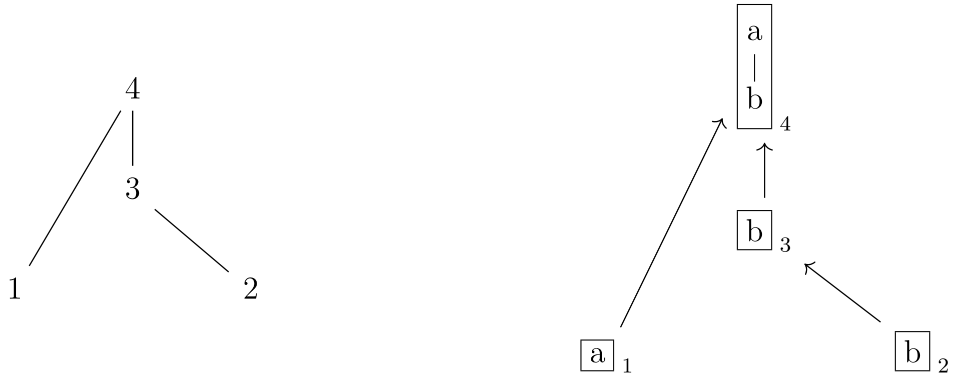


Figure 3.18: Partial order underlying a derivation and its DSOs and structural changes

group the objects $\top_3$ and $\top_4$ together, viewing them as being DSOs associated to the 'same step'. This step should be distinct from the steps associated to DSOs $\top_1$ and $\top_2$, which are also distinct from each other. We could extrinsically specify a partition on the set of points to give this information. However, when the intended DSOs are all connected, we can use the existing structure to define a relation which is often a partition of the appropriate form.

Definition 3.5.3. We say that y ***is close to*** x, written $x \preceq y$, if $x \leq y$, and for all $z \in U_x$ such that $!_x(z) =!_x(y)$, we have $y \leq z$. We say that a morphism $\phi : \Delta \to \Gamma$ is ***coherent*** if $x \preceq y$ implies that $\phi x \preceq \phi y$. We say that Δ is ***transitive*** if $\preceq$ is transitive.

Essentially, $x \preceq y$ are close if the y dominates all points projecting to $!_x(y) \in \top_x$. If Δ is transitive, then closeness is a partial ordering on $|\Delta|$. In this case, we denote the set of connected components with respect to $\preceq$ as $T(\Delta)$. For any transitive derivation, there is a function $q_\Delta : |\Delta| \to T(\Delta)$ taking a point x to the $\preceq$-component it belongs to. The ordering $\leq$ on $|\Delta|$ induces a preordering on $T(\Delta)$:

if $x \leq y$ in Δ, then we declare $q_\Delta(x) \leq q_\Delta(y)$ on $T(\Delta)$, and we take the smallest preordering on $T(\Delta)$ containing these relations.

Definition 3.5.4. If Δ is transitive and the induced ordering on $T(\Delta)$ is a partial order, then we say that Δ is ***separated***. Let K be an element of $T(\Delta)$, ordered by $\preceq$. If K has a $\leq$-least element (root) t, we say that t is a ***term***. If Δ is separated and each component K of $T(\Delta)$ has a root, we say that Δ is ***separated by terms***.

Notice that the requirement that $q_\Delta : |\Delta| \to T(\Delta)$ induces a partial order on $T(\Delta)$ is identically the requirement that q_Δ is the quotient function associated to the equivalence relation 'is in the same $\preceq$-component as'. Therefore, if ϕ is coherent, then there is an induced order-preserving map $T(\phi) : T(\Delta) \to T(\Gamma)$ mapping a component K to the $\preceq$-component containing $\phi(x)$ for any $x \in K$. We now want to generalize the results of §2.5.2 about constituent-preserving maps to separated derivations. We first generalize open maps.

Definition 3.5.5. Call $\phi : \Delta \to \Gamma$ ***open*** if $|\phi| : (|\Delta|, \leq_\Delta) \to (|\Gamma|, \leq_\Gamma)$ is an open map.

Claim 3.5.9. Let $\phi : \Delta \to \Gamma$ be an open morphism between arbitrary derivations. Then the induced map $\top_x \to \top_{\phi x}$ is surjective for each $D \in \mathcal{O}(\Delta)$.

This follows immediately from the surjectivity of $\phi_x : U_x \to U_{\phi(x)}$ by openness. However, such maps do not have to preserve constituency 'on the nose'. We restrict them further to capture constituent-preserving maps, and show that these lead to traditional constituent-preserving maps between the spaces $T(\Delta)$ of states.

Definition 3.5.6. Let $\phi : \Delta \to \Gamma$ be a morphism between transitive derivations. We say that ϕ is ***constituent-preserving*** if (1) ϕ is coherent; (2) ϕ is an open morphism; (3) if $K \in T(\Delta)$, then the map $\phi : K \to C$ is surjective, where C is the component of Γ containing the image of K.

Claim 3.5.10. If $\phi : \Delta \to \Gamma$ is a constituent-preserving morphism between separated derivations, then $T(\phi) : T(\Delta) \to T(\Gamma)$ is an open map.

Proof. In Appendix A □

Hence, in the case when $T(\Delta)$ and $T(\Gamma)$ are trees, we will get a constituent-preserving map between them in the traditional sense. We give an example in Fig. 3.19.

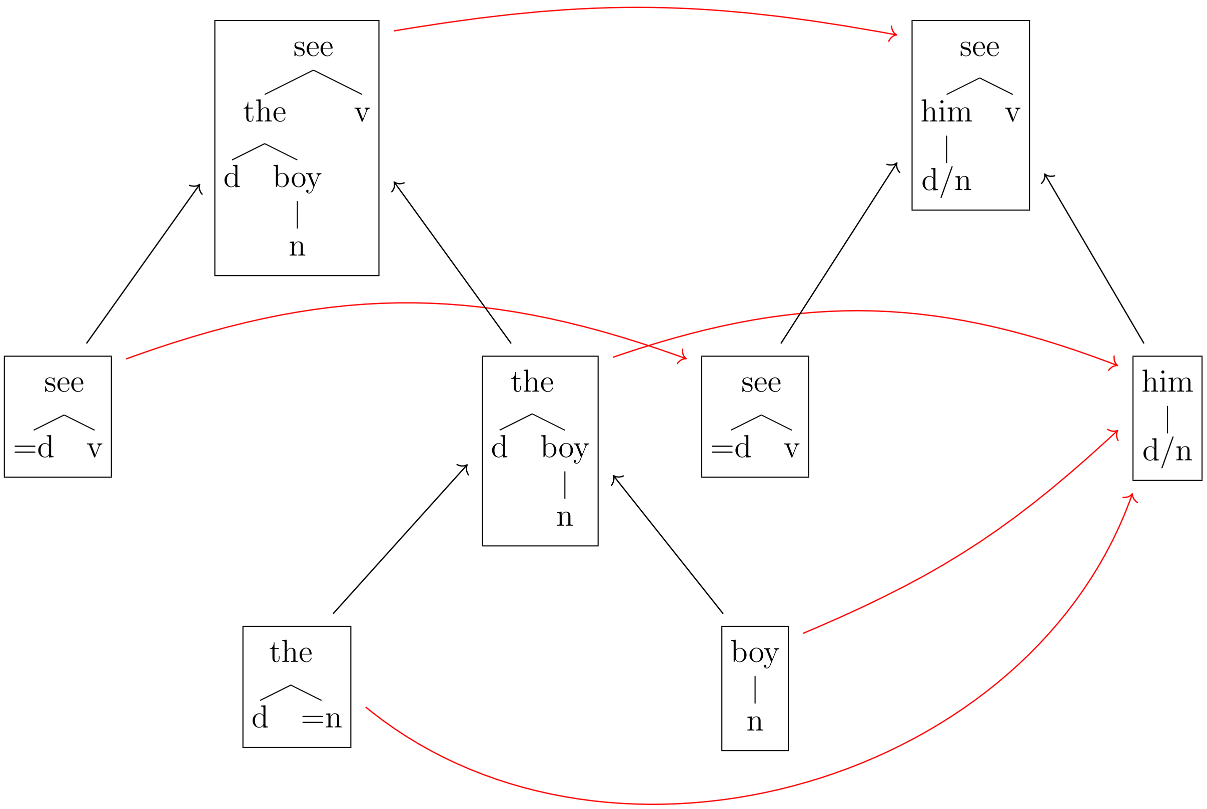
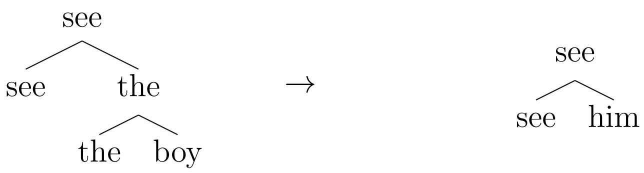


Figure 3.19: A constituent-preserving map between trees.

The quotient of the above map into terms gives an open map of (order-theoretic) trees. Each point in the quotient corresponds to an equivalence class represented by the elements in each box in the space of points. Since the map is open, it is constituent-preserving, and preserves c-command up to those in the image.



Closeness and separation (by terms) are well-behaved under products and sums.

Claim 3.5.11. $(a, b) \preceq (a', b')$ in $\Delta \times \Gamma$ if and only if $a \preceq a'$ and $b \preceq b'$. $x \preceq y$ in $\Delta + \Gamma$ if and only if $x \preceq y$ in Δ or $x \preceq y$ in Γ.

Claim 3.5.12. Let Δ and Γ be separated (by terms). Then $\Delta \times \Gamma$ and $\Delta + \Gamma$ are separated (by terms). Furthermore, $T(\Delta + \Gamma) = T(\Delta) + T(\Gamma)$ and $T(\Delta \times \Gamma) = \{K \times C \mid K \in T(\Delta), C \in T(\Gamma)\}$.

Proof. The first claim follows since dominance holds in the product if and only if it holds in each coordinate. The second claim follows upon noticing that the components of a product are the products of components, applying that observation to the $\preceq$ orderings on separated derivations. To see that products of derivations separated by terms are separated by terms, simply note that a term of a product (t, s) is a pair of terms since domination must hold in each coordinate. □

However, closeness is not well-behaved under arbitrary embeddings. We call a subderivation *coherent* if its embedding is a coherent morphism. There is an important class of coherent subderivations: those which correspond to a subobjects of some DSO.

Claim 3.5.13. Let $i : \mathcal{S} \hookrightarrow \Delta$ be any embedding. If $x, y \in \mathcal{S}$ and $x \preceq y$ in Δ, then $x \preceq y$.

Proof. Suppose that $!^{\mathcal{S}}_x(z) =!^{\mathcal{S}}_x(y) \in \top^{\mathcal{S}}_x$. Then $!_x(z) = !_x(y)$ since i is an embedding and $y \leq z$ in Δ. But then $y \leq z$ in $\mathcal{S}$, so $x \preceq y$ in $\mathcal{S}$. □

Claim 3.5.14. Let $\mathcal{K} \hookrightarrow \Delta$ be any coherent embedding. If Δ is transitive, then $\mathcal{K}$ is transitive. Let Δ be any derivation, $x \in \Delta$ any point, and $K \subset \top_x$ any subset. The subderivation inclusion $\mathcal{K} \hookrightarrow \Delta$ on the set $K^{-1} = \{y \in \Delta \mid (x \leq y) \text{ and } (!_x(y) \in K)\}$ is always coherent.

Proof. Transitivity in $\mathcal{K}$ follows from the fact that $x \preceq y$ in $\mathcal{S}$ if and only if in Δ when the embedding is coherent. Suppose that $a \preceq b$ in K^{-1}. If $z \in \Delta$ is such that $!_a(z) = !_a(b)$, then $!_x(z) = !_x(b)$, so $z \in K^{-1}$, and hence $b \leq z$ in Δ. So $a \preceq b$ holds in Δ, and the embedding is coherent. □

We now give some examples of subderivations and constituent-preserving maps. We give a base derivation in Fig. 3.20, where we have drawn boxes around the $\preceq$-components. We do not write all the derivational-dominance relations, but instead only the close ones. We use arrows to represent the morphisms between $\top_x$ where x is a term. For the derivations presented, for each term x, $\top_x$ is isomorphic to $\{y \text{ such that } x \preceq y\}$, so this depiction is unambiguous. The top derivation in Fig. 3.21 corresponds to a constituent-preserving embedding. The middle derivation is an open subderivation, but which is not constituent-preserving, since it includes 'fragments' of steps. It gives an example of a subderivation of the form K^{-1} for $K = \{\text{dog}, \text{red}, \text{N}', \phi\} \subset \top_\Delta$. Note that this subderivation corresponding to K represents that there was a fragment of the head *the* - specifically the selection feature $\{-\text{N}\}$ - which went into constructing K. Finally, the bottom derivation gives an example of an embedding which is not open, but still coherent. This more abstract kind of subderivation ignores certain steps, which means it must also take sub-DSOs

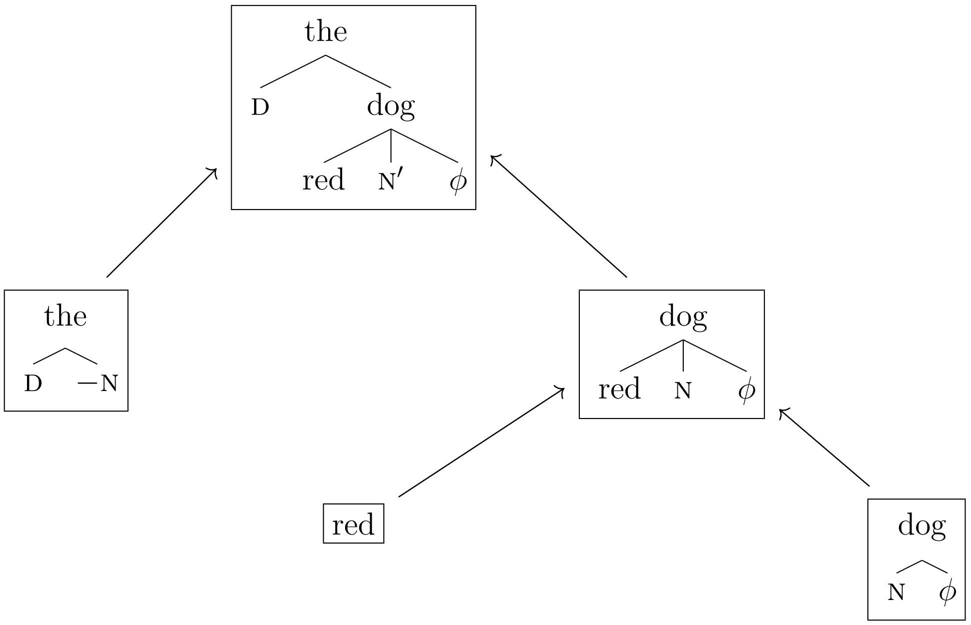


Figure 3.20: Informal picture of a derivation representing a DP.

at each corresponding step. However, allowing non-open subderivations in the theory is very helpful: it models, e.g., the fact that the derivation for *the dog* seems to be a substructure of that for *the red dog* in that similar operations were performed on analogous objects, and that similar grammatical relations were introduced between them.

We finally give an example of an incoherent subderivation. The subderivation on the subset {the′, the, dog} of the derivation depicted in Fig. 3.22 is not coherent. In the subderivation, we have a closeness relation the′ $\preceq$ dog; however, in the superderivation, this relation does not hold due to the intervening element dog′.

We finally describe an analogy of tree-like derivations, forest-like derivations, and factorization of the latter into the former. For any subset $S \subset |\Delta|$, we represent by (S) the set $\{x \in \Delta \text{ such that } \exists s \in S, s \leq x\}$.

Definition 3.5.7. Let Δ be separated (by terms). We say that Δ is a ***forest (of***

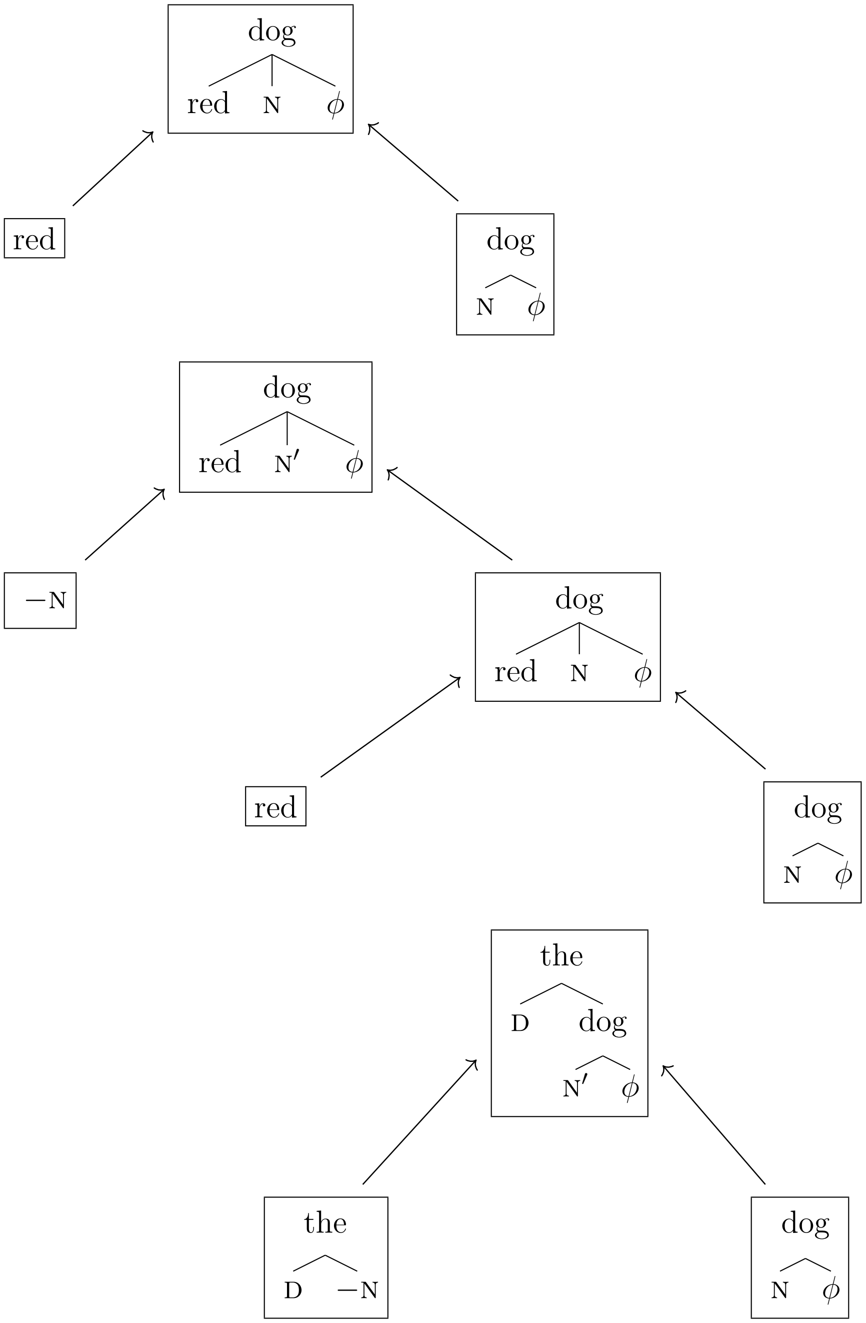


Figure 3.21: We give three examples of subderivations whose associated inclusions into the derivation in Fig. 3.20 are substructure embeddings in **Der**.

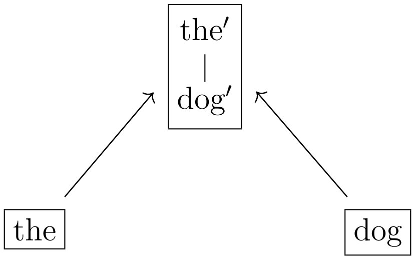


Figure 3.22: The subderivation structure on {the′, the, dog} is incoherent.

terms) if for any components $K, C \in T(\Delta)$, we have $(K) \cap (C) = (K), (C)$, or $\emptyset$. We say that a forest (of terms) is a ***tree (of terms)*** if there is a unique component $R \in T(\Delta)$ such that $(R) \cap (K) = (K)$ for any component $K \in T(\Delta)$.

Claim 3.5.15. If Δ is a forest, then $T(\Delta)$ is a forest. If Δ is a tree, then $T(\Delta)$ is a tree.

Claim 3.5.16. Let Δ be separated (by terms), and let $U \subset \Delta$ be an open subset such that $\Delta/U \hookrightarrow \Delta$ is a constituent-preserving embedding.[12] Then Δ/U is separated (by terms). Furthermore, if Δ is a forest (of terms), then so is Δ/U.

Proof. Since Δ/U is a coherent subderivation, we have $x \preceq y$ in Δ/U if and only if the relation holds in Δ, so Δ/U is transitive, and hence if x and y are in the same component in Δ/U, so are they in Δ. Suppose that $x \in K \in T(\Delta/U)$, and $x \in C \in T(\Delta)$. Then the inclusion $K \hookrightarrow C$ must actually be the identity since it must be surjective. So for any $x \in U$, if $x, y \in K \in T(\Delta)$, then $y \in U$. Blocks in Δ/U then correspond to blocks in Δ, and the ordering on $T(\Delta/U)$ is a partial order since it is a sub-order of $T(\Delta)$. If Δ is separated by terms, each $K \in T(\Delta/U)$ is also rooted by order-preservation and surjectivity on each component.

[12] Where Δ/U denotes the subderivation on U.

Now suppose Δ is a forest. Let $K, C \in T(\Delta/U)$, and suppose $(K) \cap (C) \neq$, and choose some point $x \in (K) \cap (C)$ from U. Then $(K) \cap (C) = (K)$ or (C) in Δ since it is a forest, and hence in Δ/U since it is open.

□

It is an immediate corollary that when Δ is a tree of terms, each constituent-preserving embedding $\Delta/U \hookrightarrow \Delta$ factors into trees of terms $\Delta/(K_i)$. Since each inclusion $\Delta/(K_i) \hookrightarrow \Delta/U$ and $\Delta/U \hookrightarrow \Delta$ are constituent-preserving embeddings, their composites are, and we get constituent-preserving embeddings $\Delta/(K_i) \hookrightarrow \Delta$ of trees of terms, viewed as 'derivational constituents' of Δ. In this way, each constituent-preserving embedding corresponds uniquely to a family of derivational constituents of Δ. A constituent-preserving embedding from a tree of terms into a tree of terms corresponds to all the points up to some term ('completed step') of Δ.

3.5.5 Extensions, Grammars, and Equivalences for **Der**

We generalize the results of §3.3.3 to construct derivations recursively. We again define an n-ary rule G as a mapping which takes in a tuple of derivations $(\Delta_1, \ldots, \Delta_n)$ and returns a set $G(\Delta_1, \ldots, \Delta_n)$ of SCs $(f_i : \Delta_i \to Z : 1 \leq i \leq n)$ such that no two are isomorphic. By the adjunction $\top \dashv i$, these again correspond to DSO morphisms $f_i : \top_{\Delta_i} \to Z$. We again characterize extensions using a universal property. However, we will have to be more subtle with concrete derivations, since for an operation $h : \Delta \to Z$, we will want to add points for every point of Z. We should add order relations $z \leq x$ for $z \in Z$ and $x \in \Delta$ if $z \leq_Z h(x)$. Each added order relation should correspond to structural updates determined by h. Finally, we should add these points in such a way that connected parts of Z are 'close' to each other, but not to any points of Δ.

Definition 3.5.8. Let $h : \Delta \to Z$ be any operation. Consider any derivation E and pair of morphisms $i : \Delta \to E$ and $j : Z \to E$, or equivalently, a single morphism $k : \Delta + Z \to E$. We say that k ***takes h-images to projection*** if for every $x \in \Delta$, $k(h(x)) \sqsubset k(x)$, that is, kx projects to $k(hx)$ in E.

Claim 3.5.17. For any operation $h : \Delta \to Z$, there is a universal derivation $\mathrm{ext}(h)$ together with a map $k : \Delta + Z \to \mathrm{ext}(h)$ which takes h-images to projection, in the sense that if $k' : \Delta + Z \to E$ is any morphism taking h-images to projection, then there is a unique derivation morphism $u : \mathrm{ext}(h) \to E$ make the following diagram commute.

$$\begin{tikzcd}
\Delta + Z \arrow[r, "k"] \arrow[dr, "k'"'] & \mathrm{ext}(h) \arrow[d, dashed, "u"] \\
& E
\end{tikzcd}$$

We can again build a functor from this construction from a comma category back to **Der**. For an operation $h : \Delta \to Z$, we will often denote the extension as Δ^h.

Definition 3.5.9. Define **Der/FPos** to be the comma category defined by the following equation.

$$1_{\mathbf{Der}} : \mathbf{Der} \to \mathbf{Der} \leftarrow \mathbf{FPos} : i$$

We call this the ***category of operations***.

Concretely, an object of **Der/FPos** is a triple $\langle \Delta, P, f : \Delta \to i(P) \rangle$, which we can unambiguously write as $f : \Delta \to P$. A morphism in this category is a pair $\langle \phi : \Delta \to \Gamma, h : P \to Q \rangle$ such that the following diagram commutes.

$$\begin{tikzcd}
\Delta \arrow[r, "f"] \arrow[d, "\phi"'] & P \arrow[d, "h"] \\
\Gamma \arrow[r, "g"'] & Q
\end{tikzcd}$$

Claim 3.5.18. The assignment ext : **Der/FPos** $\to$ **FPos** sending $h : \Delta \to Z$ to Δ^h extends to a functor by taking the induced maps between universal objects: $u \circ (\phi + f) : \Delta + Z \to \Gamma + Y \to \Gamma^g$ takes h-images to projection, so there is a unique $k : \Delta^h \to \Gamma^g$, which we write ϕ^f. This functor is determined uniquely up to natural iso.

We now show many ways in which this functor acts intuitively like an 'extension of Δ along h' on operations $h : \Delta \to Z$.

Claim 3.5.19. For any operation $f : \Delta \to Z$, consider the map to the extension $k : \Delta + Z \to \Delta^h$, or equivalently, two morphisms $i : \Delta \to \Delta^h$ and $j : Z \to \Delta^h$. We have the following properties:

(a) $i : \Delta \to \Delta^f$ is an open subderivation inclusion

(b) $j : Z \to \Delta^h$ is a subderivation embedding, and $Z \to \Delta^h \to \top_{\Delta^h}$ is an isomorphism (call it m), where the second map is yield$_{\Delta^h}$

(c) The composite $\Delta \to \Delta^h \to \top_{\Delta^h} \to Z$ is h, where the last map is the inverse m^{-1}.

A morphism $\langle \phi, h \rangle$ between $f : \Delta \to Z$ and $g : \Delta \to Z'$ is taken to a derivation morphism $\phi^h : \Delta^f \to \Gamma^g$ with the following properties:

(a) ϕ^h restricted to the subdervation Δ factors through $\Gamma \hookrightarrow \Gamma^g$ as ϕ:

$$\begin{tikzcd}
\Delta^f \arrow[r, "\phi^h"] & \Gamma^g \\
\Delta \arrow[u, hook] \arrow[r, "\phi"'] & \Gamma \arrow[u, hook]
\end{tikzcd}$$

(b) ϕ^h restricted to the subderivation Z factors through $Z' \hookrightarrow \Gamma$ as h:

$$\begin{tikzcd}
\Delta^f \arrow[r, "\phi^h"] & \Gamma^g \\
Z \arrow[u, hook] \arrow[r, "h"'] & Z' \arrow[u, hook]
\end{tikzcd}$$

(c) $\top_{\phi^h} : \top_{\Delta^f} \to \top_{\Gamma^g} \approx h : Z \to Z'$ are isomorphic morphisms of partial orders under the isomorphisms $\top_{\Delta^f} \approx Z$ and $\top_{\Gamma^f} \approx Z'$

It is straightforward to check all of these by direct construction of the extension, which follows a procedure very similar to the functor for derivations $\mathscr{D}(\mathbf{A})$. In the case when the codomain Z of an operation $h : \Delta \to Z$ is not connected, we cannot expect closeness to group Z into one step, since we have by definition considered a step a connected component of sorts. If we would like to allow such a distinction of a disconnected object as a single step, we must extrinsically state the partitioning of the points of the derivation. However, when Z is connected, and Δ is already separated, the extension is well-behaved. We can observe the following from direct construction.

Claim 3.5.20. If Δ is separated, and $h : \Delta \to Z$ is any morphism, then Δ^h is separated. If Δ is a forest and Z is connected, then Δ^h is a tree.

We can now give the definitions for a grammar and a generative construction of a language from it. A grammar $\mathscr{G}$ is a (typically finite) set of rules RULES together with a (typically finite) set of base items LEX of finite partial orders. We describe the language derived by $\mathscr{G}$ as the class of derivations $\mathscr{L}(\mathscr{G})$:

Definition 3.5.10. If $\mathscr{G}$ is a grammar, then we define a language $\mathscr{L}(\mathscr{G})$ recursively as follows.

1. If $X \in \text{LEX}$, then $X \in \mathscr{L}(\mathscr{G})$
2. If $(\Delta_1, \ldots, \Delta_n)$ is a tuple of derivations in $\mathscr{L}(\mathscr{G})$, $G \in \text{RULES}$, and $(h_i : \Delta_i \to Z : 1 \leq i \leq n) \in G(\Delta_1, \ldots, \Delta_n)$, writing the sum of these morphisms as $h : \Delta_1 + \ldots + \Delta_n \to Z$, then $(\Delta_1 + \ldots + \Delta_n)^h \in \mathscr{L}(\mathscr{G})$

We consider $\mathscr{L}(\mathscr{G})$ a category by taking it to be a full subcategory of **Der**. That is, the $\mathscr{L}(\mathscr{G})$-maps between a pair of objects Δ and Γ is just the set of **Der**-maps between them. Notice that if each of the lexical items is a connected space, and each Z in each structural change $(h_i : \Delta_i \to Z : 1 \leq i \leq n)$ is connected, then each derivation in $\mathscr{L}(\mathscr{G})$ is a tree. If each lexical item is rooted along with each Z in each SC, then each derivation in $\mathscr{L}(\mathscr{G})$ is a tree of terms. We can again describe an equivalence of languages as a pair of functors $F : \mathscr{L}(\mathscr{G}) \leftrightarrows \mathscr{L}(\mathscr{H}) : G$ such that $jF \approx iG$ are naturally isomorphic, where $i : \mathscr{L}(\mathscr{G}) \hookrightarrow \mathbf{Der}$ and $j : \mathscr{L}(\mathscr{H}) \hookrightarrow \mathbf{Der}$ are subcategory inclusions. When the languages consist only of trees, we have a result analogous to Claim 3.3.7 about equivalences of languages in terms of properties of isomorphisms of lexical items and SCs by the same reasoning.

Finally, we can consider grammatical relations in **Der**. One way to determine lexical items is positionally. For a separated derivation Δ, we call a step $L \in T(\Delta)$ a *lexical* *item* if it is maximal in $T(\Delta)$, i.e. for all $K \in T(\Delta)$, if $L \leq K$, then $L = K$. Suppose that $\mathscr{G}$ is a grammar with all rooted lexical items, with all SCs having a connected output. Then $\mathscr{L}(\mathscr{G})$ will consist of trees such that each lexical item is rooted. In such a case, in asking about the grammatical relations between lexical items L and M at some stage $K \in T(\Delta)$ such that $K \leq L$ and M, we can consider the net SCs $\top_{(L)} \to \top_{(K)}$ and $\top_{(M)} \to \top_{(K)}$. We will be interested in the case when $\overline{m} < \bar{l}$ is an immediate domination relation in $\top_{(K)}$, where l and m are the images of the roots of L and M in $\top_{(L)}$ and $\top_{(M)}$. We can then look at the pullbacks of $\top_{(L)} \to U_{\bar{l}} \hookrightarrow \top_{(K)}$ and $\top_{(M)} \to U_{\overline{m}} \hookrightarrow \top_{(K)}$ to compute the grammatical relation from the lP to the mP at the step K, as in §3.2.1.

3.5.6 Adding Structure to **Der**

We will want to generalize **Der** to allow derivations over DSOs with more structure than just the underlying dominance ordering. We will be interested in 'dominance partial orders with extra structure', which can be formalized as constructs $U : \mathbf{A} \to \mathbf{FPos}$ over **FPos**. We first construct **ADer** of derivations of **A**-objects, and then describe axioms which guarantee the ability to carry out recursive constructions with them.

Let $U : \mathbf{A} \to \mathbf{FPos}$ be any faithful functor. We construct **ADer** as the category whose objects are partial orders $(|\Delta|, \leq_\Delta)$ together with for each point $x \in \Delta$ an **A**-object $\top_x$, together with a surjective order-preserving function $U_x \to U(\top_x)$. Additionally, the induced order-preserving maps $f_{y,x} : U(\top_x) \to U(\top_y)$

must exist and underlie **A**-morphisms $\top_x \to \top_y$. A morphism of **ADer** is just like a morphism in **Der**, except the local maps $\phi_x : U(\top_x) \to U(\top_{\phi x})$ must underlie **A**-morphisms. Hence, **ADer** is also a construct. **ADer** always has coproducts, computed exactly like those in **Der**.

We must also be given an inclusion $i^{\mathbf{A}} : \mathbf{A} \to \mathbf{ADer}$. We will again want **A** to be representably concrete, and we will want the object representing its forgetful functor to be carried to a derivation representing the functor turning **ADer** into a construct. Formally, we ask that $\mathbf{A} \to \mathbf{FPos} \to \mathbf{Set}$ is representable (say by $\blacksquare$) such that $U(\blacksquare) \cong \mathbf{1}$, which will imply that $U : \mathbf{ADer} \to \mathbf{Der} \to \mathbf{Set}$ is representable by $i^{\mathbf{A}}\blacksquare$.

We can also state the universal properties for extensions in any such category. Given an operation $h : \Delta \to i^{\mathbf{A}}(Z)$ on an **ADer**, we can still describe the property of 'taking h-images to projection' for an **ADer** morphism $\phi : \Delta + Z \to \Gamma$ by looking at underlying derivations. We ask that such extensions exist, and if they do they again determine a functor $\mathrm{ext}^{\mathbf{A}} : \mathbf{ADer}/\mathbf{A} \to \mathbf{ADer}$.

We now axiomatize pairs $U^{\mathbf{A}} : \mathbf{A} \to \mathbf{FPos}$, $i^{\mathbf{A}} : \mathbf{A} \to \mathbf{ADer}$ which we will call *extendable* which allow recursive construction of languages from grammars using the same method.

- $U \circ U^{\mathbf{A}} : \mathbf{A} \to \mathbf{FPos} \to \mathbf{Set}$ is represented by an object $\blacksquare$ such that $U^{\mathbf{A}}(\blacksquare) \approx \mathbf{1}$. It is a general fact that if a functor is representable, it determines its representer up to isomorphism, so choice of representer does not matter.

- $i^{\mathbf{A}}$ has a left adjoint $\top^{\mathbf{A}}$.

- For every operation $h : \Delta \to i^{\mathbf{A}}(Z)$ in $\mathbf{ADer}/\mathbf{A}$, there is a universal map $\mathrm{ext}^{\mathbf{A}}_h : \Delta + Z \to \mathrm{ext}(h)$ taking h-images to projection. We write the functor that this determines as $\mathrm{ext}^{\mathbf{A}} : \mathbf{ADer}/\mathbf{A} \to \mathbf{ADer}$.

- (Ambiguously) writing U for all of the forgetful functors $U : \mathbf{A} \to \mathbf{FPos}$, $U : \mathbf{ADer} \to \mathbf{Der}$, and $U : \mathbf{ADer}/\mathbf{A} \to \mathbf{Der}/\mathbf{FPos}$, we have natural isomorphisms $U \circ \mathrm{ext}^{\mathbf{A}} \approx \mathrm{ext} \circ U$, $U \circ i^{\mathbf{A}} \approx U \circ i$, and $U \circ \top^{\mathbf{A}} \approx \top \circ U$.

These guarantee that (1) **ADer** is representably concrete; (2) the underlying finite partial order of the yield of an **A**-derivation Δ is the yield of its underlying derivation; (3) the extension of an **A**-derivation Δ by an **A**-operation h is the extension of the underlying derivation by the underlying operation. We can again describe SCs as morphisms and rules as assignments of SCs to tuples.

We give an example. Let **A** be the collection of finite sets X together with a dominance partial order $\leq_X$, a precedence preorder $\preceq_X$, and a collection of unary predicates λ_X, one for each element λ in a fixed set of labels $\mathbf{L}$. Morphisms of **A** are again $\leq$ and $\preceq$-preserving functions $f : X \to Y$ such that if $\lambda_X(x) = \text{true}$, then $\lambda_Y(f(x)) = \text{true}$. There is a functor $U^{\mathbf{A}} : \mathbf{A} \to \mathbf{FPos}$ mapping $(X, \leq_X, \preceq_X$ $, \{\lambda_X\}_{\lambda \in \mathbf{L}})$ to $(X, \leq_X)$, and a functor $i^{\mathbf{A}} : \mathbf{A} \to \mathbf{ADer}$ sending $(X, \leq, \preceq, \{\lambda_X\}_{\lambda \in \mathbf{L}})$ to the **A**-derivation with set of points X, derivational dominance relation $\leq$, and for each point $x \in X$, $\top_x$ the substructure on U_x, with $!_x : U_x \to U(\top_x)$ an isomorphism of partial orders. This pair of functors gives an extendable category of derivations.

3.6 Summary

We introduced the main contribution of this research, structural changes, modeled by tuples of morphisms of underlying DSOs. Projection and grammatical relation information can be recovered totally by looking at images of points under net SCs. We in fact showed that for dependency structures with features, we can recover traditional grammatical relations from a typology of connectivity information. This information was computed using pullbacks, and nonemptiness of certain pullbacks comparing the minimal and maximal projections of two lexical items produced a set of grammatical relations they can bear to each other which came with a natural ordering by ‘degree of connectivity’. We then presented two related models of derivations. For both, we constructed a yield functor which returns the derived object associated to a derivation, sums of derivations, and extensions of derivations by an operation. We also showed that yield functors arise as adjoints to natural inclusion functors. These allowed us to construct languages recursively from a grammar, where a grammar is a collection of basic DSOs together with a set of rules which assign SCs to tuples of derivations. We then used a special kind of equivalence of categories to characterize equivalences of languages. For special recursively constructed languages, an equivalence of languages gives the same information as a family of isomorphisms between lexical items and SCs provided by the rules. We described a special category of derivations of finite partial orders **Der**, and we showed that this category possesses many good mathematical properties considered as a ‘set with structure’. Specifically, it has sufficient structure to be able to characterize subderivations, products of derivations, quotients of derivations and many other basic constructions. Considered as ‘partial orders with structure’, we also generalized results about constituency to objects of **Der**.

Chapter 4: Rules

4.1 Overview

In formal models of grammar such as BPS [21, 34] and Minimalist Grammars [27], grammatical operations are given as functions which take in some tuple $(A_1, \ldots, A_n)$ of DSOs and return another DSO $f(A_1, \ldots, A_n) = Z$. However, we embellished such assignments with SCs from each A_i to Z. Just as embellishing derivations with SC data leads to a richer theory of grammatical relations, subderivations, isomorphisms of derivations and equivalences of languages, etc., embellishing rules with SC data leads to a richer theory of what structure they manipulate, when they are equivalent, and other interesting properties.

In traditional transformational grammars [1], grammatical operations were described via a Structural Analysis (SA), which described the properties of strings they targeted, along with a Structural Change (SC), which described string manipulations performed by the operation.[1] Also, given a specific string replacement rule,

[1] Chomsky [17] used a slightly different implementation. Strings in a set could be manipulated by movement, dropping, addition, or adjunction of elements. The transformations were restricted by giving an analysis K of the strings and restricting class Q of the kind of structural properties a string must have with respect to a constituent analysis. Various methods were given for assigning

one could generate all the *productions* associated to that rule - every instance of that replacement rule being applied in some context (those admissible by the SA). Conversely, given all the productions associated to a rule, it is often possible to recover basic SCs and the SA. We will develop a general theory for extracting out of any rule a SA and will characterize when a set of basic SCs generates all productions associated to that rule. Using basic generating SCs, we can describe formally comparisons between rules like 'induces the same SC in different contexts', such as how local and long-distance agreement both seem to involve a common SC, namely agreement. We will also be able to give a theory about *compilations* of SCs, which allow us to deconstruct a SC into component parts. For example, COMPLEMENT-MERGE may be deconstructed into a PHRASAL-ATTACHMENT component and a SELECTION component. Such deconstructions turn out to be more subtle than function composition, and allow us to unify properties of different syntactic relations, such as many of them involving PHRASAL-ATTACHMENT 'plus something else'. These give a much more general theory along the lines of Hunter [8].

4.2 Maximal Condition Categories (Structural Analysis)

The basic goal of this section is to formalize the following: given a rule G, what structural properties of the inputs does G care about? For example, a rule that attaches the root of one argument to the root of another cares about the roots of each. A rule that targets the closest feature of some type α with respect to some

the derived strings a constituent structure.

metric cares about the type α and some positional notion of closeness. A rule that adds precedence relations from a sequence $a_1 \preceq \ldots \preceq a_n$ to a sequence $b_1 \preceq \ldots \preceq b_k$ to build the sequence $a_1 \preceq \ldots \preceq a_n \preceq b_1 \preceq \ldots \preceq b_k$ cares about the last element of the first sequence and the first element of the second sequence. We will give a general method to answer this question for any assignment G.

The way we frame this question formally is asking for a given tuple of morphisms between DSOs or derivations where a rule is defined $f : (X_1, \ldots, X_n) \to (Y_1, \ldots, Y_n)$, does f preserve the structure that is relevant to this rule? For example, if the rule cares about the root of the i^{th} argument (because it is going to target it for manipulation in some way), both X_i and Y_i should have roots, and f_i should take the root of X_i to the root of Y_i - otherwise, f will have taken a piece of structure relevant for applying the rule to a context where the rule is no longer applicable. That is, we will be interested in determining if f preserves the 'conditions' which are relevant for determining whether a rule can apply.

We will develop the technique for the simplest case of rules first, involving only DSOs. Fix some category $\mathbf{A}$ of DSOs. We temporarily take an n-ary *rule* on $\mathbf{A}$ to be any assignment G which takes in n-tuples of objects of $\mathbf{A}$ and returns a *set* $G(A_1, \ldots, A_n)$ whose elements are objects Z of $\mathbf{A}$ together with a tuple of morphisms $f_i : A_i \to Z$, one for each $1 \leq i \leq n$. We again require that this set is not redundant, in that it only includes each SC once up to isomorphism. We allow this to be partially defined, in that it does not have to take in all possible n-tuples, and also multiply (non-deterministically) defined, in that it returns a set of SCs.

For any n-ary rule G, define $\mathcal{E}_G$ to be the collection of tuples of objects of

$\mathbf{A}$ where G is defined. We will turn $\mathcal{E}_G$ into a category which we will call the ***maximal condition category*** of G. The idea is that we will allow as many n-tuples of morphisms as possible between operands which 'preserve aspects of the structure relevant for determining how to apply G'. Given an SC $(f_i : A_i \to Z : 1 \leq i \leq n) \in G(A_1, \dots, A_n)$, any tuple $(B_1, \dots, B_n)$ in $\mathcal{E}$, and any tuple of $\mathbf{A}$-morphisms $k_i : A_i \to B_i$, we would like to describe how to 'translate' the f_i along the k_i to a SC on $(B_1, \dots, B_n)$. We can then ask if the resulting SC is in G. We will call a SC $(g_i : B_i \to Y : 1 \leq i \leq n)$ together with a morphism $j : Z \to Y$ the (n-ary) *pushout* of $(f_i : A_i \to Z : 1 \leq i \leq n)$ along $(k_1, \dots, k_n)$ if it has the following two properties: (1) for each f_i, we have $j \circ f_i = g_i \circ k_i$; (2) if $(g'_i : B_i \to Y' : 1 \leq i \leq n)$ and $j' : Z \to Y'$ are any collection of morphisms meeting condition (1), then there exists a *unique* $\mathbf{A}$-morphism $u : Y \to Y'$ such that $g'_i = u \circ g_i$ for each $1 \leq i \leq n$ and $j' = u \circ j$. A pushout, if it exists, is unique up to isomorphism: if $(g_i : B_i \to Y : 1 \leq i \leq n)$ and $(g'_i : B_i \to Y' : 1 \leq i \leq n)$ are both pushouts, then there is a unique isomorphism $u : Y \to Y'$ such that $g'_i = u \circ g_i$ and $j' = u \circ j$ by the second condition. See Fig. 4.1.

Given any n-tuple of $\mathbf{A}$-morphisms $(k_1, \dots, k_n) : (A_1, \dots, A_n) \to (B_1, \dots, B_n)$ between objects of $\mathcal{E}_G$, we can ask whether $(k_1, \dots, k_n)$ preserves the relevant aspects of the DSOs to evaluate the SA for G, in which case we call it ***condition-preserving***. We let $(k_1, \dots, k_n)$ be a morphism of $\mathcal{E}_G$ if for every SC $(f_i : A_i \to Z : 1 \leq i \leq n) \in G(A_1, \dots, A_n)$, the pushout of $(f_i : A_i \to Z : 1 \leq i \leq n)$ along $(k_1, \dots, k_n)$ is an element of $G(B_1, \dots, B_n)$. We give an example pushout in Fig. 4.2. We let $\mathbf{A}$ be the category of finite sets X with a dominance partial order,

$$\begin{tikzcd}
A_1 \arrow[d, "k_1"'] \arrow[rrrrr, bend left=30, "f_1"] & \cdots & A_i \arrow[d, "k_i"] \arrow[rrr, bend left=20, "f_i"] & \cdots & A_n \arrow[d, "k_n"] \arrow[r, "f_n"] & Z \arrow[d, dashed, "j"] \arrow[rdd, "j'"] & \\
B_1 \arrow[rrrrr, dashed, bend right=30, "g_1"'] \arrow[rrrrrrd, dotted, bend right=40, "g'_1"'] & \cdots & B_i \arrow[rrr, dashed, bend right=20, "g_i"'] \arrow[rrrrd, dotted, bend right=30, "g'_i"'] & \cdots & B_n \arrow[r, dashed, "g_n"'] \arrow[rrd, dotted, bend right=20, "g'_n"'] & Y \arrow[rd, "\exists ! u"'] & \\
 & & & & & & Y'
\end{tikzcd}$$

Figure 4.1: An n-ary pushout of a SC along a condition-preserving morphism

precedence preorder, and unary predicate λ_X for each λ in a fixed set of labels $\mathbf{L}$. We will intuitively describe why the diagram in Fig. 4.2 is a pushout. Fixing the objects in the upper left, upper right, and lower right corners, as well as (f_1, f_2) and (k_1, k_2), we want to compute the pushout. We call them (A_1, A_2), Z, and (B_1, B_2), respectively. Given any Y and maps $g_1 : B_1 \to Y$, $g_2 : B_2 \to Y$, and $j : Z \to Y$, the conditions $k_1 \circ g_1 = j \circ f_1$ and $k_2 \circ g_2 = j \circ f_2$ put restrictions on g_1, g_2, and Y. g_1 and g_2 must map *her* and *pet* to whatever j maps *the* and *drank* to. And since *pet* $\leq$ *her* in Z and j must preserve this dependency. So the commutivity condition requires that $g_2(\text{pet}) \leq g_1(\text{her})$. Similarly, g_1 and g_2 must send *parents* and *pet* to whatever j sends *detective* and *drank* to. And since *detective* $\preceq$ *drank* in Z, the commutivity condition requires that $g_1(\text{parents}) \preceq g_2(\text{pet})$. So the commutivity condition requires that g introduce certain dominance and precedence relations. The universality requirement of the pushout makes sure that no more is done then necessary to meet these conditions.

When $\mathbf{A}$ has coproducts, we can simplify this to the usual pushout as follows. Given an SC $(f_i : A_i \to Z : 1 \leq i \leq n)$, we can take the coproduct to obtain a single morphism $f : \coprod A_i \to Z$. For a tuple $A = (A_1, \ldots, A_n)$, denote $\overline{A} = A_1 + \ldots + A_n \equiv \coprod A_i$, and for a map $k = (k_1, \ldots, k_n)$, denote $\overline{k} = u$ for the unique map from the coproduct $\overline{A}$ to $\overline{B}$ induced by the maps $\kappa_i \circ k_i$, where $\kappa_i : B_i \to \overline{B}$ is the coprojection of the i^{th} coordinate into the coproduct $\overline{B}$. See Fig. 4.3. In this case, we can just take the usual pushout of f along $\overline{k}$ to obtain a morphism $g : \overline{B} \to Y$. Precomposing this morphism with each of the coprojections leads to a family of morphism $g_i = g \circ \kappa_i : B_i \to Z$.

It is not immediately obvious that these morphisms form a category: why should the composite $(k_1, \ldots, k_n) \circ (m_1, \ldots, m_n) = (k_1 \circ m_1, \ldots, k_n \circ m_n)$ be an $\mathcal{E}_G$ morphism? Intuitively, if m preserves the relevant conditions, and k preserves the relevant conditions, then their composite preserves the relevant conditions. Formally, the universal property of pushouts guarantees this, and so $\mathcal{E}_G$ is actually a category. We prove this using an important property of pushouts.

Claim 4.2.1. (Pushout Lemma) In any category, given the diagram in Fig. 4.4 such that the left square is a pushout, the right square is a pushout if and only if the outer rectangle is a pushout.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "k"'] & B \arrow[r, "g"] \arrow[d, "j"'] & C \arrow[d, "c"] \\
D \arrow[r, "a"'] & E \arrow[r, "b"'] & F
\end{tikzcd}$$

Figure 4.4: Pasting pushouts

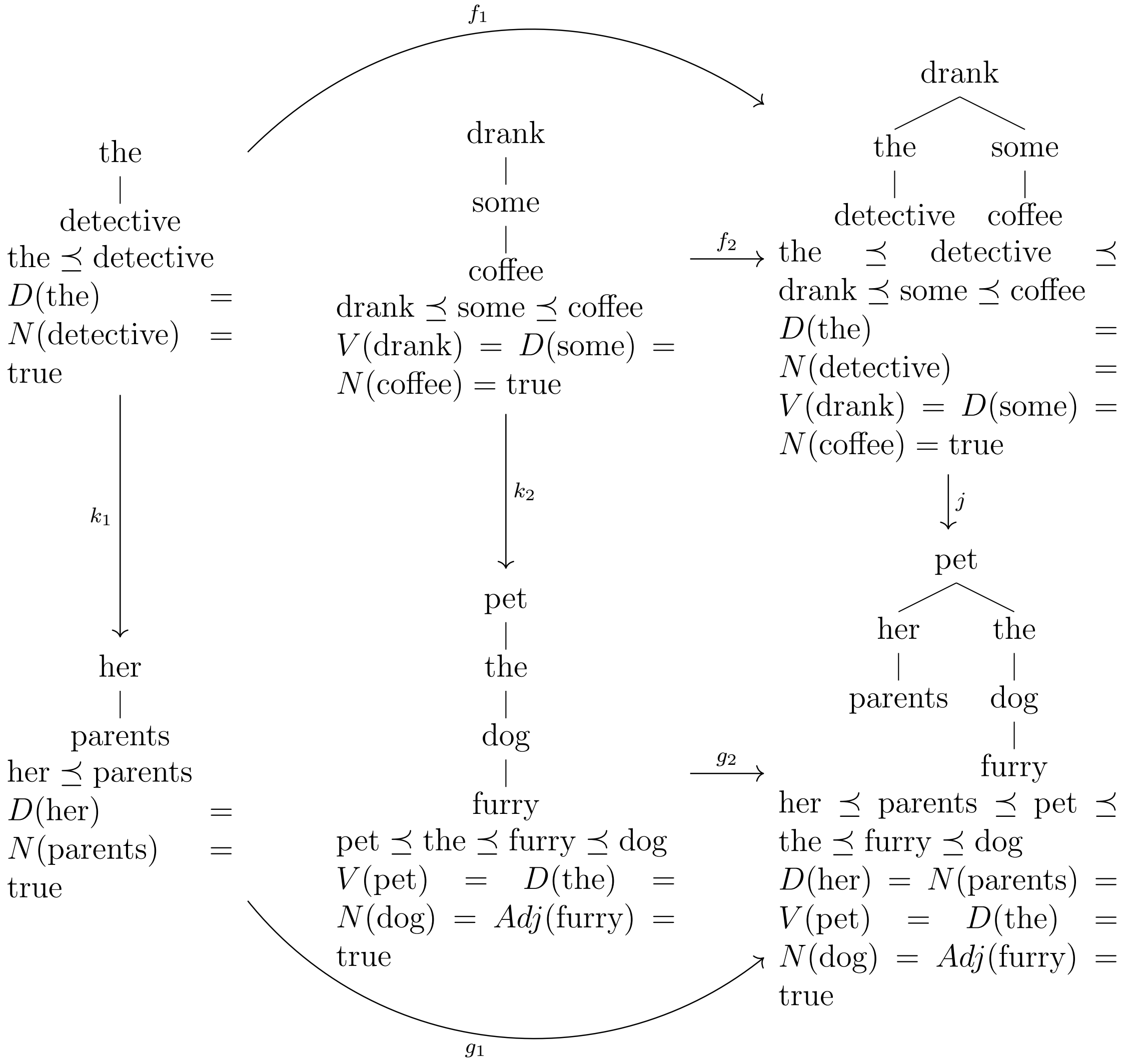

f_1
the | detective
the ⪯ detective
D(the) = N(detective) = true
drank | some | coffee
drank ⪯ some ⪯ coffee
V(drank) = D(some) = N(coffee) = true
f_2
drank / the, some / detective, coffee
the ⪯ detective ⪯ drank ⪯ some ⪯ coffee
D(the) = N(detective) = V(drank) = D(some) = N(coffee) = true
k_1
k_2
j
her | parents
her ⪯ parents
D(her) = N(parents) = true
pet | the | dog | furry
pet ⪯ the ⪯ furry ⪯ dog
V(pet) = D(the) = N(dog) = Adj(furry) = true
g_2
pet / her, the / parents, dog / furry
her ⪯ parents ⪯ pet ⪯ the ⪯ furry ⪯ dog
D(her) = N(parents) = V(pet) = D(the) = N(dog) = Adj(furry) = true
g_1

Figure 4.2: An example pushout of $\mathbf{A_L}$ objects. Here, k_1 is the obvious isomorphism, and k_2 is the only possible morphism between those objects. The induced map j maps drank ↦ pet, the ↦ her, detective ↦ parents, some ↦ the, and coffee ↦ dog

$$\begin{tikzcd}
A_1 \arrow[d, "\phi_1"'] \arrow[rrr, bend left=50] & \ldots \arrow[d] \arrow[rr, bend left=30] & A_n \arrow[d, "\phi_n"] \arrow[r, bend left=15] & A_1 + \ldots + A_n \arrow[dll, dashed, bend left=40, "u"] \\
B_1 \arrow[dr, "\kappa_1"'] & \ldots \arrow[d] & B_n \arrow[dl, "\kappa_n"] & \\
& B_1 + \ldots + B_n & &
\end{tikzcd}$$

Figure 4.3: A sum u of a tuple of maps

We will sketch a proof of the pushout lemma since many later arguments will take the same form, but first we show how it helps us. In a category $\mathbf{A}$ with coproducts, suppose we have a SC $(f_i : A_i \to Z : 1 \leq i \leq n)$ which pushes out to a SC $(g_i : B_i \to Y : 1 \leq i \leq n)$ along $(k_1, \ldots, k_n)$ such that $(g_i : B_i \to Y : 1 \leq i \leq n)$ pushes out to a SC $(h_i : C_i \to X : 1 \leq i \leq n)$ along $(l_1, \ldots, l_n)$. Then, by the pushout lemma, f pushes out to h along $l \circ k$. See Fig. 4.5.

$$\begin{tikzcd}
\overline{A} \arrow[r, "f"] \arrow[d, "\bar{k}"'] & Z \arrow[d, "j"] \\
\overline{B} \arrow[r, "g"] \arrow[d, "\bar{l}"'] & Y \arrow[d, "m"] \\
\overline{C} \arrow[r, "h"'] & X
\end{tikzcd}$$

Figure 4.5: The pushout lemma applied to a structural change translated along two condition-preserving maps.

We now prove the pushout lemma for the diagram in Fig. 4.4. (Right ⇒ Outer) Let $x : C \to Z$ and $y : D \to Z$ be morphisms such that $x \circ (g \circ f) = y \circ k$.

Since the left square is a pushout, we have a unique morphism $u : E \to Z$ such that $u \circ a = y$ and $u \circ j = x \circ g$. Since the right square is a pushout, we then have a unique function $v : F \to Z$ such that $v \circ b = u$ and $v \circ c = x$. Then $v \circ c = x$ and $v \circ (b \circ a) = u \circ a = y$. Furthermore, by the uniqueness of u and v, this v is unique. (Outer ⇒ Right) Let $x : C \to Z$ and $y : E \to Z$ be morphisms such that $x \circ g = y \circ j$. Then $x \circ (g \circ f) = y \circ j \circ f = (y \circ a) \circ k$. Since the outer square is a pushout diagram, there is a unique $u : F \to Z$ such that $u \circ c = x$ and $u \circ b \circ a = y \circ a$. Now consider the maps $u \circ b, y : E \to Z$. $y \circ j = x \circ g$ by hypothesis, and $(u \circ b) \circ j = u \circ c \circ g = x \circ g$; also $(u \circ b) \circ a = y \circ a$. Then, since the left square is a pushout diagram, and both of these morphisms factor $x \circ g$ and $y \circ a$ through j and a respectively, $y = u \circ b$, since this factorization must be unique. Thus we have produced a unique $u : F \to Z$ such that $u \circ c = x$ and $u \circ b = y$.

We now describe a rule G and its condition category. We define a binary unrestricted SPECIFIER-MERGE rule on $\mathbf{A}$ which takes in two (dominance) rooted $\mathbf{A}$-objects (A, B) such that the precedence orders on A and B are linear orders and returns a singleton consisting of the SC which adds: (1) domination relations $r \leq a$ from the root r of B to each element of A; (2) precedence relations $a \preceq b$ for each element $a \in A$ and $b \in B$; and (3) leaves syntactic type unchanged. The operations in Fig. 4.2 are examples of such SCs, and (k_1, k_2) is one of the morphisms of its condition category. We describe $\mathcal{E}_G$ for this SPECIFIER-MERGE rule. Given two pairs of input objects (A, B) and (A', B') - $\mathbf{A}$ objects with roots with respect to dominance ($\leq$) which are linear orders with respect to precedence ($\preceq$) - a pair of maps $(k_A, k_B) : (A, B) \to (A', B')$ pushes G-SCs out to G-SCs if and only if k_A and

k_B both map the respective roots with respect to dominance to roots, and take the $\preceq$-final (respectively, initial) element of A (respectively, B) to the $\preceq$-final (respectively, initial) element of A' (respectively, B'). This is indicative of the fact that only the dominance-roots are relevant for determining which dominance relations to add; we only care about the precedence-final element of A and precedence-initial element of B to determine which precedence relations to add; and we do not care about syntactic type for this unrestricted version of the rule.

The method we have sketched works for general rules. For example, if a rule has (relativized) minimality constraints, etc. the morphisms of the maximal condition category will have to take an element which is minimal in the relevant sense and of the relevant type to an element which is minimal in the relevant sense and of the relevant type. Taking the maximal condition category turns the mapping $G : \mathcal{E}_G \to \mathbf{Set}$ into a functor. Given a tuple of objects $(A_1, \ldots, A_n)$, G returns the set of G-SCs on it; given a condition-preserving morphism $k : (A_1, \ldots, A_n) \to (B_1, \ldots, B_n)$, the function $G(k) : G(A_1, \ldots, A_n) \to G(B_1, \ldots, B_n)$ takes each SC in the first set to its pushout along k.

We say that a subcategory $\mathcal{D} \hookrightarrow \mathcal{C}$ is *replete* if for any object D of $\mathcal{D}$ and isomorphism $f : D \to C$ in $\mathcal{C}$, C and f are in $\mathcal{D}$. Every subcategory has a *repletion* obtained as the category containing all objects of $\mathcal{C}$ which admit an isomorphism to some object in $\mathcal{D}$, whose morphisms are morphisms of $\mathcal{C}$ which can be obtained as a composite of morphisms of $\mathcal{D}$ and isomorphisms of $\mathcal{C}$. Such repletions have especially intuitive meanings for rules that are *structural.*

Claim 4.2.2. A subcategory $\mathcal{D} \hookrightarrow \mathcal{C}$ is equivalent to its repletion if and only if the inclusion is *pseudomonic*. That is, for any objects x, y in $\mathcal{D}$, the $\mathcal{D}$-isomorphisms between them are exactly the $\mathcal{C}$-isosmorphisms between them: $\mathcal{D}_{\text{isos}}(x, y) = \mathcal{C}_{\text{isos}}(x, y)$.

Proof. See, e.g. `https://ncatlab.org/nlab/show/replete+subcategory`. □

We say that a rule G is ***structural*** if the inclusion $\mathcal{E}_G \hookrightarrow \mathbf{A}^n$ is pseudomonic: that is, if $(A_1, \ldots, A_n)$ and $(B_1, \ldots, B_n)$ are any pair of isomorphic tuples of objects where the rule is defined, then it assigns them isomorphic SCs, and all (n-tuples of $\mathbf{A}$) isomorphisms between them are condition-preserving. Every structural rule admits a repletion in an obvious way. We add tuples $(B_1, \ldots, B_n)$ isomorphic some $(A_1, \ldots, A_n)$ already in its condition category, and define SCs on them by translating across these isomorphisms.

The above results generalize to rules which take in objects of $\mathscr{D}(\mathbf{A})$ or objects of $\mathbf{ADer}$. However, for SCs on derivations, instead of taking a pushout in the underlying category, we will want to take a SC $(f_i : \Delta_i \to Z : 1 \leq i \leq n)$ and tuple of morphisms of derivation $(k_1, \ldots, k_n) : (\Delta_1, \ldots, \Delta_n) \to (\Gamma_1, \ldots, \Gamma_n)$ and translate f along k to obtain another *SC*. We then want to find the universal such SC instead of taking the pushout (the universal such *derivation*), since the pushout would produce a more general morphism of derivations, not necessarily one which maps to a DSO.

Claim 4.2.3. Consider any extendable category of derivations $\mathbf{ADer}$ such that $\mathbf{A}$ has all finite colimits. Let $h : \Delta \to Z$ be any operation and let $\phi : \Delta \to \Gamma$ be any morphism. Then there is an $\mathbf{A}$-object P together with maps $h' : \Gamma \to P$ and $f : Z \to P$ such that $fh = h'\phi$, which is universal with respect to partial orders with this property. That is, if Q is any $\mathbf{A}$-object and $k : \Gamma \to Q$ and $g : Z \to Q$ any pair of $\mathbf{A}$-derivation morphisms such that $gh = k\phi$, then there is a unique $\mathbf{A}$-morphism $u : P \to Q$ such that $k = uh'$ and $g = uf$.

$$\begin{tikzcd}
\Delta \arrow[r, "h"] \arrow[d, "\phi"'] & Z \arrow[d, "f"] \arrow[ddr, bend left, "g"] & \\
\Gamma \arrow[r, "h'"'] \arrow[drr, bend right, "k"'] & P \arrow[dr, dotted, "u"] & \\
& & Q
\end{tikzcd}$$

As with all universal constructions, (P, h', f) is determined up to unique isomorphism, in that if $(\overline{P}, \overline{h'}, \overline{f})$ is any other such $\mathbf{A}$-object, then there is a unique isomorphism $i : P \to \overline{P}$ such that $h'i = \overline{h'}$ and $fi = \overline{f}$. We say that (P, h', f) is ***universal*** with respect to ϕ and h. We often abuse terminology and call h' the pushout of h along ϕ.

Proof. We can construct this 'pushout' explicitly. We apply $\top^{\mathbf{A}}$ to the whole diagram and take the pushout in $\mathbf{A}$ to obtain P and maps $Z \to P$ and $\top_\Gamma \to P$. We then compose the morphism $\eta_\Gamma : \Gamma \to i(\top_\Gamma)$ associated to the yield functor with $\top_\Gamma \to P$. □

G will still be a functor $G : \mathcal{E}_G \to \mathbf{Set}$ in this case. We formalize a rule on $\mathbf{A}$-derivations as follows.

Definition 4.2.1. Given a condition category $\mathcal{E}_G \subset \mathbf{ADer}^n$, a ***rule*** is a functor $G : \mathcal{E}_G \to \mathbf{Set}$ such that:

- $G(\Delta_1, \ldots, \Delta_n)$ is a set of SCs $(f_i : \Delta_i \to Z : 1 \leq i \leq n)$ such that no two are isomorphic.

- For any morphism $k : (\Delta_1, \ldots, \Delta_n) \to (\Gamma_1, \ldots, \Gamma_n)$, $G(k)$ is a function mapping each SC to its pushout along k.

For any such rule, we can expand $\mathcal{E}_G$ by adding all of the morphisms of the maximal condition category, or even further to its repletion.

Two rules G and H are intuitively *equivalent* if for each tuple $(\Delta_1, \ldots, \Delta_n)$ where G is defined, there is a tuple $(\Gamma_1, \ldots, \Gamma_n)$ such that $\Delta_i \approx \Gamma_i$ where H is defined, and for each SC $(f_i : \Delta_i \to Z : 1 \leq i \leq n)$ in $G(\Delta_1, \ldots, \Delta_n)$, we have a SC $(g_i : \Gamma_i \to Y : 1 \leq i \leq n)$ in $H(\Gamma_1, \ldots, \Gamma_n)$ such that $Z \approx Y$, and these isomorphisms commute with the f_i and g_i, and conversely for any tuple $(\Gamma_1, \ldots, \Gamma_n)$ where H is defined. Since we rely on translating across isomorphisms, this notion makes most sense when both G and H are structural. We state this precisely below.

Definition 4.2.2. Let G and H be two structural n-ary rules with (pseudomonic) inclusions $k : \mathcal{E}_G \hookrightarrow \mathbf{Der}^n$ and $j : \mathcal{E}_H \hookrightarrow \mathbf{Der}^n$. An ***equivalence*** between two structural rules G and H is given by the following data.

- Functors $A : \mathcal{E}_G \to \mathcal{E}_H$ and $B : \mathcal{E}_H \to \mathcal{E}_G$
- Natural isomorphisms $\alpha : jA \to k$ and $\beta : kB \to j$
- Natural isomorphisms $\eta : G \to HA$ and $\epsilon : H \to GB$

These must meet the following conditions. Writing $A(\Delta_1, \ldots, \Delta_n)$ as $(A\Delta_1, \ldots, A\Delta_n)$, the bijection $\eta_{(\Delta_i)_{1 \leq i \leq n}} : G(\Delta_1, \ldots, \Delta_n) \to H(A\Delta_1, \ldots, A\Delta_n)$ sends each SC $(f_i :$

$\Delta_i \to Z : 1 \leq i \leq n)$ in $G(\Delta_1, \ldots, \Delta_n)$ to an SC $(g_i : A\Delta_i \to Y : 1 \leq i \leq n)$ in $H(A\Delta_1, \ldots, A\Delta_n)$ such that there is an isomorphism $\iota : Z \to Y$ such that for each i, we have $\iota \circ f_i = g_i \circ \alpha_i$, where α_i is the isomorphism $\alpha_{\Delta_i}^{-1} : \Delta_i = k\Delta_i \to jA\Delta_i = A\Delta_i$. Conversely, $\epsilon_{(\Gamma_i)_{1 \leq i \leq n}} : H(\Gamma_1, \ldots, \Gamma_n) \to G(B\Gamma_1, \ldots, B\Gamma_n)$ sends each SC $(g_i : \Gamma_i \to Y : 1 \leq i \leq n)$ in $H(\Gamma_1, \ldots, \Gamma_n)$ to an SC $(f_i : B\Gamma_i \to Z : 1 \leq i \leq n)$ in $G(B\Gamma_1, \ldots, B\Gamma_n)$ such that there is an isomorphism $\iota : Y \to Z$ such that for each i, we have $\iota \circ g_i = f_i \circ \beta_i$, where β_i is the isomorphism $\beta_{\Gamma_i}^{-1} : \Gamma_i = j\Gamma_i \to kB\Gamma_i = B\Gamma_i$. Finally, we ask that $G(\alpha\beta A) \circ \epsilon_{A\Delta} \circ \eta_\Delta : G(\Delta_1, \ldots, \Delta_n) \to H(A\Delta_1, \ldots, A\Delta_n) \to G(BA\Delta_1, \ldots, BA\Delta_n) \to G(\Delta_1, \ldots, \Delta_n)$ is the identity function, where $\alpha\beta A$ is the morphism with components $\alpha_{\Delta_i} \circ \beta_{A\Delta_i} : BA\Delta_i = kBA\Delta_i \to jA\Delta_i \to k\Delta_i = \Delta_i$, and similarly that $H(\beta\alpha B) \circ \eta_{B\Gamma} \circ \epsilon_\Gamma$ is the identity.

In other words, an equivalence of structural rules is given by an equivalence between their maximal condition categories, together with a bijection between SCs on objects in correspondence given by the equivalence of categories, such that these bijections take isomorphic SCs to isomorphic SCs while translating along the equivalence, and are mutually inverse. The information (η, ϵ) is actually determined by (A, B, α, β). Each of η and ϵ determine each other: given η, we can recover ϵ just by looking at η and the inverse of $G(\alpha\beta A)$ at each coordinate, and conversely. To see that η is itself determined by (A, B, α, β), note that η_Δ must take each SC to an isomorphic one, composing with the isomorphisms α_i. However, since each SC is in each set only once up to isomorphism, there is only ever one such SC. So an equivalence of rules is given by an equivalence of categories between $\mathcal{E}_G$ and $\mathcal{E}_H$

which admits certain associated bijections at each coordinate. Finally, A and B determine the other up to natural isomorphism. This can be seen by noting that A and B give rise to an equivalence of categories between $\mathcal{E}_G$ and $\mathcal{E}_H$ with natural transformations $\alpha\beta A$ and $\beta\alpha B$ (*since* i and j are pseudomonic), and hence they are mutually adjoint. So, up to natural isomorphism, an equivalence is just given by a functor $A : \mathcal{E}_G \to \mathcal{E}_H$ which admits an inverse and certain isomorphisms between SCs. It is straightforward to check that this relation gives an equivalence relation on n-ary rules given by composition of such equivalences.

We say that two rules G and H are ***isomorphic*** if $\mathcal{E}_G$ and $\mathcal{E}_H$ are *equal*, and we have a natural isomorphism $\eta : G \to H$ such that $\eta_\Delta : G(\Delta_1, \ldots, \Delta_n) \to H(\Delta_1, \ldots, \Delta_n)$ takes each SC to an isomorphic one, in that if $\eta_\Delta(f_i : \Delta_i \to Z : 1 \leq i \leq n) = (g_i : \Delta_i \to Y : 1 \leq i \leq n)$, then there is a (necessarily unique) isomorphism $\iota : Z \to Y$ such that $\iota \circ f_i = g_i$. We have the following natural result.

Claim 4.2.4. Let G and H be two n-ary structural rules. Then G and H are equivalent if and only if their repletions are isomorphic.

Proof. $\Rightarrow$) If they are equivalent rules, they lead to isomorphic repletions, since they have isomorphic tuples with isomorphic SCs on them.

$\Leftarrow$) Denote the repletion of $\mathcal{E}_G$ in $\mathbf{Der}^n$ by $\rho\mathcal{E}_G$. If G and H have isomorphic repletions, then $\rho\mathcal{E}_G = \rho\mathcal{E}_H$. By Claim 4.2.2, the inclusions $\mathcal{E}_G \hookrightarrow \rho\mathcal{E}_G = \rho\mathcal{E}_H \hookleftarrow \mathcal{E}_H$ are equivalences of categories, since G and H are structural. As equivalences, they determine inverses up to natural isomorphism, and we may use these inverses to obtain an equivalence between $\mathcal{E}_G$ and $\mathcal{E}_H$ which gives rise to an equivalence of rules. □

This intuitively makes sense. Two rules are equivalent iff they do isomorphic things to isomorphic inputs. By constructing their repletions, we extend their behavior, already isomorphic, onto all tuples isomorphic to ones where they are already defined. They will have isomorphic behavior on these new tuples if and only if they already did equivalent things to isomorphic structures.

4.3 Generating Structural Changes

Now we would like to characterize basic generating SCs for rules, having defined an abstracted version of SAs. The basic idea is that a set of SCs $\{(f_{i,j} : A_{i,j} \to Z_j : 1 \leq i \leq n)\}_{j \in J}$ *generates* a rule G if every G-SC can be obtained as one of the SCs in the set pushed out along a morphism of $\mathcal{E}_G$.

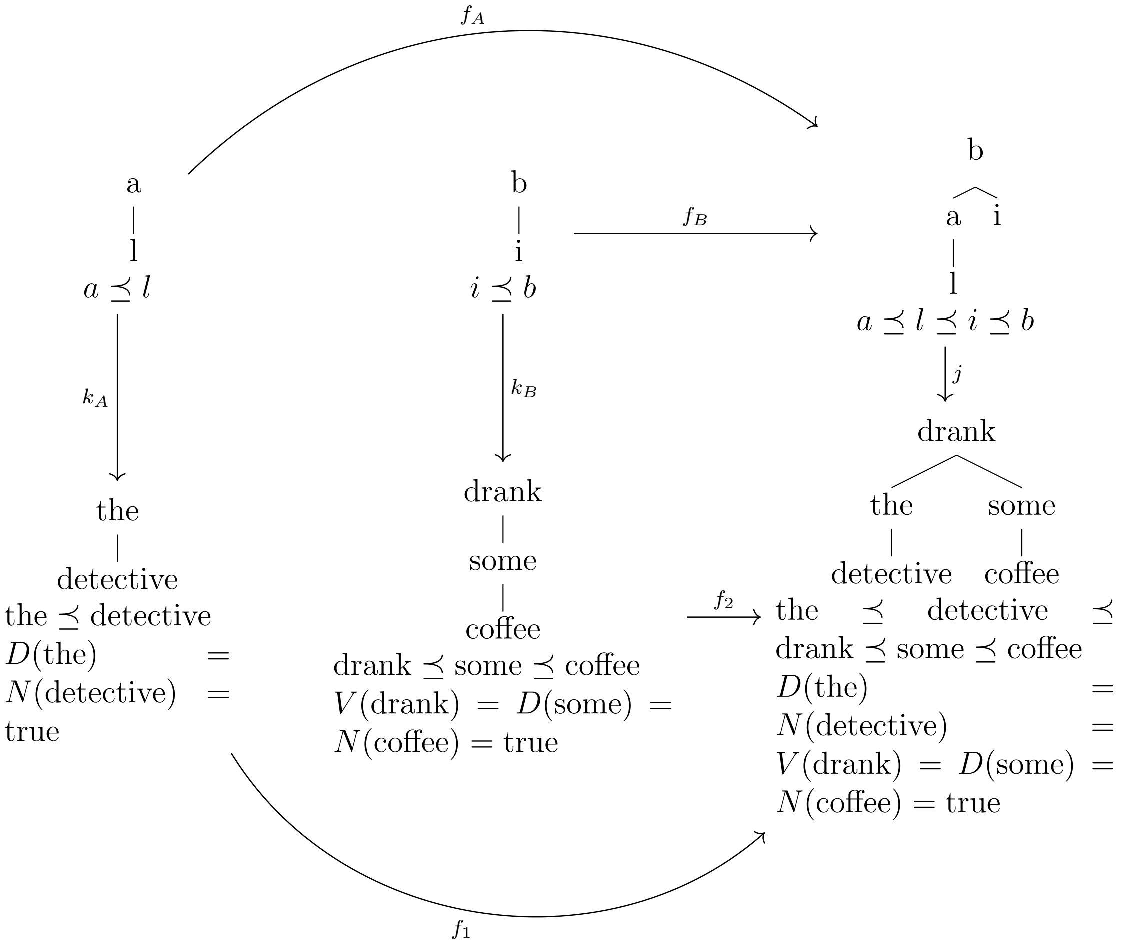
We give a single generating SC for our SPECIFIER-MERGE rule which uses precedence ordering and syntactic type, so that we can show the technique works even when our objects have 'extra structure' beyond dependencies. We will then state the technique totally abstractly, showing that it doesn't depend on any particular assumptions about the DSOs. A will consist of two elements $\{a, l\}$ with root a - i.e. $a \leq l$ - and linearly final element l - i.e. $a \preceq l$. B will consist of two elements $\{b, i\}$, a root b and linearly initial element i. In both objects, the elements have no syntactic type - i.e. $\lambda(x) =$ false for any $\lambda \in \mathbf{L}$ and any $x \in A$ or B. These objects have elements which act as dummy roots with dummy precedence final and initial elements, and are syntactically untyped since this version of the rule does not care about syntactic type. The output DSO Z consists of $\{a, b, i, l\}$ with dominance relations $b \leq i, a, l$; $a \leq l$; and the reflexive relations. It has precedence relations $a \preceq l \preceq i \preceq b$. All elements have no syntactic type, indicating that the rule doesn't change syntactic type. The pair of SC functions map each element to themselves. Denote this pair (f_A, f_B). For any pair (X, Y) of $\mathcal{E}_G$ objects, there is exactly one $\mathcal{E}_G$ morphism $(k_A, k_B) : (A, B) \to (X, Y)$ taking a to the root of X, l to the precedence-final element of X, b to the root of Y, and i to the precedence-initial element of Y. The pushout (f_A, f_B) along (k_A, k_B) is the desired SC on (X, Y). See Fig. 4.6 for an example.[2]

We can formulate generation simply using representable functors. Consider for any category $\mathcal{C}$ the category whose objects are functors $f : \mathcal{C} \to \mathbf{Set}$ and whose morphisms are natural transformations. We denote this category as $\mathbf{Set}^{\mathbf{C}}$, and we denote a set of morphisms as $\mathbf{Nat}(F, G)$. We denote a functor $\mathcal{C}(C, -)$ as $\mathbf{y}C$.

Claim 4.3.1. (Yoneda Lemma) The natural transformations $\mathbf{Nat}(\mathbf{y}C, \mathcal{F})$ from $\mathbf{y}C$ to $\mathcal{F}$ are in bijection with the elements of the set $\mathcal{F}(C)$. That is, $F(C) \cong \mathbf{Nat}(\mathbf{y}C, F)$

Proof. To construct the correspondence, see that $1_C \in \mathcal{C}(C, C) = \mathbf{y}C(C)$, and choose any natural transformation $\mu \in \mathbf{Nat}(\mathbf{y}C, \mathcal{F})$. Then $\mu_C(1_C) \in \mathcal{F}(C)$. If we can argue that this map fully determines the behavior of the natural transformation, then the proof is complete. Choose an element $f \in \mathcal{C}(A, C) = \mathbf{y}C(A)$. $1_C \in \mathcal{C}(C, C)$

[2] Note that we can manipulate feature activity using pushouts. If $\mathbf{A}$ is a category which has a predicate α 'inactive' on the sets of nodes, take the morphism $\{x\} \to \{x\}$ where $\alpha(x) =$ false in the first object and $\alpha(x) =$ true in the second. The pushout of this morphism along a morphism mapping $x \mapsto a$ will deactivate a.



f_A
a
l
a ⪯ l
k_A
the
detective
the ⪯ detective
D(the) = N(detective) = true
b
i
i ⪯ b
k_B
drank
some
coffee
drank ⪯ some ⪯ coffee
V(drank) = D(some) = N(coffee) = true
f_B
b
a i
l
a ⪯ l ⪯ i ⪯ b
j
drank
the some
detective coffee
the ⪯ detective ⪯ drank ⪯ some ⪯ coffee
D(the) = N(detective) = V(drank) = D(some) = N(coffee) = true
f_2
f_1

Figure 4.6: A basic SC generating SPECIFIER-MERGE. There is only one $\mathcal{E}_G$ morphism sending the basic generating pair to any other $\mathcal{E}_G$ object. Here, it maps $a \mapsto$ the, $l \mapsto$ detective, $b \mapsto$ drank, and $i \mapsto$ drank. Intuitively, the basic SC adds a dominance relation $b \leq a$ between the roots, and precedence relation $l \preceq i$, while leaving syntactic type alone. (f_A, f_B) and (k_A, k_B) determine the output DSO as well as (f_1, f_2).

maps to $f \in \mathcal{C}(C, A)$ under the function $\mathbf{y}(f) \equiv f \circ -$. But the naturality condition means that the following diagram must commute.

$$\begin{tikzcd}
1_C \arrow[d, mapsto] & \in & \mathbf{y}C(C) = \mathcal{C}(C, C) \arrow[r, "\mu_C"] \arrow[d, "f\circ -"'] & \mathcal{F}(C) \arrow[d, "\mathcal{F}(f)"] & \ni & \mu_C(1_C) \arrow[d, mapsto] \\
f & \in & \mathbf{y}C(A) = \mathcal{C}(A, C) \arrow[r, "\mu_A"'] & \mathcal{F}(A) & \ni & \mu_A(f) = \mathcal{F}(f)(\mu_C(1_C))
\end{tikzcd}$$

So, $\mu_A(f)$ must equal $\mathcal{F}(f)(\mu_C(1_C))$. So the natural transformation $\mu : \mathbf{y}C \to \mathcal{F}$ is totally determined by the value of $\mu_C(1_C)$. The bijection $\mathcal{F}(C) \approx \mathbf{Nat}(\mathbf{y}C, F)$ is given by the correspondence $\mu_C(1_C) \leftrightarrow \mu$. □

A natural transformation $\mu : F \to G$ in $\mathbf{Set}^{\mathcal{C}}$ for any $\mathcal{C}$ is an epimorphism if and only if for every object C in $\mathcal{C}$, $\mu_C : F(C) \to G(C)$ is surjective. Given a SC $r \equiv (f_i : \Delta_i \to Z : 1 \leq i \leq n) \in G(\Delta_1, \ldots, \Delta_n)$, there will be an associated natural transformation $\tilde{r} : \mathbf{y}(\Delta_1, \ldots, \Delta_n) \to G$.

Definition 4.3.1. We say that an SC r ***generates*** G if $\tilde{r}$ is an epimorphism in $\mathbf{Set}^{\mathcal{E}_G}$.

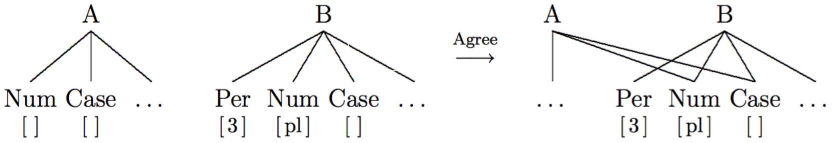
We can then think of this single SC as *covering* the whole rule G. This is equivalent to the statement that for any tuple $(\Gamma_1, \ldots, \Gamma_n)$, the function $\tilde{r}_{(\Gamma_1, \ldots, \Gamma_n)} : \mathcal{E}_G((\Delta_1, \ldots, \Delta_n), (\Gamma_1, \ldots, \Gamma_n)) \to G(\Gamma_1, \ldots, \Gamma_n)$ taking each condition-preserving morphism $k : (\Delta_1, \ldots, \Delta_n) \to (\Gamma_1, \ldots, \Gamma_n)$ to the pushout of r along k is surjective. Hence, every G-SC on $(\Gamma_1, \ldots, \Gamma_n)$ arises as the pushout of r along some condition-preserving morphism.

There is a natural generalization for describing generation by a set of SCs. Given a set of SCs $\{r_i\}_{i \in I}$, possibly on different objects, we can obtain a set of natural transformations into G. We say that this set of SCs *generates* G if their sum is epimorphic. This is equivalent to saying that each SC in $G(\Gamma_1, \ldots, \Gamma_n)$ arises as the pushout of some r_i along some condition-preserving morphism. When a rule G is generated by a single operation r, grammatical operations essentially reduce to the *pushout method* of applying rules in graph grammars [5]. The condition categories play the generalized role of constraints and application conditions in graph grammars [35]. We think of $\mathcal{E}_G$ as the SA, and a generating SC $r \equiv (f_i : \Delta_i \to Z : 1 \leq i \leq n) \in G(\Delta_1, \ldots, \Delta_n)$ as a basic production rule. Intuitively, a basic SC and SA should determine the rule G. This is true.

Claim 4.3.2. Let $\mathcal{E}_G \subset \mathbf{ADer}^n$ be a category of tuples of $\mathbf{A}$-derivations, and let $r \equiv (f_i : \Delta_i \to Z : 1 \leq i \leq n) \in G(\Delta_1, \ldots, \Delta_n)$ be a SC. Then r determines a rule G up to isomorphism, in that if G and G' are any two rules generated by r, then $G \approx G'$, and furthermore the bijections $G(\Delta_1, \ldots, \Delta_n) \approx G'(\Delta_1, \ldots, \Delta_n)$ take each SC to an isomorphic one.

Proof. This follows straightforwardly from the definition of rules and generation. If $G : \mathcal{E}_G \to \mathbf{Set}$ is any rule, and $\tilde{r} : \mathbf{y}(\Delta_1, \ldots, \Delta_n) \to G$ is an epimorphism, then every element of $G(\Gamma_1, \ldots, \Gamma_n)$ for any tuple can be obtained as the pushout of r along a morphism in $\mathcal{E}_G((\Delta_1, \ldots, \Delta_n), (\Gamma_1, \ldots, \Gamma_n))$. Then, each SC is determined up to isomorphism of SCs. The result generalizes naturally to generation by a set of SCs. □

It is a welcome result that the SA $\mathcal{E}_G$ and basic SC determine a rule G, but only up to natural isomorphism. Consider the following formalization of feature-sharing (which notably we use for selection) in Frampton & Gutmann [12], given in the Bare Phrase Structure framework.

> Consider [(2)] and suppose that Agree applies to the pair of nodes.
>
> (2) $\{\mathrm{Num}_1, \mathrm{Case}_2, \ldots\}, \{\mathrm{Per}_3, \mathrm{Num}_4, \mathrm{Case}_5, \ldots\}$
>
> [...] suppose that Agree induces feature sharing, so that matching features coalesce into a single shared feature, which is valued if either of the coalescing features is valued. So [(2)] produces:
>
> (3) $\{\mathrm{Num}_6, \mathrm{Case}_7, \ldots\}, \{\mathrm{Per}_3, \mathrm{Num}_6, \mathrm{Case}_7, \ldots\}$
>
> The value of Num_6 is the coalescence of the values of Num_1 and Num_4. The value of Case_7 is the coalescence of the values of Case_2 and Case_5. New indices were chosen, but index 6, for example, could just as well have been 1 or 4. **The choice of index is not a substantive question, assuming that it is suitably distinguished.** [my emphasis]
>
> If the two coalescing features are both valued to start with, it is not clear that the result is coherent. But this will never arise, because Agree is driven by an unvalued feature.
>
> A picture will make the idea clearer. Agree takes [(4a)] into [(4b)], assuming that none of the features indicated by the ellipsis marks match.
>
> (4)
>
> 
>
>
> Frampton & Gutmann [12], p. 4

The emphasized statement can be interpreted as meaning exactly that the SC is determined only up to isomorphism. It is the structure of the output object and the maps into it which are determined, not the name of the element used to represent the identified points.

4.3.1 Inclusiveness and Weak Extension

One common assumption about grammatical operations is that they are *inclusive*. One statement of this is given by Chomsky. "The Inclusiveness Condition: No new features are introduced by C_{HL} [the computational procedure for human language]," Chomsky [36], p. 113. We would like to see how this can be stated naturally in the language of our categories of DSOs. One way to interpret the Inclusiveness Condition (IC) is to say that a SC may only introduce *structure* but not *new material*. If the IC is too strong, and cannot even do that, then operations can't do anything. This can be formalized on concrete categories by asking that a SC $f_i : A_i \to Z$ is *jointly surjective*, that is, for each $z \in Z$, there is some $a \in A_i$ such that $f_i(a) = z$. When the forgetful functor commutes with sums, this can be stated simply as saying that $f : \coprod A_i \to Z$ is surjective.

We note that epimorphisms in many of our categories of DSO are often identically the morphisms with underlying surjective functions.

- The epimorphisms in **Set** are exactly the surjective functions.
- The epimorphisms in **FPos** are exactly the order-preserving surjections.
- The epimorphisms in **A** of sets X with dependency partial order $\leq$ and prece-

dence preorder $\preceq$ are exactly the surjections.

Claim 4.3.3. In any construct, surjections are epimorphisms.

Proof. Let $f : A \to B$ be any surjective morphism. Let $x, y : B \rightrightarrows C$ be any two morphisms such that $xf = yf$. For any $b \in B$, we can find some $a \in A$ such that $f(a) = b$ by surjectivity. But for any a, $x(f(a)) = y(f(a))$, so $x(b) = y(b)$ for all $b \in B$, and $x = y$ as functions, and hence as morphisms by the faithfulness of the forgetful functor. □

Sometimes, a construct may have epimorphisms which are not surjections, but we can often discover that the two notions are coextensive using a proof technique like that for Claim 3.5.6. Using the more abstract notion *epimorphism* has many technical advantages. For one, we have the following fact.

Claim 4.3.4. Let $e : A \to B$ be any epimorphism, and let $f : A \to C$ be any morphism. Then the pushout $e' : C \to D$ of e along f is also an epimorphism.

Consider why this is useful. We can say that a rule G is *inclusive* if for every G-SC $(f_i : \Delta_i \to Z : 1 \leq i \leq n)$, the morphism $\top_f : \coprod \top_{\Delta_i} \to Z$ is epimorphic. It is an immediate consequence that if a rule is generated by an epimorphic SC, then G is inclusive.

When we build derivations over constructs, we can also state a weak form of the Extension Condition (EC). "One possibility is to stipulate that the Extension Condition always holds: operations preserve existing structure," Chomsky [36], p. 136. There are various levels of strengths of the EC which we can implement. A very strong one is that given two objects A and B which combine to form Z, A and B should be constituents of Z. That will clearly be too strong for our dependency structures, where root attachment always only implies that the attachee is not a constituent. A weaker form can be stated: a SC is *weakly extensive* if each morphism $\top_{f_i} : \top_{\Delta_i} \to Z$ is an embedding. However, this is still quite strong - this will not allow us to do many reasonable things like deactivate features. We may only want to require that a SC be weakly extensive on its underlying finite partial order in the case we are working in an extendable category $U^{\mathbf{A}} : \mathbf{A} \to \mathbf{FPos}$. This will imply that we add no order relations *within* each operand, only *between* elements of different operands. We can similarly say that a rule G has the relevant property when every SC does.

4.3.2 Example Grammar: Boston et al. 2010

We demonstrate a fragment of a more fully-fledged grammar with ‘extra structure’. We demonstrate the main properties of ‘nice’ additions of structure to DSOs by sketching a model based on Boston, et al. (BHK) [16], from which we can generalize. Like their model, we will manipulate dependency structures. Unlike their model, we will put features directly in these structures. Also unlike their model, we will not allow totally empty components, though this will not affect derivations actually used by the model. We review their model below.

> We define dependency trees in terms of their nodes, with each node in a dependecy tree labeled by an *address*, a sequence of positive inte-

gers. We write λ for the empty sequence of integers. Letters u, v, w are variables for addresses, $\mathbf{s}$, $\mathbf{t}$ are variables for sets of addresses, and letters x, y are variables for sequences of addresses. If u and v are addresses, then the concatenation of the two is as well, denoted by uv. Given an address u and set of addresses $\mathbf{s}$, we write $\uparrow_u \mathbf{s}$ for the set $\{uv \mid v \in \mathbf{s}\}$. Given an address u and a sequence of addresses $x = v_1, \ldots, v_n$, we write $\uparrow_u$ for the sequence $uv_1, \ldots, uv_n$. Note that $\uparrow_u \mathbf{s}$ is a *set* of addresses, whereas $\uparrow_u x$ is a *sequence* of addresses.

A *tree domain* is a set $\mathbf{t}$ of addresses such that, for each address u and each integer $i \in \mathbb{N}$, if $ui \in \mathbf{t}$, then $u \in \mathbf{t}$ (prefix-closed), and $uj \in \mathbf{t}$ for all $1 \leq j \leq i$ (left-sibling closed). A *linearization* of a finite set S is a sequence of elements of S in which each element occurs exactly once. For the purposes of this paper, a *dependency tree* is a pair $(\mathbf{t}, x)$, where $\mathbf{t}$ is a tree domain, and x is a linearization of $\mathbf{t}$. A *segmented dependecy tree* is a non-empty sequence $(\mathbf{s})_1, x_1), \ldots, (\mathbf{s})_n, x_n)$, where each $\mathbf{s}_i$ is a set of addresses, each x_i is a linearization of $\mathbf{s}_i$, all sets $\mathbf{s}_i$ are pairwise disjoint, and the union of the sets $\mathbf{s}_i$ forms a tree domain. A pair $(\mathbf{s}_i, x_i)$ is called a *component*, which corresponds to *chains* in Stabler and Keenan's terminology [4].

An *expression* is a sequence of triples $(c_1, \tau_1, \gamma_1), \ldots, (c_1, \tau_1, \gamma_1)$, where $(c_1, \ldots, c_n)$ is a segmented dependency tree, each τ_i is a type (lexical or derived), and each γ_i is a sequence of features. We write these triples as $c_i :: \gamma_i$ (if the type is lexical), $c_i : \gamma_i$ (if the type is derived), or $c_i \cdot \gamma_i$

$$\dfrac{(\{\lambda\},\langle\lambda\rangle) :: {=}f\gamma \qquad (\mathbf{t},x)\cdot f,\alpha_1,\ldots,\alpha_k}{(\{\lambda\}\cup\uparrow_1\mathbf{t},\langle\lambda\rangle\cdot\uparrow_1 x) : \gamma,\uparrow_1\alpha_1,\ldots,\uparrow_1\alpha_k}\,\mathtt{merge1}_{DG}$$

$$\dfrac{(\mathbf{s},x) : {=}f\gamma,\alpha_1,\ldots,\alpha_k \qquad (\mathbf{t},y)\cdot f,\iota_1,\ldots,\iota_l}{(\mathbf{s}\cup\uparrow_i\mathbf{t},\uparrow_i y\cdot x) : \gamma,\alpha_1,\ldots,\alpha_k,\uparrow_i\iota_1,\ldots,\uparrow_i\iota_l}\,\mathtt{merge2}_{DG}$$

$$\dfrac{(\mathbf{s},x)\cdot{=}f\gamma,\alpha_1,\ldots,\alpha_k \qquad (\mathbf{t},y)\cdot f\delta,\iota_1,\ldots,\iota_l}{(\mathbf{s},x) : \gamma,\alpha_1,\ldots,\alpha_k,(\uparrow_i\mathbf{t},\uparrow_i y) : \delta,\uparrow_i\iota_1,\ldots,\uparrow_i\iota_l}\,\mathtt{merge3}_{DG}$$

$$\text{where } i = \mathit{next}((\mathbf{s},x)\cdot{=}f\gamma,\alpha_1,\ldots,\alpha_k)$$

Figure 4.7: Merge rules on dependency trees [16]

(if the type does not matter). We use the letters α and ι as variables for elements of an expression. Given an element $\alpha = ((\mathbf{s},x),\tau,\gamma)$ and an address u, we write $\uparrow_u \alpha$ for the element $((\uparrow_u \mathbf{s},\uparrow_u x),\tau,\gamma)$.

Given an expression d with associated tree domain $\mathbf{t}$, we write $\mathit{next}(d)$ for the minimal positive integer i such that $i \notin \mathbf{t}$. BHK [16]

We reproduce their grammatical MERGE rules in Fig. 4.7.

We then construct a more general category of expressions than those in Boston, et al. Define an ***expression*** to be any finite partial order $\mathbf{s}$ whose 'dominance/dependency' order is written $\leq$, together with (1) a partition $S = \{\mathbf{s}_1,\ldots,\mathbf{s}_n\}$ and a 'chain' preordering $\rightharpoonup$ on S; (2) for each each $\mathbf{s}_i$ a subset $\pi_i \subset \mathbf{s}_i$ of pronouncable nodes and a 'precedence' preordering $\preceq_i$ on π_i; (3) a unary predicate δ 'derived' on the set S; and (4) a subset $\gamma_i \subset \mathbf{s}_i$ of 'active' features on each partition, together with a preordering $\ll_i$ on each set γ_i representing 'checking order'. Usually, $\rightharpoonup$ and each $\preceq_i$ and $\ll_i$ are linear orders, but we generalize so that the category is more well-behaved: for example, we may like to take sums of syntactic objects, where the chains are linearly ordered in each of the summands, but they are not ordered with respect to chains from the other summand.

Writing the above expression simply as σ, a morphism $f : \sigma \to \tau$ between derivations is a function $f : \mathbf{s} \to \mathbf{t}$ between partial orders, preserving $\leq$ such that (1) if $x, y \in \mathbf{s}$ are in the same partition $\mathbf{s}_i$, then $f(x), f(y)$ are in the same partition $\mathbf{t}_j$ such that $\hat{f}$ preserves $\rightharpoonup$, where $\hat{f} : \{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \to \{\mathbf{t}_1, \ldots, \mathbf{t}_m\}$ is the map taking $\mathbf{s}_i$ to $\mathbf{t}_j$ if there exists $x \in \mathbf{s}_i$ such that $f(x) \in \mathbf{t}_j$; (2) if $x \in \mathbf{s}_i$ is such that $x \notin \pi_i$, then $f(x) \notin \pi'_j \subset \hat{f}(\mathbf{s}_i) = \mathbf{t}_j$, and if $x \preceq_i y$ in π_i and $f(x), f(y) \in \pi'_j \subset \hat{f}(\mathbf{s}_i) = \mathbf{t}_j$ are still pronounceable, then $f(x) \preceq'_j f(y)$; (3) if $\delta(\mathbf{s}_i) =$ true is 'derived', then $\delta'(\hat{f}(\mathbf{s}_i)) =$ true is 'derived'; (4) if $x \in \mathbf{s}_i$ is such that $x \notin \gamma_i$, then $f(x) \notin \gamma'_j \subset \hat{f}(\mathbf{s}_i) = \mathbf{t}_j$, and if $x \ll_i y$ in γ_i and $f(x), f(y) \in \gamma'_j \subset \hat{f}(\mathbf{s}_i) = \mathbf{t}_j$ are still active, then $f(x) \ll'_j f(y)$.[3]

It must be checked that such expressions and morphisms form a category, and in fact they do, and we denote it $\mathbf{A}$. The map $U : \mathbf{A} \to \mathbf{FPos}$ taking σ to $\mathbf{s}$ and a morphism f to its underlying order-preserving function on nodes is a functor, and it is faithful. Taking the underlying set of UA gives a functor $U' : \mathbf{A} \to \mathbf{Set}$ which is faithful and turns $\mathbf{A}$ into a construct. U' is in fact represented by $\bullet$, the expression with one element which is active and pronounceable. Finally, we want to add various 'types' to features. Let $\mathbf{B}$ be a (finite) set of types N, V, T, wh, etc. Since having

[3] Notice, we allow a generalization common in morphosyntactic theories such as Distributed Morphology [31] and nanosyntax [15] - we do not require that the pronounceable nodes are disjoint from the features. A feature may belong both to γ, indicating that it may be actively involved in selection, licensing, or agreement, but also π, indicating it is part of the pronounceable structure.

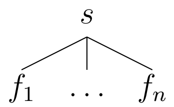
dominance order	
partition (chains)	$\mathrm{Ch}_\sigma = \{S\}$ $S = \{s, f_1, \dots, f_n\}$ $\delta(S)$ = false (S is underived)
π sets	$\pi_S = \{s\}$
γ sets	$\gamma_S = f_1 \ll \dots \ll f_n$

Figure 4.8: A BHK lexical item as an $\mathbb{A}$-object.

multiple properties is possible in principle (e.g. an element may be both N and wh), we 'type' elements of the DSOs by treating the elements of $\mathbf{B}$ as independent unary predicates on their sets of elements. That is, we add $\mathbf{B}$-data to $\mathbf{A}$ by equipping the nodes of an object σ with predicates σ_N, σ_V, etc., and morphisms must 'preserve' these determinations in that $f : \sigma \to \tau$ should be a homomorphism only if it is an $\mathbf{A}$-morphism and if $\sigma_X(a) = \text{true}$ for some type $X \in \mathbf{B}$, then $\tau_X(f(a)) = \text{true}$. Call this category $\mathbf{A}^{\mathbf{B}}$ or simply $\mathbb{A}$ when $\mathbf{B}$ is obvious. It is again representably concrete by the same object $\bullet$ with $\bullet_X(*) = \text{false}$ for all $X \in \mathbf{B}$.

We can represent the syntactic objects of BHK with objects of $\mathbb{A}$, and we can represent their structural changes as morphisms of $\mathbb{A}$. Given base types $F = \{V, N, T, wh, \dots\}$, we take the product

$\{\mathit{base}\ (\text{selectee}), =\ (\text{selector}), +\ (\text{licensor}), -\ (\text{licensee})\} \times F$ to obtain types $=N$, $+wh$, $-wh$, etc., and we let $\mathbf{B}$ have at least these types. We sketch a translation.

A lexical item in BHK is roughly a tree consisting of a single node, which has type 'underived', together with a finite linear order of features $f_0, \dots, f_n$. To each lexical item $s :: f_1 \dots f_n$, we associate the $\mathbb{A}$ object σ depicted in Fig. 4.8.

We implicitly assume that each feature f_i also returns 'true' for certain **B**-predicates. For example, if a feature f_i is $=n$ or 'select noun' type, then $\sigma_{(=,N)}(f_i) =$ true.

A general expression in BHK is essentially a tree $\mathbf{t}$ with a partition[4] $\{\mathbf{t}_1, \ldots, \mathbf{t}_n\}$ of its nodes, together with a linear precedence ordering $\preceq_i$ on each partition. Each partition is designated as 'derived (:)' or 'underived (::)', and has associated to it a finite linear order of features. BHK's first MERGE rule is the case when a lexical item selects a feature f which is the only feature associated with the first chain in some expression. Consider a lexical item $s :: f_0 \ldots f_n$ and expression $e = (\mathbf{t}_1, \preceq_1)\cdot g_0, \alpha_1, \ldots, \alpha_k$.[5] BHK say that MERGE1 is defined when f_0 is a selector, g_0 a selectee, and both have the same syntactic type (e.g. n). In this case, they define MERGE1 to be the operation which associates to this pair a new expression. This expression has as nodes the disjoint union of $\{s\}$ and the nodes of e, with all of their dominance

[4] This is not quite true: while BHK require that collection of $\mathbf{t}_i$ unions to $\mathbf{t}$ and the $\mathbf{t}_i$ are pairwise disjoint, they leave open the possibility that a particular segment $\mathbf{t}_i$ is empty. This will allow chains with features which are not associated to any node(s) in the dependency tree. However, for us, to be empty, you must not only be 'phonologically empty', but also have no syntactic features, since we will represent features in our dependency structures explicitly. Featureless chains may never be selected or moved, nor may they select, so they play little role in the theory (e.g. featureless words can never enter any derivation). Featureless chains may then only arise as the 'final' output of a derivation. However, the way we model expressions does not truly delete features, but rather just 'deactivates' them, so even in this case the object will not be truly empty.

[5] Here, α_i is an arbitrary component of the partition, together with precedence order, (un)derived type, and feature sequence. $\cdot$ means that it doesn't matter if the first chain is derived or underived.

dominance order	σ has a (unique) root $s \leq \sigma$	τ has a root $t \leq \tau$
partition (chains)	$\mathrm{Ch}_\sigma = \{S\}$, $\delta S = \text{false}$	Ch_τ has root $T \rightharpoonup \mathrm{Ch}_\tau$
π sets	$s \in \pi_S$	
γ sets	γ_S has root $f \ll \gamma_S$	γ_T has root $g \ll \gamma_T$

Figure 4.9: SA of $\mathbb{A}$ objects which we will apply MERGE1 to.

relations, as well as a relation from s to the root of e. The first chain is the partition block $\{s\} \cup \mathbf{t}_1$ of derived type, with string of features $f_1, \ldots, f_n$. This partition has precedence order $s \preceq x$ for all $x \in \mathbf{t}_1$, as well as any precedence relations from $\mathbf{t}_1$. The remaining chains are $\alpha_1, \ldots, \alpha_n$. These changes can intuitively be described as taking the disjoint union of nodes, adding certain $\leq$ dependencies, combining certain partitions, adding certain precedence relations, deleting certain features, etc., while leaving other structure intact. We wish to formalize this fact by representing the operation using $\mathbb{A}$-pushouts.

We give a (vastly) generalized description in terms of a pair of $\mathbb{A}$ objects (σ, τ). If $(X, \lhd)$ is any preorder and $x \in X$ is a *unique least element* in that $x \lhd y$ for all $y \in X$ and if z also has this property, then $x = z$, we call it a *root* and write $x \lhd X$. We first assume that the head is overtly pronounceable (has a node which will be precedence-ordered), and we other write minimal assumptions about the DSOs we want to apply MERGE1 to in Fig. 4.9. We additionally assume that f is of type $(=, X)$ and g of type $(base, X)$ for some $X \in F$. We construct a binary rule on these objects by a universal property.

Let E be any expression in $\mathbb{A}$. We can say that a pair of morphisms $h : \sigma \to E$

and $k : \tau \to E$ ***meet the* merge1 *condition*** if its images meet certain conditions. For an element x of σ or τ, we write $\overline{x}$ for its image under h or k in E. Similarly, for any partition P of σ or τ, we write $\overline{P}$ for the partition containing $\overline{x}$ for any $x \in P$. h and k meet the MERGE1 condition if (1) $\bar{s} \leq \bar{t}$; (2) $\overline{f} = \overline{g}$, which implies that $\overline{S} = \overline{T}$ which we denote ST; (3) $\overline{f}, \overline{g} \notin \gamma_{ST}$; (4) such that if $x \in \pi_T$ and $\overline{s}, \overline{x} \in \pi_{ST}$, then $\overline{s} \preceq \overline{x}$; and (5) ST is derived ($\delta(ST) =$ true). There is a *universal* such expression E which meets the MERGE1 condition determined up to isomorphism, which we write MERGE1(σ, τ). By this, we mean that there is a pair of morphisms $m_\sigma : \sigma \to$ MERGE1(σ, τ) and $m_\tau : \tau \to$ MERGE1(σ, τ) such that for any h and k meeting the MERGE1 condition, there is a unique $u :$ MERGE1$(\sigma, \tau) \to E$ such that $h = u \circ m_\sigma$ and $k = u \circ m_\tau$. It can be constructed by adding the requisite relations, deactivating the relevant features, etc. Restricted to objects of the kind from BHK (such as when the Ch, π, and γ sets are linear orders, $\pi_S = \{s\}$, etc.) this construction produces the correct structural changes and assignments of derived objects. The category of such objects is clearly replete in $\mathbb{A}^2$, and we can find the maximal condition category for these structural changes. To account for the case when the head is not associated with an element which is linearized in the pronounceable string (i.e. it is phonologically null), we may drop the assumption that s in σ is in π_S, in which case (4) of the MERGE1 condition becomes vacuous since $\bar{s} \notin \pi_{ST}$.

In simple cases, we can generate instances of MERGE1 and the structural change maps as pushouts from a finite set of basic structural changes. This must be broken down into cases, depending on whether the first component of each operand

dominance order	σ has a (unique) root $s \leq \sigma$	τ has a root $t \leq \tau$
partition (chains)	$\mathrm{Ch}_\sigma = \{S\}$, $\delta S = \text{false}$	Ch_τ has root $T \rightharpoonup \mathrm{Ch}_\tau$
π sets	$\{s\} = \pi_S$	π_T has root $l \preceq \pi_T$
γ sets	γ_S has root $f \ll \gamma_S$	γ_T has root $g \ll \gamma_T$

Figure 4.10: A more restrictive SA for MERGE1.

has pronounceable elements which we intend to introduce precedence relations between. We make extra assumptions about objects in the domain of MERGE1 in Fig. 4.10 for the case when both have pronounceable elements, still assuming that f and g are of types $(=, X)$ and $(base, X)$ for some $X \in F$.

Note that for any pair of such $\mathbb{A}^2$ objects, a morphism $(u_1, u_2) : (\sigma, \tau) \to (\sigma', \tau')$ which preserves the 'properties relevant for evaluating the MERGE1 condition' belongs to the maximal condition category. By 'preserves the relevant properties', we mean that u_1 and u_2 take the $\leq$, $\rightharpoonup$, $\preceq$, and $\ll$ roots of σ and τ to the $\leq$, $\rightharpoonup$, $\preceq$, and $\ll$ roots of σ' and τ'. For example, if s' and t' are the roots of σ' and τ', then for any $k : \sigma' + \tau' \to E$, if there exists a morphism $v : \text{MERGE1}(\sigma, \tau) \to E$ such that for all $x \in \sigma$, $k(u_1(x)) = v(m_\sigma(x))$ and for all $x \in \tau$, $k(u_2(x)) = v(m_\tau(x))$, we must have $ks' \leq kt'$. Similar arguments show that the existence of such a v puts exactly the constraint on k that it meet the MERGE1 conditions. We can then show that for all such pairs (σ, τ), the MERGE1 operations on it are generated by a finite set of SCs, pushed out along one such (u_1, u_2) 'preserving the relevant properties'. Hence, MERGE1 will be finitely generated when restricted to such pairs. We describe one of the generating operations in Fig. 4.11.

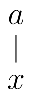
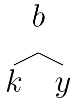
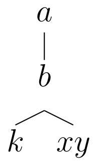
dominance order			
partition	$\mathrm{Ch}_A = \{\alpha\}$, $\alpha = \{a, x\}$, $\delta(\alpha) = \text{false}$	$\mathrm{Ch}_B = \{\beta\}$, $\beta = \{b, k, y\}$, $\delta(\beta) = \text{false}$	$\mathrm{Ch}_C = \{\kappa\}$, $\kappa = \{a, b, xy\}$, $\delta(\kappa) = \text{true}$
π sets	$\pi_\alpha = \{a\}$	$\pi_\beta = \{b, k, y\}$, $k \preceq b, y$	$\pi_\kappa = \{a, b, k\}$, $a \preceq k \preceq b$
γ sets	$\gamma_\alpha = \{x, a\}$, $x \ll a$	$\gamma_\beta = \{y\}$	$\gamma_\kappa = \{a\}$

Figure 4.11: Generating SC for simplified MERGE1.

We denote these expressions in Fig. 4.11 as A, B, and C, respectively. These objects together with the relevant condition-preserving morphisms describe the following properties: (A) the left operand has a pronounceable node which is the root, and an active feature x which is at the front of the feature-list, and it consists of a single component which is underived; and (B) the right operand has first component with k precedence-initial amongst the pronounceable nodes, and a single active feature y. For each syntactic type $X \in F$, we must take one such triple (A, B, C) with x of type $(=, X)$ and y of type (base, X), and hence xy will have types $(=, X)$ and (base, X). In each case, there is an obvious $\mathbb{A}$-morphism $r : A + B \to C$ mapping a to a, b to b, k to k, and x and y to xy. For any (σ, τ) meeting the descriptions in Fig. 4.10 such that f is of type $(=, X)$ and g is of type (base, X) for some $X \in F$, there will be at most one condition-preserving morphism from one of these generating SCs to (σ, τ). Specifically, there will be one morphism $(u_1, u_2) : (A, B) \to (\sigma, \tau)$ from the generating operation such that (i) x and f and (ii) y and g have matching type. We take the pushout of r along the map $u_1 + u_2 : A + B \to \sigma + \tau$ which takes the

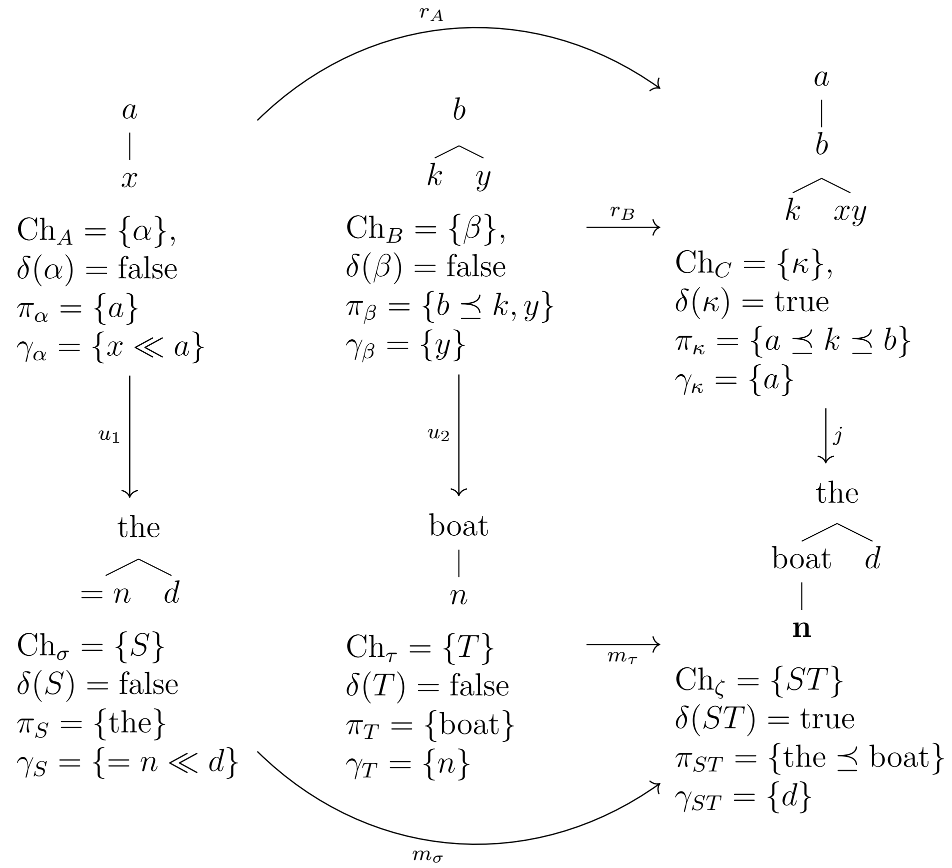

r_A
a
x
b
k y
a
b
k xy
r_B
$\mathrm{Ch}_A = \{\alpha\}$,
$\delta(\alpha) = \text{false}$
$\pi_\alpha = \{a\}$
$\gamma_\alpha = \{x \ll a\}$
$\mathrm{Ch}_B = \{\beta\}$,
$\delta(\beta) = \text{false}$
$\pi_\beta = \{b \preceq k, y\}$
$\gamma_\beta = \{y\}$
$\mathrm{Ch}_C = \{\kappa\}$,
$\delta(\kappa) = \text{true}$
$\pi_\kappa = \{a \preceq k \preceq b\}$
$\gamma_\kappa = \{a\}$
u_1
u_2
j
the
$=n$ d
boat
n
the
boat d
n
m_τ
$\mathrm{Ch}_\sigma = \{S\}$
$\delta(S) = \text{false}$
$\pi_S = \{\text{the}\}$
$\gamma_S = \{= n \ll d\}$
$\mathrm{Ch}_\tau = \{T\}$
$\delta(T) = \text{false}$
$\pi_T = \{\text{boat}\}$
$\gamma_T = \{n\}$
$\mathrm{Ch}_\zeta = \{ST\}$
$\delta(ST) = \text{true}$
$\pi_{ST} = \{\text{the} \preceq \text{boat}\}$
$\gamma_{ST} = \{d\}$
m_σ

Figure 4.12: The pushout of a generating BHK rule along a condition-preserving morphism (u_1, u_2)

root a to the root of σ, the element x to f, the root b to the root of τ, the element y to the feature g, and k to l. It can be checked that the pushout of r along $u_1 + u_2$ induces the appropriate structural changes. These base generating rules handle all of the original cases under the assumption that both operands have initial components with pronounceable elements. We can produce variants lacking x or k with empty π-sets for when one operand or the other has a silent initial component; the condition-preserving maps between the associated DSOs will have empty π-sets and preserve roots otherwise.

We walk through an example in Fig. 4.12 for the case when f is of type $(=, N)$

and g is of type $(base, N)$ where both of the relevant π-sets are nonempty. The map $u_1 : A \to \sigma$ must map $a \mapsto$ *the* since it must map the root to the root; it must map $x \mapsto = n$ since it is the root with respect to the active feature ordering $\lll$. $u_2 : B \to \tau$ must map $b \mapsto$ *boat* since it must map the root to the root; it must map $k \mapsto$ *boat* since this element is also initial with respect to the pronounceable node order $\preceq$; it must map $y \mapsto n$ since this is initial with respect to the active feature ordering $\lll$. If $(j : C \to \zeta, (m_\sigma, m_\tau) : (\sigma, \tau) \to \zeta)$ are to complete this diagram, the images of *the* and *boat* must have the dependency relation $m_\sigma(\text{the}) \leq m_\tau(\text{boat})$ since j must preserve dependency order; we must also have the precedence ordering $m_\sigma(\text{the}) \preceq m_\tau(\text{boat})$ since j must preserve precedence; the components S and T must map into the same component since j must preserve components, and it must be of derived type since j must take derived components to derived components; the images of the two nominal features must be the same element since they are mapped to the same element in C, and their image $m_\sigma(= n) = m_\tau(n)$ must not be in γ_{ST} since $xy \notin \gamma_\kappa$, which 'deletes' the features. In ζ, $\mathbf{n}$ will be of types $(=, N)$ and $(base, N)$. d will be of type $(base, D)$, since we are not forced to change syntactic type. The map j will map $a \mapsto$ *the*, $b \mapsto$ *boat*, $k \mapsto$ *boat* and $xy \mapsto \mathbf{n}$. These maps form a pushout since it is the 'simplest' such maps meeting these requirements.

While writing out the structural changes as pushouts is a bit cumbersome, it is straightforward, though some care should be taken when adding precedence ordering $\preceq$ or chain ordering $\rightharpoonup$. We must make sure we have 'generic representatives' for $\preceq$ and $\rightharpoonup$ 'anchors' (such as the first or last elements) so that we can specify when ordering is added between them.[6] For example, k above was a 'dummy' element which represents the $\preceq$ initial node in the first chain of some expression e, and adding a relation $x \preceq k$ will then add $x \preceq y$ for all π elements y of this first chain. We must create alternate pushouts (which simply make no reference to a pronounced node or node in a particular chain) for variants when they are empty, and no such relation should be added. Continuing in the manner above gives operations on $\mathbb{A}$ comparable to those in BHK [16]. Traditional Minimalist Grammars can be modeled similarly, where the underlying dependency trees are all discrete: the only data used are the feature types, component partitions, and π and γ-set distinction.

4.4 Compilations of Structural Changes

In this section, we explore an analytic benefit of having structured rules. We will study compiling a sequence of n-ary SCs. For example, we may have an operation which merges a specifier but also agrees with it/checks some licensing configuration. We would like to see this single binary operation as a 'compilation' of two others - elementary MERGE *additionally* with an AGREE operation. Similarly, COMPLEMENT-MERGE might be a compilation of MERGE along with selection, while adjunction is just MERGE. This is similar, but not identical, to the theory developed in Hunter [8]. The tools presented here are mainly for analytical purposes - to make sense of how certain SCs can be seen as composed of more primitive ones. However, we do not incorporate this technology into the grammar, we just analyze simple

[6] None of the operations in BHK add ordering relations between active features, so we do not need to worry about similar constructions with $\ll$.

cases where we can see certain rules G as arising from 'compiled' rules.

4.4.1 Categories of Sequences

We first need to connect up the conditions of m rules we wish to compile. We will develop the theory only for **Der**-rules for simplicity, though similar constructions can be carried out in categories of derivations with more structure. Fix some finite sequence $\mathbf{S} = (S, S', \ldots, S^{(m)})$ of subcategories of $\mathbf{Der}^n$. These are to represent the conditions for m many n-ary rules. We define a ***sequence of* S** to be an m-tuple $(A, A', \ldots, A^{(m)})$ with $A^{(i)} \in S^{(i)}$, considered with (n-tuples of) derivation homomorphisms $l^{(i)} : A^{(i)} \to A^{(i-1)}$ for all $m \geq i \geq 1$. We can write a sequence as $A^{(m)} \xrightarrow{l^{(m)}} \ldots \xrightarrow{l'} A$. It is to be thought of as a *generalized parameter*, in that we will use it to combine rules $r^{(i)} : A^{(i)} \to B^{(i)}$, taking the output of $r^{(i)}$, and using $A^{(i+1)}$ to select parameters in it with $r^{(i)}l^{(i+1)} : A^{(i+1)} \to A^{(i)} \to B^{(i)}$.

A ***morphism of sequences of* S** is an m-tuple of (n-tuples of) derivation homomorphisms with $\phi^{(i)} : A^{(i)} \to X^{(i)}$ in $S^{(i)}$, creating a commutative diagram:

$$\begin{tikzcd}
A^{(m)} \arrow[r, "\phi^{(m)}"] \arrow[d, "l^{(m)}"'] & X^{(m)} \arrow[d, "k^{(m)}"] \\
\ldots \arrow[d, "l''"'] & \ldots \arrow[d, "k''"] \\
A' \arrow[r, "\phi'"] \arrow[d, "l'"'] & X' \arrow[d, "k'"] \\
A \arrow[r, "\phi"] & X
\end{tikzcd}$$

We write the category of $\mathbf{S}$-sequences as $\mathbf{S}^*$. Each $S^{(i)}$ is by definition a subcategory of $\mathbf{Der}^n$. We can take the pullback of all these subcategories, writing it $\bigcap \mathbf{S} = \mathcal{C}$. An object in this category is an n-tuple $(\Delta_1, \ldots, \Delta_n)$ which is in each $S^{(i)}$. A morphism in this category is an n-tuple of derivation homomorphisms $(\phi_1, \ldots, \phi_n) : (\Delta_1 \ldots, \Delta_n) \to (\Gamma_1, \ldots, \Gamma_n)$, i.e. a morphism in $\mathbf{Der}^n$, such that this $\mathbf{Der}^n$-morphism is in each $S^{(i)}$. $\mathcal{C} \subset \mathbf{Der}^n$ can be thought of as the subcategory of n-tuples which are in $S^{(i)}$ for all i and preserve $S^{(i)}$-properties for all i. That is, it describes the conditions which are the conjunction of the conditions in $\mathbf{S}$. There is a natural 'diagonal' inclusion functor $\delta : \mathcal{C} \to \mathbf{S}^*$, sending each $(\Delta_1, \ldots, \Delta_n)$ to the $\mathbf{S}$-sequence consisting of only $(\Delta_1, \ldots, \Delta_n)$'s with the identity morphism between each object in the sequence. We intend to compile the m rules into a single rule on the condition category $\mathcal{C}$.

Take any object $A^* \equiv A^{(m)} \to \ldots \to A$ in $\mathbf{S}^*$, and consider the representable functor $\mathbf{S}^*(A^*, -)$. We compose δ with this functor, which gives for each $\mathcal{C}$-object $(\Delta_1, \ldots, \Delta_n)$ the set $\mathbf{S}^*(A^*, \delta(\Delta_1, \ldots, \Delta_n))$. We claim that each morphism in this set is totally determined by its first component, $\phi : A \to (\Delta_1, \ldots, \Delta_n)$. We write ϕ suggestively as l, and notice that there is only one way to make the diagrams commute:

$$\begin{tikzcd}
A^{(m)} \arrow[r, "ll' \ldots l^{(m)}"] \arrow[d, "l^{(m)}"'] & (\Delta_1, \ldots, \Delta_n) \arrow[d] \\
\ldots \arrow[d, "l''"'] & \ldots \arrow[d] \\
A' \arrow[r, "ll'"] \arrow[d, "l'"'] & (\Delta_1, \ldots, \Delta_n) \arrow[d] \\
A \arrow[r, "l"] & (\Delta_1, \ldots, \Delta_n)
\end{tikzcd}$$

Then, we can think of the sequence map $A^* \to \delta(\Delta_1, \ldots, \Delta_n)$ simply as a morphism $l : A \to (\Delta_1, \ldots, \Delta_n)$ in S such that $l \cdots l^{(i)}$ is in $S^{(i)}$ for each i. In other words, this gives a 'spear' of selection of parameters in $(\Delta_1, \ldots, \Delta_n)$ as depicted in the diagram $A^{(m)} \xrightarrow{l^{(m)}} \ldots \to A' \xrightarrow{l'} A \xrightarrow{l} (\Delta_1, \ldots, \Delta_n)$. Looking at the composite of the mappings from $A^{(i)}$ to $(\Delta_1, \ldots, \Delta_n)$ gives an $S^{(i)}$-preserving setting of parameters in $(\Delta_1, \ldots, \Delta_n)$. Since $l : A \to (\Delta_1, \ldots, \Delta_n)$ totally determines all coordinates of the $\mathbf{S}^*$ morphism, we informally write the map of sequences as $l : A^* \to (\Delta_1, \ldots, \Delta_n)$.

Claim 4.4.1. The inclusion $\delta : \mathcal{C} = \bigcap \mathbf{S} \to \mathbf{S}^*$ is always full. That is, for any pair of objects $\Delta, \Gamma \in \mathcal{C}$, the inclusion $\mathcal{C}(\Delta, \Gamma) \to \mathbf{S}^*(\delta\Delta, \delta\Gamma)$ is actually bijective.

Proof. We show the map is surjective. Choose any $l^* : \delta\Delta \to \delta\Gamma$. We know that this is determined by the first map $l : \Delta \to \Gamma$, and hence all coordinates of l^* are l. This is the image of l under δ. □

4.4.2 Sequences of Operations and Compilations

Given a sequence A^* in $\mathbf{S}^*$, we define a *sequence of operations* on A^* to simply be a collection of **FPos** maps $r^{(i)} : \coprod_{1 \leq j \leq n} \top_{A_j^{(i)}} \to B^{(i)}$ which we informally write $r^{(i)} : A^{(i)} \to B^{(i)}$. That is, it is an operation in the same sense we have been using, specified for each object in the sequence. We write the whole sequence of operations informally as $r^* : A^* \to B^*$.

Given a sequence of operations $r^* : A^* \to B^*$ and a setting of parameters $l : A^* \to (\Delta_1, \ldots, \Delta_n)$ from A^* to an object of $\mathcal{C}$, we construct the result of applying the sequence of operations r^* to $(\Delta_1, \ldots, \Delta_n)$ along l as follows:

1. Construct the diagram below and take a universal diagram to push out r along

l:

$$\begin{tikzcd}
\coprod_{i\in n} A_i \arrow[r, "r"] \arrow[d, "+_i l_i"'] & B \arrow[d, dotted] \\
\coprod_{i\in n} \Delta_i \arrow[r, dotted, "\bar{r}"'] & P
\end{tikzcd}$$

This is just the typical translation of the operation r to $(\Delta_1, \ldots, \Delta_n)$ along l.

We write the above diagram in shorthand to reduce clutter:[7]

$$\begin{tikzcd}
A \arrow[r, "r"] \arrow[d, "l"'] & B \arrow[d, dotted, "\bar{l}"] \\
\Delta \arrow[r, dotted, "\bar{r}"'] & P
\end{tikzcd}$$

2. Compose $l'_1 + \ldots + l'_n : A'_1 + \ldots + A'_n \to A_1 + \ldots + A_n$ with r to obtain $r(l') : A'_1 + \ldots + A'_n \to B$,[8] thought of as applying the operation r to the parameter-translation l'. We further compose this with $\bar{l}$ to get a selection of parameters by A' in the output of the first operation acting on Δ:

$$\begin{tikzcd}
 & A' \arrow[r, "r'"] \arrow[d, "r(l')"'] & B' \\
A \arrow[r, "r"] \arrow[d, "l"'] & B \arrow[d, dotted, "\bar{l}"] & \\
\Delta \arrow[r, dotted, "\bar{r}"'] & P &
\end{tikzcd}$$

We then take the pushout of r' and $\bar{l}rl'$ to obtain a new output.

3. After m steps, the process completes, giving a ‘staircase’

[7] The notation A is now ambiguous between $A = (A_1, \ldots, A_n)$ and $\coprod A_i$. However, since B is in $\mathbf{FPos} \hookrightarrow \mathbf{Der}$, not $\mathbf{Der}^n$, the notation $r : A \to B$ disambiguates, indicating we must mean $\coprod A_i$.

[8] We also ambiguously write as l' to mean $l' = (l'_1, \ldots, l'_n)$ or their sum $+_i l'_i$. Again, the notation $r(l')$ disambiguates: r has domain $\coprod A_i$ so we must mean the sum of the coordinates of l'.

$$\begin{tikzcd}
 & & & & A^{(m)} \arrow[r, "r^{(m)}"] \arrow[d, "r^{(m-1)}l^{(m)}"'] & B^{(m)} \arrow[d, dotted] \\
 & & & \dots \arrow[r] & B^{(m-1)} \arrow[r, dotted] \arrow[dddd, dotted] & X^{(m-1)} \arrow[d, dotted] \\
 & & A'' \arrow[r, "r''"] \arrow[d, "r'(l'')"'] & \dots \arrow[rr, dotted] & & \dots \arrow[d, dotted] \\
 & A' \arrow[r, "r'"] \arrow[d, "r(l')"'] & B' \arrow[rrr, dotted] \arrow[dd, dotted] & & & X' \arrow[d, dotted] \\
A \arrow[r, "r"] \arrow[d, "l"'] & B \arrow[rrrr, dotted] \arrow[d, dotted] & & & & X \arrow[d, dotted] \\
\Delta \arrow[r, dotted, "\overline{r}"'] & P \arrow[r, dotted, "\overline{r'}"'] & P' \arrow[r, "\overline{r''}"'] & \dots \arrow[r, "\overline{r^{(m-1)}}"'] & P^{(m-1)} \arrow[r, dotted, "\overline{r^{(m)}}"'] & P^{(m)}
\end{tikzcd}$$

The process seems so ad hoc that it might not even be functorial in $\mathcal{C}$. However, this is not the case. The pushout lemma guarantees that we can ‘paste’ pushout squares together to obtain a pushout square. Repeated applications of the pushout lemma show that we could also have started from the top of the staircase, pushing the vertical map out along the horizontal map sharing its domain. In particular, we may compute $P^{(m)}$ as the pushout of l along a map $A \to \ldots \to X$, the composition of all maps lying even with the first ‘step’ of the staircase once all squares are filled in. We write $\overline{r^*} : A \to X$ for this function, so that $P^{(m)}$ can be computed directly from l and $\overline{r^*}$. We call $\overline{r^*}$ the ***compilation*** of operations $r^* : A^* \to B^*$.

4.4.3 The Rule Generated by a Sequence of Rules

Let $\mathbf{S} = (S, \ldots, S^{(m)})$ be a sequence of subcategories of $\mathbf{Der}^n$. We want to say when a given sequence of operations $r^* : A^* \to B^*$ generates a $\mathcal{C}$-rule $G : \mathcal{C} \to \mathbf{Set}$ when $\mathcal{C} = \bigcap \mathbf{S}$. Let $G : \mathcal{C} \to \mathbf{Set}$ be a $\mathcal{C}$-rule, and let $r^* : A^* \to B^*$ be an $\mathbf{S}$-sequence of operations with compilation $\overline{r^*} : A \to Z$. We construct a functor

$F_r : \mathbf{S}^*(A^*, \delta -) \to G$ as follows. Each element of $\mathbf{S}^*(A^*, \delta(\Delta))$ for a given $\Delta \in \mathcal{C}$ is essentially an S-morphism $l : A \to \Delta$. We take l to an operation $h : \Delta \to P$ if we can construct a pushout diagram of operations:

$$\begin{tikzcd}
A \arrow[r, "l"] \arrow[d, "\overline{r^*}"'] & \Delta \arrow[d, dotted, "h"] \\
Z \arrow[r, dotted] & P
\end{tikzcd}$$

We again say that the rule G is ***generated*** by this sequence of operations $r^* : A^* \to B^*$ iff this natural transformation is epimorphic.

We give a proposition of how rule compilation simplifies when the rule is generated by a sequence of operations $r^{(i)} : A^{(i)} \to B^{(i)}$ where the $A^{(i)}$ are all equal and $A^{(i)} \to A^{(i-1)}$ is the identity for all i. In other words, all the rules to be compiled act on the same object lying in $\bigcap \mathbf{S}$.

Claim 4.4.2. Let $\mathbf{S} = (S, \ldots, S^{(m)})$ be a sequence of subcategories of $\mathbf{Der}^n$, and let $G : \bigcap \mathbf{S} = \mathcal{C} \to \mathbf{Set}$ be a rule generated by the sequence $r^* : A^* \to B^*$. If $A^* = \delta A$ for some $A \in \mathcal{C}$, then G is generated by the compilation $\overline{r^*} : A \to Z \in G(A)$, in the sense that the natural transformation $\tilde{r}^* : \mathbf{y}A \to G$ associated to $\overline{r^*} \in G(A)$ by the Yoneda Lemma is epimorphic.

Proof. This follows from the fullness of the inclusion $\bigcap \mathbf{S} \subset \mathbf{S}^*$. Fullness implies $i : \mathcal{C}(A, -) \to \mathbf{S}^*(\delta A, \delta -)$ with coordinates $i_\Delta : \mathcal{C}(A, \Delta) \to \mathbf{S}^*(\delta A, \delta\Delta)$ mapping $h : A \to \Delta$ to $\delta h : \delta A \to \delta\Delta$ is a natural isomorphism. Furthermore, the composition of i with any epimorphism $\mathbf{S}^*(\delta A, \delta -) \to G$ is an epimorphism, giving a single operation generating G by the Yoneda Lemma. □

4.4.4 A Note on Sequences

While we have constructed compilations of operations in terms of sequences, it may be that what we are really interested in is a family of operations $f^{(i)} : M^{(i)} \to N^{(i)}$ on some other domain $M^{(i)}$, where we apply each operation *to* the coordinate $A^{(i)}$ of the parameter sequence. The reason we use the 'intermediate' step of introducing sequences of parameters A^* separate from these rules is that we may need more elements *in toto* than required by any one of the operations to accommodate them all simultaneously. We introduce the sequence A^* having at least the requisite structure to be able to relate all rules in the family, and then apply the family of operations to the sequence to get a sequence of rules from the family.

Formally, we say that an **S** family of operations is simply a collection of (epimorphic) operations $f^{(i)} : M^{(i)} \to N^{(i)}$ where $M^{(i)}$ is in $S^{(i)}$, with no connecting morphisms. We apply this family to the sequence A^* along family of morphisms $\phi^{(i)} : M^{(i)} \to A^{(i)}$, where each morphism is in $S^{(i)}$ respectively by taking the requisite pushout at each coordinate to obtain operations $r^{(i)} : A^{(i)} \to B^{(i)}$. With all inputs of operations now linked, we can apply the sequence as just described.

4.4.5 Examples

We look at the compilations of some sequences of rules.

Argument selection. We view argument selection as adjunction plus selection. We let (A_1, A_2) be two isomorphic trees each consisting of two elements, $\{a, c\}$ and

$\{b, d\}$, respectively, with dominance relations $a \leq c$ and $b \leq d$. We let r be the SC adding the relation $a \leq b$. We let r' be the SC identifying c and d on these same objects, and we let $(l'_1, l'_2) : (A_1, A_2) \to (A'_1, A'_2)$ be the identity morphism. We can think of r as ADJOIN or PHRASAL-ATTACHMENT, and r' as SELECT or FEATURE-IDENTIFICATION. The compilation given in Fig. 4.13 is the SC which simultaneously adds the relations $a \leq b$ and $c = d$, and can be thought of as typical MERGE involving argument selection.

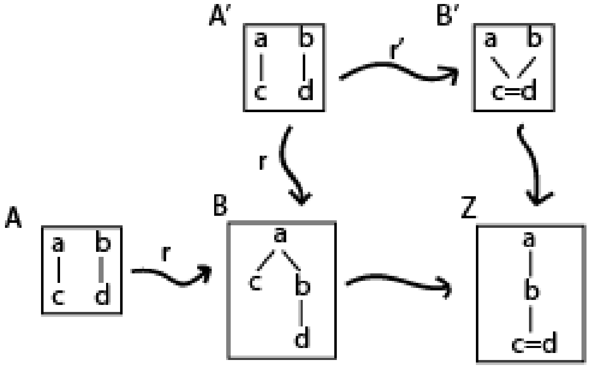


Figure 4.13: Compilation of adjunction and selection. The b phrase is an argument of the a phrase, since it is in its minimal domain, and the selection feature c and selectee d have identified.

Agreeing adjunction. Like argument selection is phrasal attachment followed by feature identification, adjoining a phrase which undergoes agreement with the adjoinee can be seen as PHRASAL-ATTACHMENT followed by FEATURAL-ATTACHMENT. Let (A_1, A_2), (A'_1, A'_2), (l'_1, l'_2), and r be as above. Let r' be the operation adding the relation $d \leq c$. The compilation is the SC which simultaneously adds the relations $a \leq b$ and $d \leq c$. We may want to view licensing, such as by an EPP or wh feature, similarly, with the distinction between agreeing adjuncts and specifiers being about the type of feature involved, if we wish to make such a distinction.

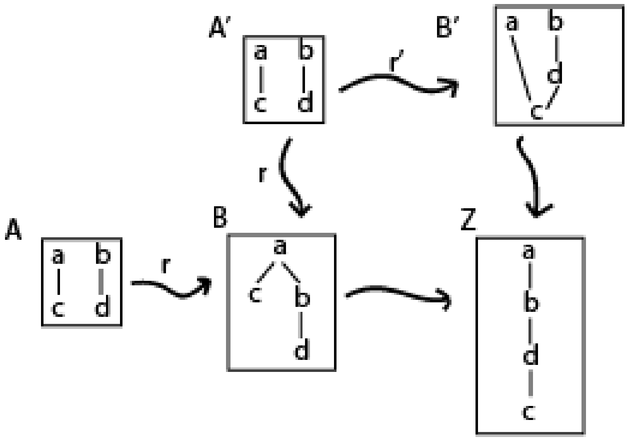


Figure 4.14: Compilation of adjunction and licensing/agreement. The b phrase is an agreeing adjunct/unselected argument of the a phrase, since b is in the minimal domain of a, and a licensing feature of the head a has attached to a feature of b (or more loosely, gone into the domain of b).

‘Pure’ adjunction can be viewed simply as the PHRASAL-ATTACHMENT SC above, involving no manipulations of features. We can then say in what sense ‘agreeing adjunction’, ‘argument selection’, and ‘pure adjunction’ all involve a *phrasal-attachment* component, but also how they differ in terms of what other things they do to features.

4.4.6 Local and Long-Distance Agreement

Suppose Δ and Γ are trees yielding rooted DSOs $\top_\Delta$ and $\top_\Gamma$ with roots d and g, and that L and M are the unique lexical items with roots l and m projecting to d and g. We can construct a SC attaching $g \leq d$, and additionally perform an agreement operation, creating a dependency $y \leq x$ for some $x \in \top_\Delta$ and $y \in \top_\Gamma$

such that x projects from some element in $\top_l$ and y projects from some element in $\top_m$. That is, some feature of the adjoinee becomes dependent on a feature of the adjoiner. We could think of this as *local* agreement - the agreement must take place between two features projecting from the heads of the phrases we are merging at that moment. For example, g could be a tense projection, y an unvalued ϕ-feature, d a determiner projection, and x a ϕ-feature which y will depend on, and hence get its value from.

There are also languages which exhibit long-distance agreement (LDA) phenomena, where a probe, like *tense*, gets a feature valued from some feature in a lower clause. Examples of this, as well a more robust analysis of conditions under which it occurs, are given in Polinsky & Potsdam [37]. We first reproduce an example of local agreement in Hindi-Urdu, whereby the main and auxiliary verbs show agreement with the surface subject. Here, we can think of some probe related to the main and auxiliary verbs *parh-taa thaa* as the probe which will get valued by a gender feature M sitting in the subject phrase *Rahul*.

(1)

Rahul	kitaab	parh-taa	thaa
Rahul.M	book.F	read-Hab.MSg	be.Pst.MSg

‘Rahul used to read (a/the) book.’

Bhatt [38], p. 759

However, when the subject has quirky case marking, the probe gets valued by a gender feature sitting in a lower nominal phrase (or TP). This is shown in the example below.

(2) Vivek-ne [[kitaab parh-nii] chaah-ii]
Vivek-Erg book.F read-Inf.F want.FSg
'Vivek wanted to read the book.'

Bhatt [38], p. 760

Let's suppose that the probe gets its value from the nearest valued gender feature on the nonfinite tense head. We show the SC associated with this select-and-LDA in Fig. 4.15. We will not represent head-adjunction for clarity. In this example, the *tense* head selects the *want* phrase (represented by the v features identifying) while also undergoing LDA with the ϕ-feature of the nonfinite tense head (represented by adding a dependency from the ϕ-feature of *tense* to the ϕ-feature of *inf*).

The SC of the first kind can be obtained as a pushout of the generating SC given in Fig. 4.16. y is the root of the attachee, and ϕ a gender probe, while x is the root of the attaching phrase, and ψ a valued feature which is the goal originating from the same head projecting x. In practice, this generating SC likely also has nodes representing the licensing of the specifier by an EPP feature on the probe, left out for simplicity. We represent a generator for a select-and-LDA rule in Fig. 4.17. The xP is again attached to the root y which contains a probe ϕ. However, the feature ϕ probes for a feature ψ which may be in a zP embedded within the xP. This pushes out to the SC in Fig. 4.15. These generating SCs will be associated with different condition categories, capturing the fact that they have different conditions on application (local or long-distance). We now focus just on the generating SCs to show that, despite this, they do similar things *to* the structure targeted. In

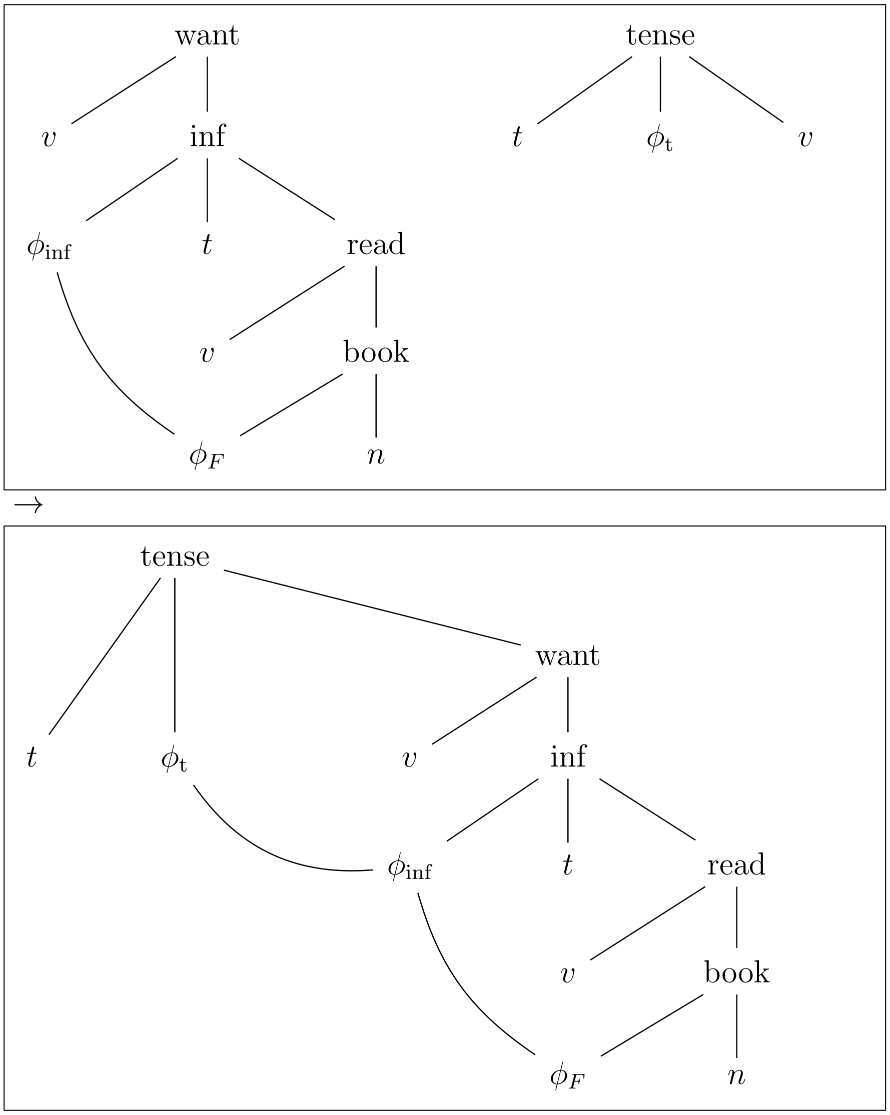


Figure 4.15: Long-distance agreement. *tense* selects the *want* phrase, indicated by the v features identifying. *tense* also undergoes long-distance agreement with the ϕ-feature.

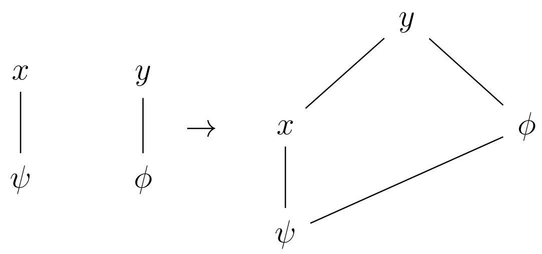


Figure 4.16: A generating SC for phrasal attachment where ψ gets valued by ϕ.

particular, we want to show that both contain an agreement component. Consider the sequence of parameters including the lefthand pair of DSOs in Fig. 4.16 into the lefthand pair of DSOs in Fig. 4.17. We intend to compile the sequence of rules given in Fig. 4.18. The bottom SC in the sequence represents selection, and can be obtained as a pushout from the basic generating selection SC from §4.4.5. The top SC represents the agreement SC from Fig. 4.16. The compilation of this sequence of rules is the SC given in Fig. 4.17. Even if we included the information in the local agreement SC which included specifier licensing, we could see that there is a sequence of rules generating each basic SC, such that in each sequence there is a SC generated by the generating agreement SC in Fig. 4.16. Each rule can then be decomposed to extract out an isomorphic agreement component, though applied in different contexts (local or long-distance). The locality difference can be seen from the fact that the comparison of inputs in the sequence of rules breaks the locality - while we intend x and ψ to originate from the same head in the top SC, they will be taken to elements which do not in the bottom SC, showing that the attaching phrase is dissociated from the agreement-goal in the long-distance case.

We can also compare local and long-distance SCs where the probe *gives* value to some feature in a goal, though it is less clear-cut empirically whether this happens

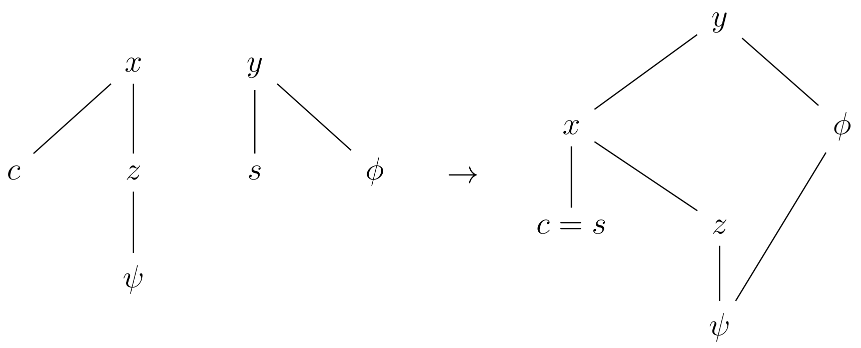


Figure 4.17: A generating SC for selection of xP by y, identifying selection features c and s, while ϕ also targets ψ in a zP for LDA.

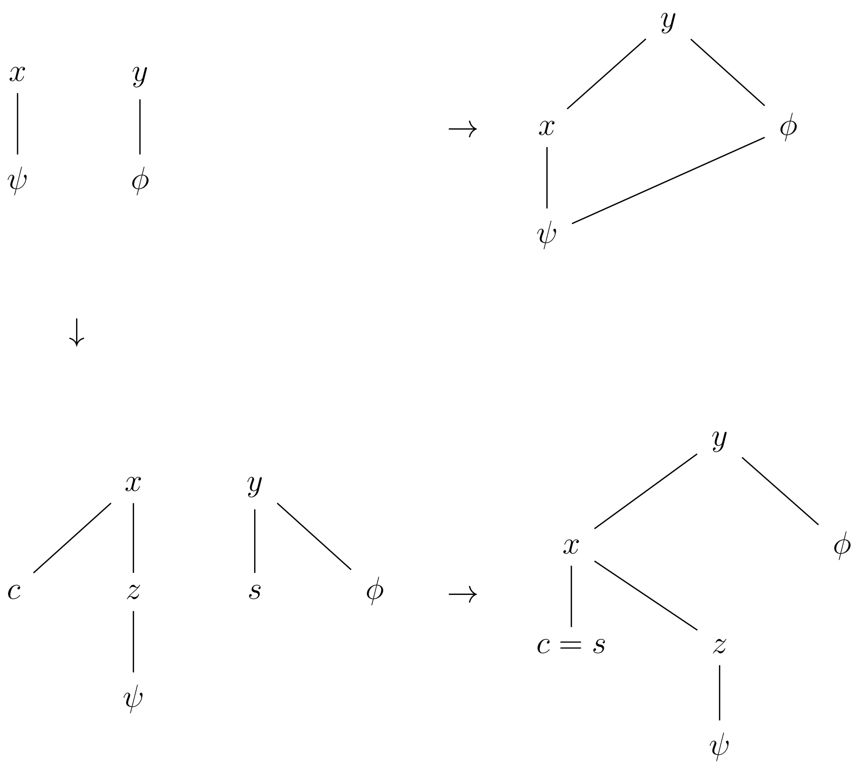


Figure 4.18: A sequence of rules which compiles to a select-and-LDA generating SC.

or not. One such example might be long-distance assignment of nominative to objects in sentences with quirky subjects, outlined in Zaenen, Maling, & Thrainsson [39] and Schütze [40], though neither take this approach.

4.5 Summary

Rules were formalized as assignments of SCs to tuples of input DSOs or derivations. We gave a technique which extracted the *structural analysis* associated to the structure a rule targets for any rule. This was done by constructing a *maximal condition category* whose objects are the tuples a rule can take as inputs and whose morphisms are exactly the morphisms preserving the properties of the structure which were relevant to applying the rule. There is also a natural *repletion* of each such category, which extends a rule to all isomorphic tuples. We then described when a set of basic SCs *generates* all the SCs in a rule. When a rule is generated by a finite set of basic SCs, rules essentially reduce to transformations like those used in graph grammars. We then formalized properties of rules like the Inclusiveness Condition and the Extension Condition in terms of SCs. For the IC, we proved that if a base generating rule is inclusive, the rule it generates will be as well. We made precise what it means for two rules to induce the same structural change in different contexts. This can be modeled by showing that two rules have isomorphic SCs in sequences which compile to their generators. We then sketched a translation of Minimalist Grammars into our model, and gave generators for one of the MERGE rules. We finally showed how to take multiple rules and compile them into a single rule. This generalized the intuition from Hunter [8] that many operations involve a PHRASAL-ATTACHMENT component ‘plus’ some other structural changes. We gave a general theory of these compilations, which extends not only to adjunct-merger versus argument-merger, but also operations which have an agreement or licensing component.

Chapter 5: Movement

5.1 Overview and Background

The SCs considered up to this point are *functions*, and hence assign each node in each input DSO to a single node in the output DSO. Hence, these SCs cannot duplicate structure. Syntactic *movement* is the theory behind many phenomena in language where a phrase seems to be interpreted or have syntactic effects in positions other than where it is pronounced. One model of movement used by mainstream grammar models is given by *copying* - total duplication of some substructure of a DSO, along with *chain* information which relates the copies [6]. This can be contrasted with non-copying approaches to movement, which do not duplicate material at all, but simply delay linearization of a string or tree until it reaches its final position.

5.1.1 Non-Copying Models of Movement

There are non-copying approaches to movement which we can implement with the technology developed already. These methods mimic Minimalist Grammars. Standard Minimalist Grammars do not actually copy elements to represent move-

$$\frac{(\mathbf{s}, x) : +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, (\mathbf{t}, y) : -f, \alpha_{i+1}, \ldots, \alpha_k}{(\mathbf{s} \cup \mathbf{t}, yx) : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k}\mathtt{move1}_{DG}$$

$$\frac{\mathbf{s} \cdot +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, \mathbf{t} : -f\delta, \alpha_{i+1}, \ldots, \alpha_k}{\mathbf{s} : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \mathbf{t} : \delta, \alpha_{i+1}, \ldots, \alpha_k}\mathtt{move2}_{DG}$$

Figure 5.1: Move rules on dependency trees [16]

ment. Rather, if a phrase is merged or moved into a non-final position, it is put into a stack, but not linearized with other elements in the phrase marker. Only at the final position is the string linearized with other elements in the sentence. We demonstrate this with the MOVE rules from the Minimalist Grammar building dependency structures in BHK [16], which we emulated in §4.3.2.

In Fig. 5.1, $\mathtt{move1}_{DG}$ is the mapping used when the moving component $(\mathbf{t}, y) : -f$ has just one feature left, and hence will be moving to its final position, since only feature drive movement in this formalism, and there will be no features remaining. In this case, the component is joined with the initial component, and its elements are linearized with respect to the elements of the initial component. Specifically, this combined component is linearized such that each element from the moved component gets ordered before each element of the head component. No dependencies are affected by this operation. We reproduce an example of a $\mathtt{move1}_{DG}$-style mapping of this kind in Fig. 5.2

When the 'moving' component has more than one feature left, such that deleting the first feature will still leave it with movement features, the initial feature is simply deleted, and the component is not actually combined with any other com-

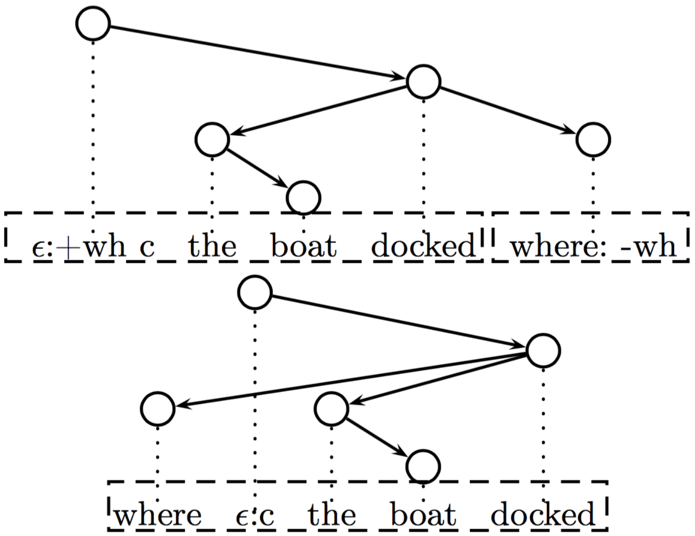


Figure 5.2: A $\texttt{move1}_{DG}$-$\texttt{style}$ mapping [16]

ponent or linearized with anything else.[1] This is the case for mappings fitting the schema in $\texttt{move2}_{DG}$. Similarly, when items are merged and will have features remaining after merger, they are put into a separate component and not linearized.

[1] This raises the question as to what 'successive cyclic' movement (see citations in, e.g., Fox & Pesetsky [41]) looks like in formal minimalist grammars. In early proposals, minimalist grammars manipulated trees [27]. In these models, constituents were put into moved positions explicitly, leaving an empty node in the earlier positions. Later models [22] got rid of this articulated structure entirely, so that only sequences of strings were produced. There are also semi-articulated models, which have 'moving windows' showing local relations like complement and specifier to the current head [8, 11]. In fully articulated minimalist grammars of the first kind, it was possible to express extensions to the grammar to model relativized minimality, phasehood/barriers, etc., though these were largely discarded in later formal minimalist grammars as they were shown to be weakly equivalent to grammars lacking this technology. In particular, the BHK model does not represent a reconfiguration of dependencies when an element is moved, and so, other than the fact that a particular head might trigger deletion of a feature, a moved item in no sense ever sits in a cyclic position.

We could straightforwardly implement this type of movement using SCs of the form we have studied so far, similar to the sketch given in §4.3.2.

5.1.2 True Copying: Kobele 2006

True copying in formal grammars was studied by Kobele [11]. Kobele remarks on a number of issues raised by copying, which we will be able to study more precisely using the technology in this thesis.

> Although we have been speaking loosely of traces as being 'structurally identical' to their antecedents, a moment's reflection should reveal that it is not clear exactly what this should be taken to mean. There are two aspects of structure that might be relevant to us here - the first is the derived structure, what Chomsky [9] calls the structural description of an expression, and the second is what we might call the derivational structure, the derivational process itself. Clearly, 'copying' the derivational structure guarantees identity of derived structure. Kobele [11], p. 138

Kobele also comments on a technical issue: "[W]e need to make sure that the syntactic features that drive the derivation don't get duplicated [. . .] [An] option, if we take the copied structure to include the features, is to reformulate our feature checking operations so as to apply to all chain members whenever any one of them enters into the appropriate configuration." Kobele develops multiple alternative grammars as formal instantiations of each of these approaches in Kobele [11]. We

$$\left(\left\{\begin{array}{l} \text{every} :: \texttt{=n -q} \\ \text{every} :: \texttt{=n -k} \\ \text{every} :: \texttt{=n d} \end{array}\right\}\right)$$

Figure 5.3: An array of 3 copies of `every`, corresponding to the features `d`, `-k`, `-q`. [11]

will focus on ‘true’ copying of structure, both of the derivational type.

We look at Kobele’s ‘true’ copying method as an example for comparison. This method (for the most part) does not copy features. We can tell whether a lexical item is going to trigger movement just by looking at its feature structure. If a lexical item contains a feature of the form $-f$, it will trigger movement. Kobele creates duplicates of a lexical item for each movement and selectee feature when it is initially inserted. For example, when the lexical item `every:: =n d -k -q` is selected, we immediately create 3 copies of it for each of the features `d`, `-k`, and `-q`. The ‘chain relation’ between these copies is represented by putting these three copies in an array, depicted in Fig. 5.3. The features in the array are ordered from bottom to top.

Kobele extends merge to a coordinate-wise merge. If a lexical item is to be merged with the array in Fig. 5.3, it must be triplicated at insertion. In this way, in Kobele [11], copying is done at the beginning of a derivation, not ‘online’. We reproduce Kobele’s example for the formation of the DP *every rotting carcass* which is to occur in three positions - a complement position associated to `d`, a case position associated to `+k`, and another position associated to `+q`, which Kobele introduces for

$$\begin{pmatrix} \{\text{rotting} :: \texttt{=n n}\} \\ \{\text{rotting} :: \texttt{=n n}\} \\ \{\text{rotting} :: \texttt{=n n}\} \end{pmatrix} \qquad \begin{pmatrix} \{\text{carcass} :: \texttt{n}\} \\ \{\text{carcass} :: \texttt{n}\} \\ \{\text{carcass} :: \texttt{n}\} \end{pmatrix}$$

$$\begin{pmatrix} \{(\epsilon,\ \text{rotting},\ \text{carcass}) : \texttt{n}\} \\ \{(\epsilon,\ \text{rotting},\ \text{carcass}) : \texttt{n}\} \\ \{(\epsilon,\ \text{rotting},\ \text{carcass}) : \texttt{n}\} \end{pmatrix}$$

Figure 5.4: Triplicating lexical items to be merged into a phrase which will be copied 3 times. [11]

$$\left(\left\{ \begin{matrix} (\epsilon,\ \text{every},\ \text{rotting carcass}) : \texttt{-q} \\ (\epsilon,\ \text{every},\ \text{rotting carcass}) : \texttt{-k} \\ (\epsilon,\ \text{every},\ \text{rotting carcass}) : \texttt{d} \end{matrix} \right\} \right)$$

Figure 5.5: An array for a derived DP which will be in a 3-part chain. [11]

the semantics of MGs. We must triplicate `carcass::n` and `rotting::=n n`. These can be coordinate-wise merged, which produces a triple of the same expression. These arrays and their merger are depicted in Fig. 5.4.[2] This array can be merged with the array for *every* to give a full DP which will be part of a 3-part chain. We reproduce this in Fig. 5.5.

Each coordinate in the array in Fig. 5.5 will be linearized separately, indicating the ‘copies’ of this DP in the derived object. Kobele derives the sentence *Every rotting carcass arrived*, first merging the DP *every rotting carcass* with `arrive::=d v`. We remove the bottom coordinate in the array, since `=d` will select the `d` feature.

[2] The triples (x, y, z) on the lefthand side of components represent the Specifier, Head, and Complement positions in this Minimalist Grammar, instead of giving a single string.

$$\left(\left\{\begin{array}{l} (\epsilon,\ \textsf{arrive, every rotting carcass}) : \texttt{v}, \\ \hfill \left[\begin{array}{l} (\textsf{every rotting carcass}, -\texttt{q}) \\ (\textsf{every rotting carcass}, -\texttt{k}) \end{array}\right] \end{array}\right\}\right)$$

Figure 5.6: The result of merging *arrive* with the DP. The bottom coordinate of the DP array is removed and linearized. [11]

$$\left(\left\{\begin{array}{l} (\epsilon,\ \epsilon,\ \textsf{arrive -ed every rotting carcass}) : +\texttt{k}\ +\texttt{q}\ \texttt{s}, \\ \hfill \left[\begin{array}{l} (\textsf{every rotting carcass}, -\texttt{q}) \\ (\textsf{every rotting carcass}, -\texttt{k}) \end{array}\right] \end{array}\right\}\right)$$

$$\left(\left\{\begin{array}{l} (\textsf{every rotting carcass},\ \epsilon,\ \textsf{arrive -ed every rotting carcass}) : +\texttt{q}\ \texttt{s}, \\ \hfill \left[\ (\textsf{every rotting carcass}, -\texttt{q})\ \right] \end{array}\right\}\right)$$

$$(\{(\textsf{every rotting carcass every rotting carcass},\ \epsilon,\ \textsf{arrive -ed every rotting carcass}):\textsf{s}\})$$

Figure 5.7: A case feature driving movement from the bottom coordinate of the second component. [11]

The result is reproduced in Fig. 5.6. More functional material is added until a `+k` feature is at the head. We show this step and movement for `+k` and `+q` in Fig. 5.7

The only remaining issue is 'remnant movement' - what to do if a coordinate which will be a target for re-merger has moving components within it. A vP which will be part of a two-part chain is depicted in Fig. 5.8, and this vP contains a DP *John* which will move two more times.[3] The base position will be associated with selection of the feature `v`, and the higher position will be a topic position associated with a `+top` feature. The issue is that when the `v` is selected, the coordinate which

[3] The strikethroughs indicate that that part of the string will not be pronounced. A substring is crossed out immediately if it was linearized by an operation when there were still coordinates left in its associated chain.

$$\left(\left\{\begin{array}{r} (\epsilon,\ \text{rot},\ \text{\sout{John}}) : \texttt{-top}, \begin{bmatrix}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{bmatrix} \\ \\ (\epsilon,\ \text{rot},\ \text{\sout{John}}) : \texttt{v}, \begin{bmatrix}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{bmatrix}\end{array}\right\}\right)$$

Figure 5.8: A vP which will be part of a 2-part chain which has components which will also move. [11]

$$\left(\left\{(\epsilon,\ \text{\sout{rot}},\ \text{\sout{John}}) : \texttt{prog}, \begin{bmatrix}(\text{rot }\text{\sout{John}},\ \texttt{-top})\end{bmatrix}, \begin{bmatrix}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{bmatrix}\right\}\right)$$

Figure 5.9: The result of merging the vP into a complement position which had a moving DP in it. The moving subexpression is deleted. [11]

it belongs to will leave behind a component with moving parts. Kobele stipulates "When a chain link is put into storage, all of its moving subexpressions are eliminated," Kobele [11], p. 169. We merge the vP with a progressive head ϵ`::=v prog`.[4] The bottom coordinate is selected and linearized, and the moving component from that coordinate is deleted immediately. This results in the structure given in Fig. 5.9. More functional material is added until a `+k` feature is at the head. We show this step and movement for `+k` and `+q` in Fig. 5.10. More functional material is added until a `+top` feature is at the head. We show this step and movement for `+top` in Fig. 5.11.

[4] Technically, this head uses a `=>v` feature which triggers head movement immediately instead of regular complement selection, but this does not matter for copying.

$$\left(\left\{(\epsilon,\ \text{will},\ \sout{\text{rot John}}) : \texttt{+k +q s}, \left[\ (\text{rot } \sout{\text{John}},\ \texttt{-top})\ \right], \left[\begin{array}{c}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{array}\right]\right\}\right)$$

$$\left(\left\{(\sout{\text{John}},\ \text{will},\ \sout{\text{rot John}}) : \texttt{+q s}, \left[\ (\text{rot } \sout{\text{John}},\ \texttt{-top})\ \right], \left[\ (\text{John},\ \texttt{-q})\ \right]\right\}\right)$$

$$\left(\left\{(\text{John } \sout{\text{John}},\ \text{will},\ \sout{\text{rot John}}) : \texttt{s}, \left[\ (\text{rot } \sout{\text{John}},\ \texttt{-top})\ \right]\right\}\right)$$

Figure 5.10: Two copies of *John* are re-merged in a higher position. [11]

$$\left(\left\{(\epsilon,\ \epsilon,\ \text{John } \sout{\text{John}} \text{ will } \sout{\text{rot John}}) : \texttt{+top s}, \left[\ (\text{rot } \sout{\text{John}},\ \texttt{-top})\ \right]\right\}\right)$$

$$(\{(\text{rot } \sout{\text{John}},\ \epsilon,\ \text{John } \sout{\text{John}} \text{ will } \sout{\text{rot John}}) : \texttt{s}\})$$

Figure 5.11: vP movement to a topic position. [11]

5.1.3 Copying In a Structured Model

This chapter will focus on copying in a category of structured derivations. Like Kobele [11], we will develop a model with ‘true’ copying and chain-formation, where substructures are multiplied. Unlike Minimalist Grammars in Stabler & Keenan [4] and BHK [16], this will allow us to represent how items in a chain engage in different dependencies in different positions. Unlike Kobele, we will implement a more standard copying model, where copies and chains are generated during the derivation, so elements do not have to be ‘pre-multiplied’ at insertion (and hence we will not need special operations which delete extraneous copies). Summarizing, we give a direct implementation of the mainstream ‘copy, form-chain, re-merge’ technology of Chomsky [6] in the language of structured derivations. We will show how it automatically leads to a system where valuation of a feature in one member of a chain values all copies of it, following one of Kobele's leads. However, we will *derivationally* capture the observation that traces are not necessarily 'structurally identical' to their antecedents, as, throughout the derivation, operations will manipulate the features of moving items.

5.2 Algebras and Chains

Considering a DSO simply as a finite partial order, we represent 'chain-data' as a function $e : T \to T$ from the DSO to itself, which preserves dominance ordering. e takes each element $x \in T$ back to the most recent element it is a copy of, and fixes x if it is a base-copy. See Fig. 5.12 for an example. Given two DSOs-with-chain-data (T, e) and (S, y), we define a chain-preserving morphism to be a morphism of DSOs $f : T \to S$ such that $f \circ e = y \circ f$: that is, if $x \in T$ is a copy of $e(x)$, then $f(x)$ is a copy of $f(e(x))$. This is the category of *1-algebras* over **FPos**. In general, given any category $\mathcal{C}$, the category of 1-algebras over it consists of objects X together with a self-morphism $e : X \to X$, and its morphisms $(X, e) \to (Y, f)$ are morphisms $k : X \to Y$ such that $k \circ e = f \circ k$. We denote this category $\mathbb{T}(\mathcal{C})$.

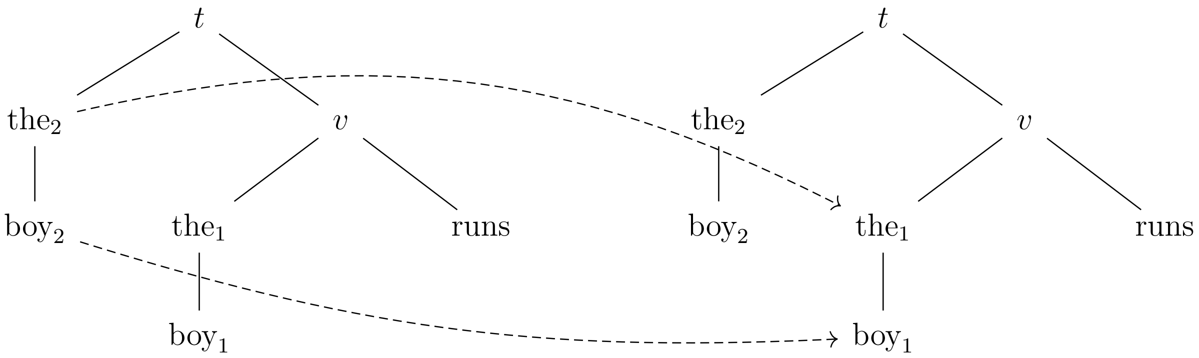


Figure 5.12: Chain-data can be given as a map from a DSO to itself. Copies are taken to elements they were copied from, while all elements where the mapping is not drawn are fixed.

The 1-algebras over **Set** are just sets X together with a unary operation $e : X \to X$, and morphisms are just homomorphisms in the usual sense of model-theory [42]. Let $e : X \to X$ be a function. We describe a set of *fixpoints* $\mathrm{Fix}(e) = \{x \in X \text{ such that } e(x) = x\}$. Given a point $x \in X$, its *orbit* is the set $\mathrm{Orb}(x) = \{y \in X \text{ such that } e^n(x) = y \text{ for some } n \in \mathbb{N}\}$, where $e^n(x)$ is short for $e(e(...e(x)...))$, where e has been applied n times. The orbit of a point naturally forms a preorder $x \lhd e(x) \lhd e(e(x)) \lhd e(e(e(x))) \lhd \ldots$, and we can consider the order relations from all the orbits as a preordering on X. If $k : (X, e) \to (Y, f)$ is a homomorphism, then it preserves the $\lhd$ preordering. We say that a point x is *cyclic* if $e^n(x) = x$ for some $n \in \mathbb{N}$. We say that (X, e) is *acyclic* if the only cyclic elements are the fixpoints. (X, e) is acyclic if and only if $\lhd$ forms a partial order on X. In this case, each orbit is clearly a linear order. The category of acyclic 1-alegbras forms a subcategory $\mathbb{A}(\mathbf{Set}) \hookrightarrow \mathbb{T}(\mathbf{Set})$, and it has a left adjoint. All of these terms and results can be found in Benabou [43]. For our purposes, (X, e) will usually be acyclic, and we read the linear order $x \lhd y$ as 'x is a copy of y'.

When T is a tree, we think of constituents dominated by the minimal elements of $T - \mathrm{Fix}(e)$ as copied constituents. If x is one such minimal element, then the constituent dominated by $e(x)$ is the copy of U_x.

Claim 5.2.1. Suppose T is a tree and $e : T \to T$ an order-preserving function such that for any $x \in T$, we have $e(x) \not< x$ and $x \not< e(x)$. If $x \in T - \mathrm{Fix}(e)$ is minimal amongst the points in that set (i.e. $y \in T - \mathrm{Fix}(e)$ and $y \leq x$ implies $x = y$), then x c-commands $e(x)$.

Proof. Suppose $y < x$. Then $e(y) \leq e(x)$, since e is order-preserving. y is fixed since x is minimal among the non-fixed points, so $e(y) = y \leq e(x)$. □

Copying a constituent $K \subset X$ is straightforward - we just form the coproduct $K + X$. We want to systematically add copy-data while copying this constituent. Let (X, e) be a 1-algebra of finite partial orders, and let $K \subset X$ be a constituent. We form another 1-algebra structure on $K + X$. We have a subset inclusion $i : K \hookrightarrow X$ and self-map $e : X \to X$, and we can sum them to get a map $i + e : K + X \to X$. This map takes each element of the copy K to the element it came from in X, and takes every element of X to whatever it was already a copy of. To turn this into a 1-algebra structure on $K + X$, we compose this function with the coproduct inclusion $\kappa_X : X \hookrightarrow K + X$, giving a map $\kappa_X(i + e) : K + X \to K + X$. If we want to combine two DSOs-with-copy-data without adding any new chain information, we can take their coproduct. Given two 1-algebras of partial orders (X, e) and (Y, f), their coproduct in $\mathbb{T}(\mathbf{FPos})$ is given by $X + Y$ with the map sending $x \mapsto e(x)$ if

$x \in X$ and $y \mapsto f(y)$ if $y \in Y$, which we will denote $e + f$.

We will take a non-copying SC on (X, e) and (Y, f) to be a map $k : (X + Y, e + f) \to (Z, g)$ which is an order-preserving homomorphism of 1-algebras. We can take a SC on (X, e) copying $K \subset X$ to be an order-preserving homomorphism of 1-algebras $k : (K + X, \kappa_X(i + e)) \to (Z, g)$. That is, if we do not copy, we just take the sum as usual, which is equivalent to looking at a pair of maps out of the DSOs-with-copy-data. We can view the process of taking (X, e) with a chosen $K \subset X$ and forming $(K + X, \kappa_X(i + e))$ as the COPY and FORM CHAIN constructions of Chomsky [6]. An n-ary non-copying rule can be described on DSOs-with-copy-data as usual. We define a *copying* rule to be an assignment of sets of non-isomorphic SCs $k : (K + X, \kappa_X(i + e)) \to (Z, g)$ to a 1-algebra of finite partial orders together with a fixed subset inclusion $((X, e), K \subset X)$. That is, the class of objects that a copying rule acts on consists of pairs $((X, e), K \subset X)$, where (X, e) is a 1-algebra of finite partial orders and $K \subset X$ is a subset inclusion. Condition categories for both types of rules can be developed. The process is identical for the non-copying case.

Claim 5.2.2. Let $k : ((X, e), i : K \hookrightarrow X) \to ((Y, f), j : C \hookrightarrow Y)$ be a 1-algebra homomorphism such that there is a function $u : K \to C$ such that the following diagram commutes.

$$\begin{tikzcd}
K \arrow[r, "i"] \arrow[d, "u"'] & X \arrow[d, "k"] \\
C \arrow[r, "j"'] & Y
\end{tikzcd}$$

In this case, $\hat{k} : (K + X, \kappa_X(i + e)) \to (C + Y, \kappa_Y(j + f))$ which maps $x \in K$ to $k(x) \in C$ and $x \in X$ to $k(x) \in Y$ is a 1-algebra morphism.

Proof. $\hat{k}$ is clearly order-preserving. If $x \in K$, then $k(i(x)) = j(k(x))$, since u must just act like k on K. If $x \in X$, then $f(k(x)) = k(e(x))$ since k is a homomorphism. So $\hat{k}$ is a homomorphism. □

We will consider such a morphism k condition-preserving for G if for every G-SC $h : (K + X, \kappa_X(i + e)) \to (Z, g)$, h pushed out along $\hat{k}$ is a G-SC. When generalizing to derivations, we will want to consider derivation morphisms $e : \Delta \to \Delta$.

Claim 5.2.3. The inclusion $i^{\mathbb{T}} : \mathbb{T}(\mathbf{FPos}) \hookrightarrow \mathbb{T}(\mathbf{Der})$ has a left adjoint $\top^{\mathbb{T}} : \mathbb{T}(\mathbf{Der}) \to \mathbb{T}(\mathbf{FPos})$.

Proof. We produce the unit natural transformation $\eta : \mathbf{1} \to i^{\mathbb{T}}\top^{\mathbb{T}}$ from the unit of the adjunction $\top \dashv i$. For (Δ, e), we apply $\top$ to e to obtain $(\top_\Delta, \top_e)$. To see that the regular unit map $\eta_\Delta : (\Delta, e) \to (\top_\Delta, \top_e)$ is in fact a homomorphism, just use its naturality property on the diagram below.

$$\begin{tikzcd}
\Delta \arrow[r, "e"] \arrow[d, "\eta_\Delta"'] & \Delta \arrow[d, "\eta_\Delta"] \\
\top_\Delta \arrow[r, "\top_e"'] & \top_\Delta
\end{tikzcd}$$

□

We will describe a non-copying SC on derivations-with-copy data as usual - a morphism $(\Delta_1 + \ldots + \Delta_n, e_1 + \ldots + e_n) \to (Z, g)$. Non-copying rules will work the same for derivations. We can describe a copying SC on a derivation-with-copy data-with-subderivation $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$, where i is a subderivation inclusion, as a homomorphism $(\mathcal{S} + \Delta, \kappa_\Delta(i + e)) \to (Z, g)$. Particularly important among such derivations-with-subderivations are those where $\mathcal{S}$ is of the form K^{-1} for some $K \subset \top_x$. That is, recalling the construction of §3.5.4, given a sub-DSO in a derivation, we will have an associated coherent subderivation which we can copy. We put off describing condition categories and generation until §5.2.1.

We can also describe extensions for $\mathbb{T}(\mathbf{Der})$ derivations. Given a morphism $h : (\Delta, e) \to (Z, g)$, we can take the usual extension $\mathrm{ext}(h)$. However, (e, g) is also a morphism of operations in the typical sense from $h : \Delta \to Z$ to itself. Hence, we apply the functor ext to it to get a derivation morphism $\mathrm{ext}(e, g) : \mathrm{ext}(h) \to \mathrm{ext}(h)$, obtaining a 1-algebra structure on $\mathrm{ext}(h)$. We write this functor $\mathrm{ext}^C : \mathbb{T}(\mathbf{Der})/\mathbb{T}(\mathbf{FPos}) \to \mathbb{T}(\mathbf{Der})$.

5.2.1 Universal Constructions On Algebras of Derivations

We now compute many universal constructions on 1-algebras of finite partial orders and derivations. It is important to note that in general, pushouts in $\mathbb{T}(\mathcal{C})$ do not coincide with pushouts in $\mathcal{C}$, though by the universal properties we will always have a unique $\mathcal{C}$-map from the pushout in $\mathcal{C}$ to the underlying $\mathcal{C}$-object of the pushout in $\mathbb{T}(\mathcal{C})$. We will see many of examples of this by construction. We start with a simple case, for which we will invoke many representative proof techniques.

Claim 5.2.4. $\mathbb{T}(\mathbf{FSet})$ has pushouts.

We will need the following definitions and lemmas.

Definition 5.2.1. Let X be a set, and $e : X \to X$ a function. We call an equivalence relation $\sim$ on X e-***compatible*** if $x \sim x'$ implies $e(x) \sim e(x')$.

Claim 5.2.5. If $e : X \to X$ is any function, and $k : X \to Y$ any surjective function, then there is at most one 1-algebra structure $f : Y \to Y$ on Y such that

$k : (X, e) \to (Y, f)$ is a 1-algebra homomorphism.

Proof. Let $f : Y \to Y$ be any function such that k is a homomorphism. For any $y \in Y$, we can select some $x \in X$ such that $k(x) = y$. Then $k(e(x)) = f(k(x)) = f(y)$, determining f. □

Claim 5.2.6. For a 1-algebra (X, e), a quotient $q : X \to \tilde{X}$ by an equivalence relation $\sim$ underlies a homomorphism if and only if $\sim$ is e-compatible.

Proof. (e-compatible ⇒ homomorphism) Suppose that $\sim$ is e-compatible. If $\tilde{X}$ has a compatible 1-algebra structure, it must be $\tilde{e} : \tilde{X} \to \tilde{X}$ mapping $[x] \mapsto [e(x)]$, since q is surjective. Since $\sim$ is e-compatible, this is well-defined, since we can choose any $x' \sim x$ and $e(x') \sim e(x)$. (Homomorphism ⇒ e-compatible) Suppose that $q : (X, e) \to (\tilde{X}, \tilde{e})$ is a homomorphism. Then for any $x \sim x'$, we have $q(e(x)) = \tilde{e}(q(x)) = \tilde{e}(q(x')) = q(e(x'))$, and so $e(x) \sim e(x')$. □

Claim 5.2.7. Let $e : X \to X$ be any function, and $R \subset X \times X$ any relation. There is a unique smallest e-compatible equivalence relation on X containing R.

Proof. Note that if $\sim$ and $\approx$ are two e-compatible equivalence relations on X, then the intersection $\sim \cap \approx$ is an e-compatible equivalence relation on X. Also, $=$ is an e-compatible equivalence relation on X, and it contains R. We take the intersection of all e-compatible equivalence relations containing R to obtain the smallest e-compatible equivalence relation containing R. □

These are special cases of standard results about kernels of algebraic homomorphisms. The general case appears in, e.g. Birkhoff [44]. We now prove the initial claim by construction.

Proof. Consider a pair of homomorphisms as in the following diagram.

$$\begin{tikzcd}
(A, x) \arrow[r, "h"] \arrow[d, "g"'] & (B, y) \\
(C, z) &
\end{tikzcd}$$

We first construct the set $B + C$. We call the coproduct inclusions κ_B and κ_C. Since we must have $\kappa_B(b) = \kappa_C(c)$ whenever there is an $a \in A$ such that $b = h(a) = g(a) = c$, we construct a relation R on $B+C$ of points we want to identify. For each $a \in A$, we add the relation $(h(a), g(a))$. We form the 1-algebra $(B+C, y+z)$, and take the smallest $(y + z)$-compatible equivalence relation on $B + C$ containing R. We then construct the quotient map $q : (B + C, y + z) \to ((B \tilde{+} C), (y \tilde{+} z))$. This quotient is the pushout, considered with the two coproduct inclusions followed by composition with the quotient map q. □

We can use these to construct pushouts in $\mathbb{T}(\mathbf{FPos})$. Note that if X is any preorder, and $e : X \to X$ an order-preserving function, then by functoriality, the soberification $s(e) : s(X) \to s(X)$ is order-preserving. Furthermore, the unit quotient map $q : (X, e) \to (s(X), s(e))$ is a homomorphism of 1-algebras of preorders.

Claim 5.2.8. $\mathbb{T}(\mathbf{FPos})$ has pushouts.

Proof. Consider a pair of homomorphisms of 1-algebras of finite partial orders as in the following diagram.

$$\begin{tikzcd}
(A, x) \arrow[r, "h"] \arrow[d, "g"'] & (B, y) \\
(C, z) &
\end{tikzcd}$$

First construct the pushout in $\mathbb{T}(\mathbf{FSet})$ to get $q : (B+C, y+z) \to ((B \tilde{+} C), (y \tilde{+} z))$.

We turn $(B \mathbin{\tilde{+}} C)$ into a preorder by adding relations $q(b) \leq q(b')$ whenever $b \leq b'$ and similarly for relations in C, and taking the smallest preorder containing those relations. However, $(y \mathbin{\tilde{+}} z)$ is not necessarily order-preserving with respect to that ordering. For each $s \leq t$ in $(B \mathbin{\tilde{+}} C)$, we add a relation $(y \mathbin{\tilde{+}} z)(s) \leq (y \mathbin{\tilde{+}} z)(t)$, and take the smallest preorder containing these relations. This produces a new preorder $\leq'$ on $(B \mathbin{\tilde{+}} C)$, though we may have added relations in the closure not preserved by $(y \mathbin{\tilde{+}} z)$. We then add relations of the form $(y \mathbin{\tilde{+}} z)(s) \leq (y \mathbin{\tilde{+}} z)(t)$ whenever $s \leq' t$, and take the smallest preorder containing these relations to obtain $\leq''$. After finitely many steps, this process will stabilize, giving a preorder $\leq^{(n)}$ such that $\leq^{(n+1)} = \leq^{(n)}$. $(y \mathbin{\tilde{+}} z)$ is then order-preserving on $(B \mathbin{\tilde{+}} C)$ with respect to this order. We then take the soberification of this map to get an order-preserving function between partial orders $s(y \mathbin{\tilde{+}} z) : s((B \mathbin{\tilde{+}} C), \leq^{(n)}) \to s((B \mathbin{\tilde{+}} C), \leq^{(n)})$. □

Claim 5.2.9. Given a SC $h : (\Delta, e) \to (Z, s)$ and morphism of 1-algebras of derivations $\phi : (\Delta, e) \to (\Gamma, g)$, there are $k : (\Gamma, g) \to (P, f)$ and $j : (Z, s) \to (P, f)$ which are *universal* among such 1-algebras of finite partial orders and maps into them. See diagram below.

$$\begin{tikzcd}
(\Delta, e) \arrow[r, "\phi"] \arrow[d, "e"'] & (\Gamma, g) \arrow[d, "k"] \arrow[ddr, bend left, "x"] & \\
(Z, s) \arrow[r, "j"'] \arrow[drr, bend right, "y"'] & (P, f) \arrow[dr, dotted, "u"] & \\
& & (Q, r)
\end{tikzcd}$$

We say that k is the *pushout* of e along ϕ by abuse of notation. When we need to distinguish it from a 'real' pushout, we will refer to the square as *(p-hom,d-hom)-universal*.

Proof. These can be obtained by applying $\top$ and taking the pushout in $\mathbb{T}(\mathbf{FPos})$, then precomposing with the unit morphisms. $\square$

We can then straightforwardly construct maximal condition categories for copying and non-copying rules on derivations-with-copy-data. However, generation of rules is less straightforward: we do not want the basic generating SCs to have any copy-data, as we do not usually think of SA being specified in terms of properties of chains/copies of DSOs/derivations, but rather only hierarchical, syntactic type, or other information on the underlying DSO.

Claim 5.2.10. Let $h : \Delta \to Z$ be any operation on a derivation such that the function underlying $\top_h : \top_\Delta \to Z$ is surjective, (Γ, g) any 1-algebra on a derivation, and let $\phi : \Delta \to \Gamma$ be any morphism of derivations. Then there is a 1-algebra on a finite partial order (P, f) together with a 1-algebra morphism $k : (\Gamma, g) \to (P, f)$ and order-preserving function $j : Z \to P$ such that $j \circ e = k \circ \phi$ as morphisms of derivations which is universal with respect to that property. See diagram below

$$\begin{tikzcd}
\Delta \arrow[r, "\phi"] \arrow[d, "e"'] & (\Gamma, g) \arrow[d, "k"] \arrow[ddr, bend left, "x"] & \\
Z \arrow[r, "j"'] \arrow[drr, bend right, "y"'] & (P, f) \arrow[dr, dotted, "u"] & \\
& & (Q, r)
\end{tikzcd}$$

We call k the *pushout* of e along ϕ by abuse of notation. When we need to distinguish this construction from a usual pushout, we will say that this diagram is *(poset,d-hom)-universal.*

Proof. To compute the pushout, first apply $\top$ to the diagram, and take the pushout of finite partial orders to obtain an order-preserving function $e' : \top_\Gamma \to S$. Since e

is surjective, e' is surjective, since epimorphisms are preserved under pushout. We form a relation $x \sim y$ if $e'(x) = e'(y)$ on $\top_\Gamma$. We take the smallest g-compatible equivalence relation containing $\sim$, call it $\equiv$, and take the quotient $q : \top_\Gamma \to \top_\Gamma / \equiv$. q induces a preordering on $\top_\Gamma / \equiv$ and also a function $g' : \Gamma / \equiv \to \Gamma / \equiv$ on it. However, g' is not necessarily order-preserving with respect to the induced ordering. We iteratively add order relations to $\top_\Gamma / \equiv$ until the process stabilizes, as in the proof of Claim 5.2.8, finally obtaining an order-preserving map of 1-algebras of *preorders* $q : (\top_\Gamma, \top_g) \to (\top_\Gamma / \equiv, g')$. We then apply the soberification functor to g' to obtain a 1-algebras of partial orders $(s(\top_\Gamma / \equiv), s(g'))$ which we write (P, f), and we compose q with the quotient to obtain a map $q' : (\top_\Gamma, \top_g) \to (P, f)$. Since $e' : \top_\Gamma \to S$ is the pushout of orders, this function factors through e' uniquely as $r : S \to P$. From the pushout of orders, we have a map $v : Z \to S$ which we compose with r to get a map $j : Z \to P$. We precompose q' with the unit map $\eta_{(\Gamma,g)} : (\Gamma, g) \to (\top_\Gamma, \top_g)$ to obtain a homomorphism of 1-algebras of derivations $k : (\Gamma, g) \to (P, f)$. (P, f), j and k as constructed give a pushout in the sense above. □

This shows that when an operation $h : \Delta \to Z$ is inclusive, and we have a translation ϕ from Δ to the derivation underlying a derivation-with-copy-data (Γ, g), we can do h in the context ϕ by finding the best morphism *preserving copy data* out of (Γ, g) which completes the diagram. The reason we require h to be inclusive is that only $(\top_\Gamma, \top_g)$ has information about copy data: if we introduced new points from Z which were not identified with points in $\top_\Gamma$, then we would not know what elements they should be copies of.

If we are to use such operations for generation of rules on derivations-with-copy data, then we must make sure that we can compose them with pushouts of operations on 1-algebras of derivations to obtain another pushout. That is, we must verify the following claim.

Claim 5.2.11. Suppose the left square is (poset,d-hom)-universal. Then the right square is a (p-hom,d-hom)-universal if and only if the whole square is (poset,d-hom)-universal.

$$\begin{tikzcd}
\mathcal{M} \arrow[r, "k"] \arrow[d, "r"'] & (\Delta, e) \arrow[r, "\phi\ (\text{hom})"] \arrow[d, "h\ (\text{hom})"] & (\Gamma, g) \arrow[d, "h'\ (\text{hom})"] \\
X \arrow[r, "g"'] & (Z, s) \arrow[r, "f\ (\text{hom})"'] & (Q, s)
\end{tikzcd}$$

Proof. Identical to the proof of the regular Pushout Lemma, using the universal properties of each square. □

We now give a precise statement of copying and non-copying rules on derivations-with-copy data in preparation for a precise statement of generation by a basic SC.

Definition 5.2.2. We define $\mathbb{T}^s(\mathbf{Der})$ to be the category whose objects are 1-algebras of derivations, together with a subderivation inclusion. A morphism k : $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to ((\Gamma, f), j : \mathcal{T} \hookrightarrow \Delta)$ is a morphism of derivations $k : \Delta \to \Gamma$ such that $k \circ e = f \circ k$, such that there is a (necessarily unique) morphism $u : \mathcal{S} \to \mathcal{T}$ such that $k \circ i = j \circ u$. u will just be k restricted to $\mathcal{S}$ and $\mathcal{T}$. This category can be thought of as the category of derivations with copy data, together with a specified subpart which is to be copied.

Definition 5.2.3. We define a functor FORM-CHAIN : $\mathbb{T}^s(\mathbf{Der}) \to \mathbb{T}(\mathbf{Der})$ which takes an object $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$ to the derivation $\mathcal{S} + \Delta$ together with the self-map $\kappa_\Delta \circ (i + e) : \mathcal{S} + \Delta \to \Delta \hookrightarrow \mathcal{S} + \Delta$. Given a morphism $k : ((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to ((\Gamma, f), j : \mathcal{T} \hookrightarrow \Delta)$, we get a $\mathbb{T}(\mathbf{Der})$-morphism $\textsc{form-chain}(k) : (\mathcal{S} + \Delta, \kappa_\Delta(i + e)) \to (\mathcal{T} + \Gamma, \kappa_\Gamma(j + f))$ which maps $x \in \mathcal{S}$ to $u(x) \in \mathcal{T}$ and $x \in \Delta$ to $k(x) \in \Gamma$).

The proof that this gives a functor is just like the proof of Claim 5.2.2, and will be proven diagrammatically in the proof of Claim 5.2.13.

Definition 5.2.4. A ***copying rule*** is an assignment G which takes in objects $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$ of $\mathbb{T}^s(\mathbf{Der})$ and returns a set of non-isomorphic homomorphisms of 1-algebras of derivations $h : \textsc{form-chain}((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to (Z, g)$. We let $\mathcal{E}_G \subset \mathbb{T}^s(\mathbf{Der})$ be the subclass of objects where G is defined. G can be turned into a functor by taking its ***maximal condition category***. We let a $\mathbb{T}^s(\mathbf{Der})$-morphism $k : ((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to ((\Gamma, f), j : \mathcal{T} \hookrightarrow \Delta)$ between two objects in $\mathcal{E}_G$ be a morphism of $\mathcal{E}_G$ if for every G-SC h on $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$, the pushout of h along $\textsc{form-chain}(k)$ is a G-SC on $((\Gamma, f), j : \mathcal{T} \hookrightarrow \Delta)$.

Definition 5.2.5. A ***(non-copying) n-ary rule*** is an assignment G which takes in n-tuples of $\mathbb{T}(\mathbf{Der})$ objects $((\Delta_1, e_1), \ldots, (\Delta_n, e_n))$ and returns a set of tuples of non-isomorphic homomorphisms of 1-algebras of derivations $h_i : (\Delta_i, e_i) \to (Z, g)$, or equivalently single morphisms $h : \coprod(\Delta_i, e_i) \to (Z, g)$. We let $\mathcal{E}_G \subset \mathbb{T}(\mathbf{Der})^n$ be the subclass of tuples where G is defined. G can be turned into a functor by taking its ***maximal condition category***. We let a $\mathbb{T}(\mathbf{Der})^n$-morphism $k : ((\Delta_1, e_1), \ldots, (\Delta_n, e_n)) \to ((\Gamma_1, f_1), \ldots, (\Gamma_n, f_n))$ between two objects in $\mathcal{E}_G$ be a morphism in $\mathcal{E}_G$ if for every G-SC h on $((\Delta_1, e_1), \ldots, (\Delta_n, e_n))$, the pushout of h along $\coprod k$ is a G-SC on $((\Gamma_1, f_1), \ldots, (\Gamma_n, f_n))$.

We now want to describe generation of rules on derivations-with-copy-data. The main problem is that our usual definition required us to use an object $\mathcal{E}_G$ together with an SC on it to generate the rule, and to 'translate' this SC along an $\mathcal{E}_G$ morphism. Suppose, for example, that we are trying to generate a binary non-copying rule using an operation $(\Delta_1 + \Delta_2, e_1 + e_2) \to (Z, g)$. The generating SC has copy-data on it, and parameter-settings $k : ((\Delta_1, e_1), (\Delta_2, e_2)) \to ((\Gamma_1, f_1), (\Gamma_2, f_2))$ must preserve this copy data. But, even in the simple case where our generating objects have the trivial identity functions for copy data, they will not be able to target anything that is a copy, since k must preserve copy data. This is why we are usually interested in generating a rule using an object *without* copy data. We showed in Claim 5.2.10 that it is sometimes possible to find maps which preserve copy data completing a diagram of morphisms, not all of which preserve copy data, which are universal with respect to this property. Specifically, given an inclusive operation $h : \Delta \to Z$ without copy data, a derivation with copy data (Γ, g), and a morphism $\phi : \Delta \to \Gamma$, there is a reasonable way to universally push h out along ϕ to obtain a map $h' : (\Gamma, g) \to (Z, t)$, adding the restriction that the pushout must preserve copy data. Further, if we take the pushout of h along ϕ to obtain h', then take the pushout of h' along $\psi : (\Gamma, g) \to (\Xi, x)$ to obtain h'', this is just the pushout of h along $\psi \circ \phi$. So this will *functorially* construct operations on derivations-with-copy data, and we have hope of describing generation by a SC without copy data.

There is a remaining problem, however. Which maps $\phi : \Delta \to \Gamma$ should we consider? We cannot simply use maps from $\mathcal{E}_G$, since these include information about copy data, and Δ, even if it arises as the underlying derivation of some derivation-with-copy data, likely will not correspond uniquely to one. We will, for simplicity, allow ourselves to stipulate which morphisms we should use. To do this, we first need to make precise functors which link derivations with and without copy data.

Claim 5.2.12. We define ${}^s\mathbf{Der}$ to be the category whose objects pairs $(\Delta, i : \mathcal{S} \hookrightarrow \Delta)$ of derivations together with a subderivation embedding. A morphism $\phi : (\Delta, i : \mathcal{S} \hookrightarrow \Delta) \to (\Gamma, j : \mathcal{T} \hookrightarrow \Gamma)$ is a morphism $\phi : \Delta \to \Gamma$ such that there is a (necessarily unique) $u : \mathcal{S} \to \mathcal{T}$ such that $\phi \circ i = j \circ u$. u must just be ϕ restricted to $\mathcal{S}$ taking values in the subset underlying $\mathcal{T}$.

There is a faithful functor ${}^sV : \mathbb{T}^s(\mathbf{Der}) \to {}^s\mathbf{Der}$ taking $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \mapsto (\Delta, i : \mathcal{S} \hookrightarrow \Delta)$ which maps each $\mathbb{T}^s(\mathbf{Der})$-morphism to itself.

There is a faithful functor $V^n : \mathbb{T}(\mathbf{Der})^n \to \mathbf{Der}^n$ mapping $((\Delta_1, e_1), \ldots, (\Delta_n, e_n)) \mapsto (\Delta_1, \ldots, \Delta_n)$ and each n-tuple of homomorphisms of 1-algebras of derivations to itself.

There is a functor COPY : ${}^s\mathbf{Der} \to \mathbf{Der}$ mapping $(\Delta, i : \mathcal{S} \hookrightarrow \Delta) \mapsto \mathcal{S} + \Delta$ such that $\text{COPY} \circ {}^sV = V \circ \text{FORM-CHAIN}$.

We first describe generation for non-copying rules. Consider $V_{\mathcal{E}} : \mathcal{E}_G \hookrightarrow \mathbb{T}(\mathbf{Der})^n \to \mathbf{Der}^n$. We call a subcategory $i : \mathcal{D} \hookrightarrow \mathbf{Der}^n$ a ***lift*** of $V_{\mathcal{E}}$, if there exists a (necessarily unique) morphism $W : \mathcal{E}_G \to \mathcal{D}$ such that $i \circ W = V_{\mathcal{E}}$. W is necessarily the functor acting just like $V_{\mathcal{E}}$ restricted to $\mathcal{E}_G$. We are going to use

$\mathcal{D}$ to specify which morphisms $\Delta \to V_{\mathcal{E}}(\Gamma, g)$ we want to push a base generating SC $r : \coprod \Delta \to Z$ out along. We say that a base generating SC $r : \coprod \Delta \to Z$ *generates* a non-copying rule $G : \mathcal{E}_G \to \mathbf{Set}$ if there is a natural transformation $\tilde{r} : \mathcal{D}(\Delta, V_{\mathcal{E}}-) \to G$ with coordinates $\tilde{r}_{(\Gamma,g)} : \mathcal{D}(\Delta, V_{\mathcal{E}}(\Gamma, g)) \to G(\Gamma, g)$ which map a $\mathcal{D}$-morphism $\phi : \Delta \to \Gamma$ to the (poset,d-hom)-universal pushout of r along $\coprod \phi$, which is epimorphic. That such universal constructions give rise to a natural transformation when they exist, as is always the case when r is inclusive, is guaranteed by Claims 5.2.10 and 5.2.11. Again, $\mathcal{D}, \mathcal{E}_G$, and r determine G uniquely up to natural isomorphism.

We can proceed similarly for copying rules. Consider $V_{\mathcal{E}} : \mathcal{E}_G \hookrightarrow \mathbb{T}^s(\mathbf{Der}) \to {}^s\mathbf{Der}$. We call a subcategory $i : \mathcal{D} \hookrightarrow {}^s\mathbf{Der}$ a ***lift*** of $V_{\mathcal{E}}$, if there exists a (necessarily unique) morphism $W : \mathcal{E}_G \to \mathcal{D}$ such that $i \circ W = V_{\mathcal{E}}$. W is necessarily the functor acting just like $V_{\mathcal{E}}$ restricted to $\mathcal{E}_G$. We are going to use $\mathcal{D}$ to specify which morphisms $(\Delta, i : \mathcal{S} \hookrightarrow \Delta) \to V_{\mathcal{E}}((\Gamma, g), j : \mathcal{T} \hookrightarrow \Gamma)$ we want to push a base generating SC $r : \text{COPY}(\Delta, i : \mathcal{S} \hookrightarrow \Delta) \to Z$ out along. We say that a base generating SC $r : \text{COPY}(\Delta, i : \mathcal{S} \hookrightarrow \Delta) \to Z$ *generates* a copying rule $G : \mathcal{E}_G \to \mathbf{Set}$ if there is a natural transformation $\tilde{r} : \mathcal{D}((\Delta, i : \mathcal{S} \hookrightarrow \Delta), V_{\mathcal{E}}-) \to G$ with coordinates $\tilde{r}_{(\Gamma,g)} : \mathcal{D}((\Delta, i : \mathcal{S} \hookrightarrow \Delta), V_{\mathcal{E}}((\Gamma, g), j : \mathcal{T} \hookrightarrow \Gamma)) \to G((\Gamma, g), j : \mathcal{T} \hookrightarrow \Gamma)$ which map a $\mathcal{D}$-morphism $\phi : (\Delta, i : \mathcal{S} \hookrightarrow \Delta) \to (\Gamma, j : \mathcal{T} \hookrightarrow \Gamma)$ to the (poset,d-hom)-universal pushout of r along $\text{COPY}(\phi)$, which is epimorphic. Note that generating rules with objects and morphisms of ${}^s\mathbf{Der}$ will allow us to control properties of where the substructure to be copied may be taken from, but ignores existing copy structure on the input.

Using this technology we can describe when movement and non-movement variants of a rule are similar in a manner similar to §4.4.6. Specifically, two rules of different types may have isomorphic generating operations, or isomorphic operations which push out to an SC in a sequence of SCs compiling to a generator of each rule.

5.2.2 Grammars with Copying

We can describe grammars on derivations with copy data. The base lexical items $\textsc{lex}$ will be a set of objects of $\mathbb{T}(\mathbf{FPos})$ and we usually specify that they all have trivial chain data - that is, they are all of the form $(P, 1_P)$. $\textsc{rules}$ will be separated into two kinds, copy and non-copying.

Definition 5.2.6. Given a grammar $\mathscr{G} = (\textsc{lex}, \textsc{rules})$, we describe the ***language*** $\mathscr{L}(\mathscr{G})$ ***generated by*** $\mathscr{G}$.

- If $(P, f) \in \textsc{lex}$, then $(P, f) \in \mathscr{L}(\mathscr{G})$.

- If G is a non-copying rule in $\textsc{rules}$, $(\Delta_i, e_i) \in \mathscr{L}(\mathscr{G})$, and $h : \coprod(\Delta_i, e_i) \to (Z, s) \in G((\Delta_1, e_1), \ldots, (\Delta_n, e_n))$, then $\text{ext}^C(h) \equiv (\text{ext}(h), \text{ext}(\coprod e_i, s)) \in \mathscr{L}(\mathscr{G})$.

- If G is a copying rule in $\textsc{rules}$, $(\Delta, e) \in \mathscr{L}(\mathscr{G})$, and $h : \textsc{form-chain}((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to (Z, s) \in G((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$, then $\text{ext}^C(h) \in \mathscr{L}(\mathscr{G})$.

We can consider such languages as full subcategories of $\mathbb{T}(\mathbf{Der})$, and describe equivalences of languages identically to how we did in §3.5.5.

5.2.3 Adding Structure to Grammars with Copying

We can add copy data systematically to any extendable category of $\mathbf{A}$-derivations to obtain an extendable category of $\mathbf{A}$-derivations with copy data. However, there are a few technical caveats right away. Given a subset $S \subset \Delta$ of the set underlying an $\mathbf{A}$-derivation Δ, it is not guaranteed that a substructure exists on it, unlike the case with regular derivations. We will then have to take care to make sure that our rules target actual substructures. Also, as with basic $\mathbf{A}$-derivations, not all pushouts/universal constructions may exist, especially once we have added the 1-algebra structure, though they can be defined in general.

Let $(U^{\mathbf{A}} : \mathbf{A} \to \mathbf{FPos}, i^{\mathbf{A}} : \mathbf{A} \to \mathbf{ADer})$ give the data for a category of extendable $\mathbf{A}$-derivations. We get an induced $i^{\mathbb{T}\mathbf{A}} : \mathbb{T}(\mathbf{A}) \to \mathbb{T}(\mathbf{ADer})$ which takes (P, f) to $(i^{\mathbf{A}}(P), i^{\mathbf{A}}(f))$. We can construct the left adjoint using $\top^{\mathbf{A}}$. Given a 1-algebra of $\mathbf{A}$-derivations (Δ, e) to the morphism e, we get a 1-algebra of $\mathbf{A}$-objects $(\top_\Delta, \top_e)$. To get the components of the unit, we just use the naturality of the unit for $\top^{\mathbf{A}} \dashv i^{\mathbf{A}}$.

The above proof shows that we can often emulate proofs and constructions from $\mathbb{T}(\mathbf{Der})$ in $\mathbb{T}(\mathbf{ADer})$ totally diagrammatically. For example, we construct $\mathbb{T}^s(\mathbf{ADer})$ as the category of 1-algebras of $\mathbf{A}$-derivations together with an embedding of $\mathbf{A}$-derivations $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$, whose morphisms $k : ((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to ((\Gamma, g), j : \mathcal{T} \hookrightarrow \Gamma)$ are homomorphisms of 1-algebras of $\mathbf{A}$-derivations $k : (\Delta, e) \to (\Gamma, g)$ such that there exists as (necessarily unique) $u : \mathcal{S} \to \mathcal{T}$ such that $k \circ i = j \circ u$. u is necessarily k restricted to $\mathcal{S}$. It is straightforward to prove the following claim diagrammatically.

Claim 5.2.13. The map FORM-CHAIN : $\mathbb{T}^s(\mathbf{ADer}) \to \mathbb{T}(\mathbf{ADer})$ sending $((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta)$ to the derivation $\mathcal{S} + \Delta$ together with the self-map $\kappa_\Delta \circ (i+e) : \mathcal{S} + \Delta \to \Delta \hookrightarrow \mathcal{S} + \Delta$, and a morphism $k : ((\Delta, e), i : \mathcal{S} \hookrightarrow \Delta) \to ((\Gamma, f), j : \mathcal{T} \hookrightarrow \Delta)$ to the $\mathbb{T}(\mathbf{Der})$-morphism FORM-CHAIN$(k) : (\mathcal{S} + \Delta, \kappa_\Delta(i+e)) \to (\mathcal{T} + \Gamma, \kappa_\Gamma(j+f))$ which maps $x \in \mathcal{S}$ to $u(x) \in \mathcal{T}$ and $x \in \Delta$ to $k(x) \in \Gamma$), is a functor.

Proof. The following diagram commutes by hypothesis on k.

$$\begin{tikzcd}
\mathcal{S} \arrow[dr, "i"] \arrow[ddd, bend right, "u"'] & \\
& \Delta \arrow[ddd, "k"] \\
\Delta \arrow[ur, "e"'] \arrow[ddd, bend right, "k"'] & \\
\mathcal{T} \arrow[dr, "j"] & \\
& \Gamma \\
\Gamma \arrow[ur, "f"'] &
\end{tikzcd}$$

So the following diagram commutes.

$$\begin{tikzcd}
\mathcal{S} + \Delta \arrow[r, "i+e"] \arrow[d, "u+k"'] & \Delta \arrow[d, "k"] \\
\mathcal{T} + \Gamma \arrow[r, "j+f"'] & \Gamma
\end{tikzcd}$$

We can sum the right vertical map with u using the coprojections κ_Δ and κ_Γ, to obtain the following commutative diagram.

$$\begin{tikzcd}
\mathcal{S} + \Delta \arrow[r, "i+e"] \arrow[d, "u+k"'] & \Delta \arrow[r, "\kappa_\Delta"] \arrow[d, "k"] & \mathcal{S} + \Delta \arrow[d, "u+k"] \\
\mathcal{T} + \Gamma \arrow[r, "j+f"'] & \Gamma \arrow[r, "\kappa_\Gamma"'] & \mathcal{T} + \Gamma
\end{tikzcd}$$

The horizontal maps give the 1-algebra structures on $\mathcal{S} + \Delta$ and $\mathcal{T} + \Gamma$, and commutivity shows that FORM-CHAIN$(k) = u + k$ is a homomorphism of 1-algebras.

□

Similarly, $\mathrm{ext}^{\mathbf{A}}$ extends to an extension functor keeping track of copy data $\mathrm{ext}^{\mathbf{A},C} : \mathbb{T}(\mathbf{ADer})/\mathbb{T}(\mathbf{A}) \to \mathbb{T}(\mathbf{ADer})$ by taking an SC $h : (\Delta, e) \to (P, f)$ to $(\mathrm{ext}^{\mathbf{A}}(h), \mathrm{ext}^{\mathbf{A}}(e, f))$, since (e, f) can be considered as a morphism of operations from $h : \Delta \to P$ to itself. Coproducts will work identically to coproducts in **ADer**. Since we have a yield functor, coproducts, extensions, and a FORM-CHAIN functor, we can recursively construct languages from grammars just as with regular derivations.

5.3 Example: Greek Case Concord

We have already mentioned that in general, a pushout in $\mathbb{T}(\mathcal{C})$ does not necessarily have the pushout in $\mathcal{C}$ as its underlying object. We will see that this is actually a desirable effect for modeling certain phenomena involving chains. Primarily, we will see that adding 1-algebra data will affect all lower copies in a chain when affecting one part, partially addressing Kobele's comment about duplicating features which drive the derivation. This aligns with our theme that interesting properties of structure are captured by morphisms: 1-algebra data describes chains on an object, while universal constructions using morphisms preserving 1-algebra structure percolate structural change effects throughout chains. Consider the following delayed-agreement effect from Ancient Greek, based on Andrews [45].

(3)

emmenomen	hois	ho:mologe:samen	dikaiois	ousz,	e:	ou?
we.abide.by	which.DAT	we.have.agreed	just.DAT	being.DAT	or	not

'Do we abide by those things which we consider just, or not?'

The important fact is that the adjective *dikaiois* and participle *ousz* both engage in case-concord with hois. However, *hois* does not get DAT case until it

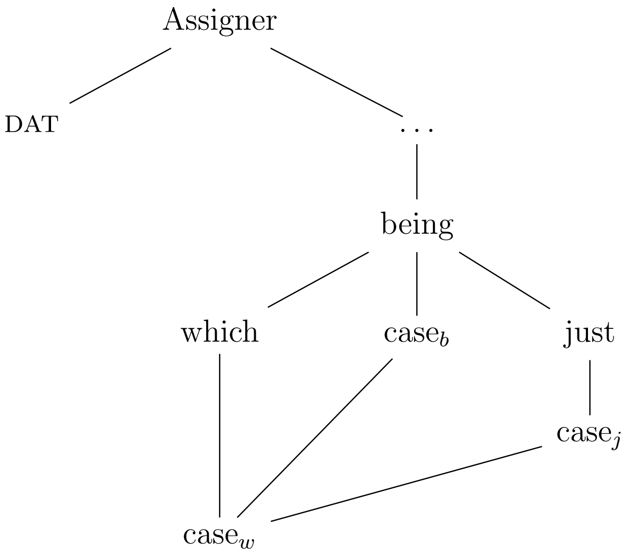


Figure 5.13: A feature-sharing structure representing concord, where the case of the participle *being* and adjective *just* depend on the case of *which*.

moves into the matrix clause for assignment from *emmenomen*. The idea is that in a feature-sharing configuration, after undergoing concord with the head wh-item, when the wh-item gets valued for case, the lower items match 'automatically'.

For our purposes, it does not matter if we believe that one or the other of the participle or adjective engages in case concord with the other, and then in turn with the wh-item, of if they both simply enter into concord with the wh-item directly. We assume the latter for simplicity. We assume that we have built up the structure in Fig. 5.13. Call this structure T, and we will assume for simplicity that no copying has yet taken place, such that the chain-data is simply the identity function on T.

We intend to copy the constituent 'which $\leq$ case$_w$' (denoted K) and attach it to Assigner (i.e. add the relation Assigner $\leq$ which), while also valuing its case (i.e. adding the relation case$_w$ $\leq$ DAT). However, if we take the 'normal' pushout, the induced copy data self-map is not order-preserving. If the map from $(K + T, \kappa_T(i +$

id)) to Z must meet the SC requirements for copying, then the induced 1-algebra operation on Z to itself must take the higher copy of *which* to the lower copy, and similarly for *case*$_w$. Denoting these higher copies with a $'$ symbol and the induced self-map $g : Z \to Z$, we have $\text{case}'_w \leq \textsc{dat}'$, yet $g(\text{case}'_w) = \text{case}_w \not\leq \textsc{dat} = g(\textsc{dat})$. The (poset,d-hom)-universal map will simultaneously add a relation from the base copy of *case*$_w$ to DAT. See Fig. 5.14.

Similar logic can be used to also explain the simpler observation - when one object in a chain gets Case, the whole chain gets Case. The explanation of this and the more complex phenomenon follow straightforwardly from this model, assuming that case may get its value under the transitivity of dominance/dependence. Whenever we add a dependency relation $a \leq b$ to some element a which is a copy of $e(a)$, the universal construction preserving 1-algebra structure will also add a dependency relation $e(a) \leq b$, so that e is order-preserving (assuming that b is a base-copy).[5] We can also extend this sort of logic when the DSOs have more structure. We start with a simple example involving feature activity. Let $\mathbf{A}$ be the category of finite partial orders together with a single predicate ι 'inactive'. That is, if X is an $\mathbf{A}$-object, and $x \in X$ is such that $\iota_X(x) = \text{false}$, then x is 'not inactive', i.e. active, but a morphism $f : X \to Y$ may deactivate it. Conversely, if $\iota_X(x) = \text{true}$, then x is inactive, and since morphisms f must preserve ι, x cannot be 're-activated' by

[5] This is incongruous with LATE-INSERTION theories, which allow an XP to be attached to a high copy in a chain without attaching to the lower copy. Such theories are usually used in explanations of phenomena like Antecedent Contained Deletion [46]. We would instead have to use richer forms of scattered deletion [6] and/or trace conversion to explain such phenomena.

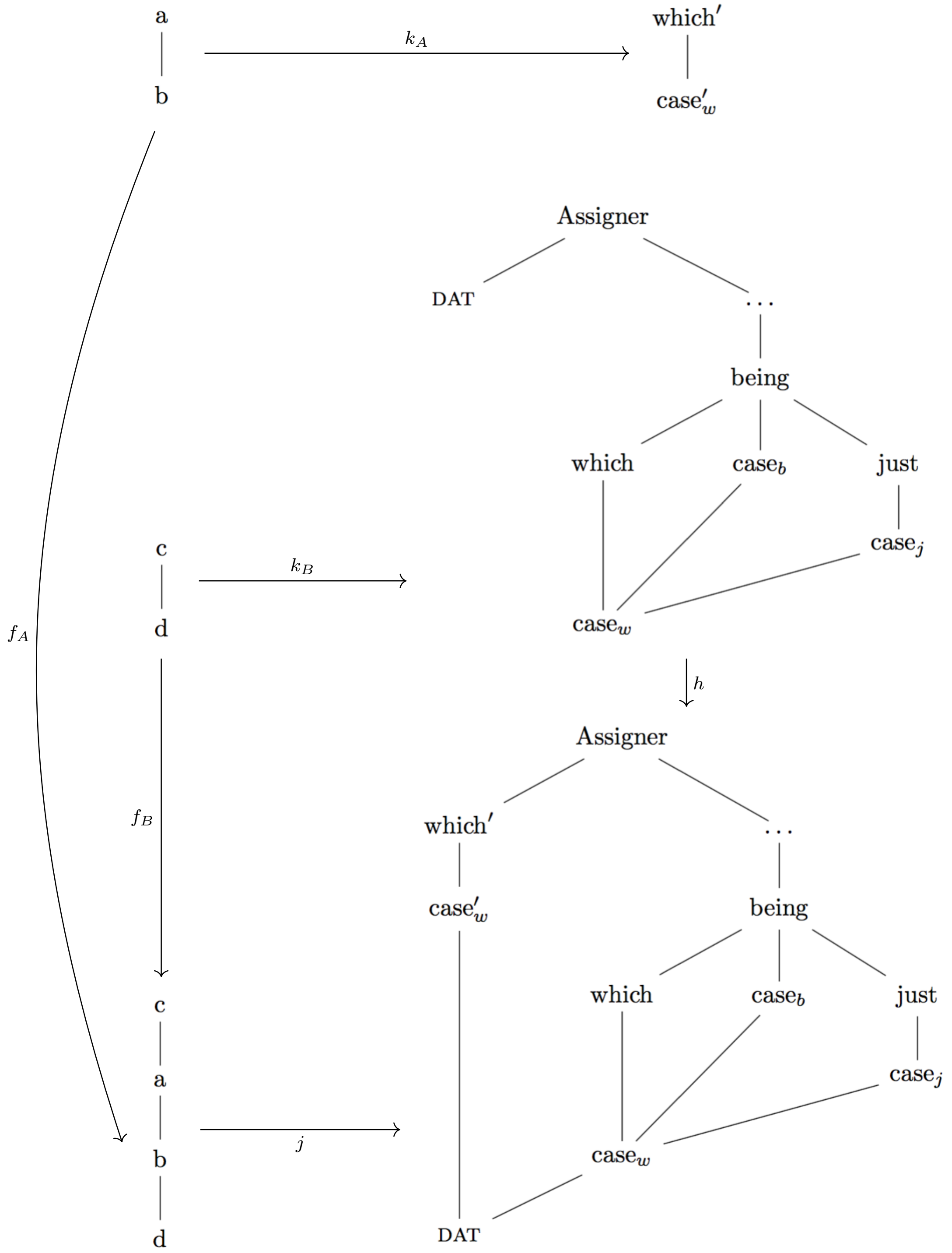


Figure 5.14: A ‘pushout’ of an SC without change-data to a copying construction. Here, the wh-phrase is $K \subset T$, and k maps $a \mapsto$ which′, $b \mapsto$ case′$_w$, $c \mapsto$ Assigner, $d \mapsto$ DAT. The base copy of case$_w$ becomes dependent on DAT, since the chain-data function on Z must be order-preserving.

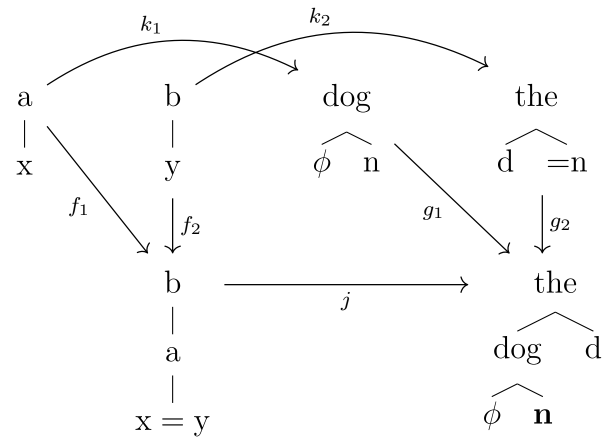


Figure 5.15: If the element x = y is inactive, then j must carry it to an inactive feature, so g_1 and g_2 must deactiveate n and =n.

some SC. We remind the reader how we can use a pushout to deactivate a feature. In Fig. 5.15, if x = y, since j must preserve ι, the image **n** must be inactive, and so g_1 and g_2 deactivate the selection and selectee feature. However, no other elements are deactivated. Now, in Fig. 5.14, suppose that $f_A(b)$ is inactive, which will require that $h(\text{case}'_w)$ is inactive. However, we cannot only deactivate this feature, since $g(h(\text{case}_w))$ must also be inactive, since g must be a morphism. Hence, the (poset,d-hom)-universal pushout for **A**-structures will also deactivate the lower feature in the chain. Thus, (poset,d-hom)-universal pushout for **A**-structures will not only attach a feature to all lower parts of a chain if attaching it to one part, but it will also deactivate all lower copies.

5.4 Summary

Many current formal models of movement do not align with the informal description of copying and chain-formation in Chomsky [6]. In the case of standard

Minimalist Grammars, they do not actually copy material, but rather delay linearization or other incorporation into the structure until an object reaches its final position. This does not structurally represent the 'multiple positions' associated with items in the chain in the DSOs, but rather requires us to reconstruct them from the derivational history. In Kobele's model, copying is done at the beginning of a derivation, as opposed to 'online'. Features are also deliberately not reduplicated in this process, though lexical items are. However, extraneous copies may arise in this model when doing remnant movement due to this 'pre-multiplication' of elements. We implement a model of copying similar to Kobele's, but which forms copies online and adds chain-link information, following the intuition in Chomsky [6]. Since copies are formed 'online', the process does not create extraneous copies. One consequence of formalizing copies in terms of unary algebras is that affecting one element in a chain affects all lower copies, since the unary algebra must preserve DSO structure, whatever structure that may be. This formalizes a version of copying where fragments of a derivation are copied, and features in a chain are all checked simultaneously. We then described maximal condition categories, grammars, and rule generation for these richer grammars with copy data. Most constructions were straightforward to abstract from simpler grammars without copy data by using similar universal constructions and diagrammatic properties relating derivations, DSOs, SCs, and other basic parts of the formalism.

Chapter 6: Comparisons to Other Models, and Conclusions

The defining property of the model of derivations sketched here is that it emphasizes DSOs as ‘structured sets’ and derivations as collections of DSOs and ‘structure-preserving functions’ between these sets. We compare notions of structure in this model to those in formal Minimalist Grammars (MGs) and Bare Phrase Structure (BPS), particularly with respect to notions of isomorphism, equivalence, and substructures. We leave with some closing remarks about the generality of the models presented here, as well as their connection to other theories of generative morphosyntax.

6.1 Comparison to Minimalist Grammars and Bare Grammars

Stabler & Keenan (S&K) [3,4] view a grammar as a set of expressions, together with partial operations. Their method is general, accounting for many variants of formal MGs. We show by example notions of *isomorphism* and *constituent* in these grammars are about the combinatorics of the grammar, while our notions are about the structure of the derivations and DSOs as structured sets. In other words, the combinatorial method formalizes properties related to ‘when operations are defined’, while the categorical approach is about ‘what operations do to structure’.

S&K define a grammar to be a tuple $(G, f_1, \ldots, f_n)$, where G is a set of expressions, and $f_i : G^+ \to G$ are partial functions taking finite tuples $(g_1, \ldots, g_k) \in G^k$ to elements of G. Recall that in the formalism presented here, DSOs are sets of nodes endowed with extra structure relating to dominance, precedence, feature type and activity, etc., and hence isomorphisms between DSOs are mutually inverse bijections which preserve this structure in both directions. However, S&K define isomorphism combinatorially: two objects $x, y \in G$ are isomorphic in their sense if there is a automorphism π of G such that $\pi x = y$ takes one object to the other. An *automorphism* of G is a bijection $\pi : G \to G$ such that $f_i(\pi g_1, \ldots, \pi g_k)$ is defined if and only if $f_i(g_1, \ldots, g_k)$ is defined, and it has value $\pi(f_i(g_1, \ldots, g_k))$. This notion of isomorphism is about what other elements an expression x can combine with for each f_i, not 'intrinsic' structure. For example, embedding a grammar in another can decrease the isomorphisms between objects, simply by virtue of there being more items to combine with. While also a useful notion, it is quite distinct from the notion of isomorphism of DSOs in our categories **A**. We give an example in Fig. 6.1. In Language 1, the lexical items ϵ:: `=x a` and ϵ:: `=y` are isomorphic, since we can interchange them and leave the operations of the grammar intact, since it just so happens that MERGE is not defined anywhere for this language. This language has an obvious inclusion into Language 2. However, in Language 2, the items are no longer isomorphic due to the presence of ϵ:: `y a`. MERGE is now defined for the pair (ϵ:: `=y`, ϵ:: `y a`) resulting in MERGE(ϵ:: `=y`, ϵ:: `y a`) = ϵ:: `a`. If we attempt to interchange the two lexical items, then MERGE(ϵ:: `=x a`, ϵ:: `y a`) would have to be defined, contrary to fact.

Language 1	$\boxed{\epsilon\text{:: =x a}}$	$\boxed{\epsilon\text{:: =y}}$		
Language 2	$\boxed{\epsilon\text{:: =x a}}$	$\boxed{\epsilon\text{:: =y}}$	$\boxed{\epsilon\text{:: y a}}$	$\boxed{\epsilon\text{:: a}}$

Figure 6.1: Language 1 is a sublanguage of Language 2. While the two lexical items in Language 1 are 'isomorphic' using the definitions in K&S, they are not in the larger language, using the same definition.

This difference extends to 'higher order' comparisons like isomorphisms of rules. Recall that for the models presented here, we embellished rules with SC data. That is, not only did a rule return output objects Z for a given tuple $(A_1, \ldots, A_n)$, but it specifically told us how structure was manipulated via giving mappings $f_i : A_i \to Z$. It is then not surprising that equivalences which are definable from 'algebraic' MGs cannot detect this information. Consider the following grammar. Let T be the set of binary unlabeled trees, where each pair of sisters has a precedence ordering $\preceq$ or $\succeq$, and let $G = T_1 + T_2 + T_3$ be the disjoint union of three copies of T. Define $f_1(t, s)$ to be the tree whose root immediately branches into t and s, with the relation $t \preceq s$, and suppose that f_1 is defined only on $t, s \in T_1$ and outputs an element of T_1. Let f_2 be an identical function, but which is defined only on T_2 and outputs an element of T_2. Finally, let f_3 be the same operation on T_3, but taking values $f_3(t, s)$ in T_3 such that the ordering between immediate daughters of the root $t \succeq s$ is reversed. Since automorphisms as defined do not allow permutation of the operations, there is no formal way to say that f_1 and f_2 induce the same structural change, while f_3 performs attachment in the opposite direction.

There is a reasonable weakening of automorphism: let an automorphism (π, κ) of $(G, f_1, \ldots, f_n)$ be a pair where π is a permutation of G and κ a permutation of the f_i such that $f_i(g_1, \ldots, g_k)$ is defined if and only if $\kappa(f_i)(\pi g_1, \ldots, \pi g_k)$ is, such that $\kappa(f_i)(\pi g_1, \ldots, \pi g_k)$ has value $\pi(f_i(g_1, \ldots, g_k))$ if defined. A permutation π interchanging the versions of each s in T_1 and T_2 and κ interchanging f_1 and f_2 is one such automorphism, but so is π interchanging s and s' in T_1 and T_3, where s' is just s with reversed $\preceq$ ordering, while κ interchanges f_1 and f_3. This shows that these combinatorics do not detect what the operations 'do', just where they are defined. In contrast, an isomorphism between structural changes §3.3.3.3 (or, at a higher level, an equivalence of rules §4.2) requires that it 'do isomorphic things' to the structures in question, so this difference can be detected easily. It is straightforward to describe a bijection π between T_1 and T_2 which takes each binary ordered tree to an isomorphic one, giving an explicit isomorphism ϕ between each tree in correspondence. Interpreting f_1 and f_2 as embellished with SCs, we can describe an isomorphism between the operations under this bijection. We view $f_1(t, s)$ as associated to a pair of morphisms $f_{1,t} : t \to f_1(t, s)$ and $f_{1,s} : s \to f_1(t, s)$. Given binary ordered trees $t, s, \pi(t), \pi(s) \in T_1$ with isomorphisms $\phi : t \to \pi(t)$ and $\psi : s \to \pi(s)$, we want to describe an 'isomorphism' between the rules f_1 and f_2. We will have a pair of morphisms $f_{2,\pi(t)} : \pi(t) \to f_2(\pi(t), \pi(s))$ and $f_{2,\pi(s)} : \pi(s) \to f_2(\pi(t), \pi(s))$, the SCs associated to f_2 at the pair $(\pi(t), \pi(s))$. An isomorphism $\chi : f_1(t, s) \to \pi(f_1(t, s)) = f_2(\pi(t), \pi(s))$ commutes with these SCs, in that $\chi \circ f_{1,t} = f_{2,\pi(t)} \circ \phi$ and $\chi \circ f_{1,s} = f_{2,\pi(s)} \circ \psi$. However, notice what happens if we try to do something similar between f_1 and f_3 using a bijection π between T_1 and

T_3, together with isomorphisms between each binary ordered tree in correspondence. There will not be isomorphisms commuting with the SCs in correspondence, since the attachment order is opposite. That is, using isomorphism $\phi : t \to \pi(t)$ and $\psi : s \to \pi(s)$ with the structural changes $f_{1,t}$, $f_{1,s}$, $f_{3,\pi(t)}$, and $f_{3,\pi(s)}$, there will be no isomorphism $\chi : f_1(t, s) \to f_3(\pi(t), \pi(s))$ commuting with the SCs, ϕ, and ψ. The induced bijection χ on underlying sets will not be a morphism, since it will not preserve the reversed sisterhood ordering between the two immediate daughters.

The difference also extends to notions like equivalences of languages. S&K [4] presents an issue in comparing different languages of different sizes. Adding a word to a language will change the number of symmetries of the language, and hence not lead to equivalent languages. However, S&K [4] note 'we would like to say that adding a single name to a language does not (significantly) change the structure of the language.' S&K [4] resolve this by defining a notion of a *free lexical extension*, or more generally a notion of a grammar G being *free for* s, where s is a new lexical item, in S&K [3]. A free lexical extension of a grammar G is a grammar G' which is the same as G, except it has more lexical items, and additionally for each lexical item t in G' which is not in G, there is some lexical item s in G which is isomorphic to t in G'.

> When grammars G, G' are identical except that $\mathrm{Lex}_G \subset \mathrm{Lex}_{G'}$, we say that G' is a lexical extension of G. A lexical extension of G' of G is ***free*** iff for every $s \in \mathrm{Lex}_{G'} - \mathrm{Lex}_G$, there is a $t \in \mathrm{Lex}_G$ such that $s \simeq_{G'} t$.

where $\simeq_{G'}$ means that there is an automorphism of G' taking s to t.

However, free lexical extensions are still not equivalences of languages, unlike in our model. Similarly, consider two free lexical extensions G' and G'' of a grammar G, each adding a different number of words, but for each new word t in G', there is some new word s in G'' such that there is a word r in G which both t and s are isomorphic to in their respective languages, and conversely. Intuitively, G' and G'' should still be equivalent, but they are not, nor is one a free lexical extension of the other. We showed that our definition of equivalence of languages from §3.3.3.3 can detect this similarity straightforwardly using a uniform notion of equivalence of languages.

The number and 'names' of operations can also affect language equivalence. If (G, f) is a grammar, we call a partial function $g : G^+ \to G$ a *subfunction* of f if for every tuple $(a_1, \dots, a_n)$ that g is defined on, f is defined there as well, and $g(a_1, \dots, a_n) = f(a_1, \dots, a_n)$, which we will write $g \subset f$. Intuitively, given a grammar (G, f), adding a subfunction $g \subset f$ to the grammar should not change the language - each instance of a g SC could already be performed by f. However, this is not the case. The languages generated by (G, f) and (G, f, g) do not have to have the same symmetry groups, which is the main formal notion defining the structure of a language in those S&K [3, 4]. For example, let $G = \{a, b, c, x, y\}$ be the grammar with f the partial function taking values $f(a, b) = x$ and $f(a, c) = y$ and undefined everywhere else. The bijection $p : G \to G$ interchanging x with y and b with c is an automorphism, producing a nonidentity automorphism. Now let (G, f, g) be the same grammar, with g the partial function taking value $g(a, b) = x$ and undefined everywhere else. Every derivation in one language, in the sense of sequences of mappings of specified tuples to elements in the language, can be performed in the other, ignoring the 'name' of the operation used. However, (G, f, g) has only the trivial automorphism: p is not an automorphism because $g(pa, pb) = g(a, c)$ is undefined. Hence, the isomorphisms between words in the two languages are not the same, showing that the languages do not have the same 'structure'.

This is clearly not the case for equivalences between languages of structured derivations. Since a grammar with f and grammar with f and g will overall induce isomorphic SCs and derivations, our definition of equivalence of languages from §3.3.3.3 can detect this similarity. Relatedly, recall our example where f_2 and f_3 induce the same structural change, but in opposite directions - i.e. they would behave the same if their inputs were reversed. We can still tell these apart as rules - which keep track of the ordering of the inputs - but overall they will lead to equivalent languages, which ignore that distinction, as derivations only keep track of which tuples of DSOs (ignoring order) map to other DSOs, and the overall SC. This is essentially the content of Claim 3.3.7.

Constituency in K&S-style grammars is defined roughly through being an operand. More precisely, x is a constituent of y if there is a sequence of operations, the first of which takes x in as an argument, where we can successively apply the other operations eventually yielding y. However, this actually only gives the information *that* x eventually maps to y; it does not in any sense identify a 'subpart' of y that x is associated to. Since the DSOs are not structured sets, we cannot define substructures as in §2.3. Moreover, constituency in this model only refers to the *existence* of such a sequence of operations mapping x into y; that is, x may still be a constituent of y even if there is a derivation of y not using x. For example, suppose that we have lexical items λ::=yx, λ::y, λ::x, and λ::=xx in a minimalist grammar. Then λ::=yx and λ::y will be constituents of λ:x, even though there is a derivation $\textsc{merge}(\lambda\text{::=xx}, \lambda\text{::x})$ of λ:x without them. This also shows that constituency in this sense is not an embedding relation of any form - there is no 'subpart' of λ:x corresponding to λ::y, for example.

6.2 Comparison to Bare Phrase Structure

Bare Phrase Structure (BPS) (Collins & Stabler (C&S) [47], [34, 36]) takes an alternative approach to phrase-markers. BPS uses the set-theoretic $\in$-relation to describe constituency. We fix the instantiation of BPS described in Chomsky [21,36] and formalized in C&S. In these models, MERGE is a structure-building operation which takes two objects A and B and forms $\{A, B\}$.[1] From this definition, we can recover an 'immediately contains' relation between the objects A and B and $\{A, B\}$ by using the elementhood relation. Explicitly, we say that X is *immediately contained* in Y if and only if $X \in Y$.[2] General *containment* is defined as the transitive closure of this relation. Explicitly, we can inductively define containment by saying that X is contained in Y if $X \in Y$ or $X \in Z$ for some Z contained in Y.

Strictly speaking, this is a relation which is defined on the entire model of the ambient set-theory, not on a single set X which represents a single syntactic object, as in the case of the precedence and dominance relations between elements of a

[1] C&S, Def. 13

[2] C&S, Def. 8

DSO. That is, containment is a relation between sets in the entire class of sets, not between elements ('nodes') of a single syntactic object. Accordingly, a substructure with respect to the $\in$-relation refers not to a subset of any object in the model, but rather to a *submodel* of the model of set theory [42].

It is straightforward to show that constituents are not in general subsets of a BPS syntactic object X.

(23) Let A, B, C, and D be lexical items or complex syntactic objects.

Construct $X = \textsc{merge}(A, \textsc{merge}(B, \textsc{merge}(C, D))) = \{A, \{B, \{C, D\}\}\}$. Then, $\{C, D\}$ is contained in X, but $\{C, D\} \not\subset X$.

As syntactic objects X are also not models of set-theory, but rather the elements of such a model, the submodel relationship which preserves the $\in$ relation also cannot be the correct notion of substructure for syntactic objects.

We now present arguments that the $\in$ relation, and its transitive closure, while providing an accurate characterization of the *containment* relation,[3] do not provide a *substructure* relation between syntactic objects. Unfortunately, constituency cannot be used to determine the appropriate notion of substructure, since, in trees, 'A contains B' is coextensive with 'the constituent dominated by B is a substructure of the constituent dominated by A'. In other words, we cannot tell the containment relation apart from substructure inclusions between constituents. However, in slightly relaxed notions of substructures, $\in$ is clearly behaving as a primitive

[3] Ignoring issues relating to 'occurrences' of lexical items - i.e. non-tree structures resulting from the elementhood graphs of sets.

containment relation between nodes, and not a substructure inclusion. We turn to some motivating examples.

In C&S, lexical items are treated as a triple of sets of features (SEM, SYN, and PHON). The features of a syntactic object X are formalized externally with a TRIGGERS function. C&S keep track of which features have been satisfied by removing elements from the sets of features associated to X via TRIGGER. Chomsky suggests in "Categories & Transformations" (CT) that certain formal features may be *erased* upon satisfaction, or at the interfaces.[4] We first look at how C&S formalize their calculus of features.

(24) (C&S Def. 26) TRIGGERS is any function from each syntactic object A to a subset of the trigger features of A, meeting the following conditions:

(i) If A is a lexical item with n trigger features, then $\textsc{triggers}(A)$ returns all of those n trigger features. (So when $n = 0$, $\textsc{triggers}(A) = \{\}$.)

(ii) If A is a set, then $A = \{B, C\}$ where $\textsc{triggers}(B)$ is nonempty, and $\textsc{triggers}(C) = \{\}$, and $\textsc{triggers}(A) = \textsc{triggers}(B) - \{\text{TF}\}$, for some trigger feature $\text{TF} \in \textsc{triggers}(B)$.

(iii) Otherwise, $\textsc{triggers}(A)$ is undefined.

This goes hand in hand with their definition of triggered merge.

(25) (C&S Def. 27) Given any syntactic objects A, B, where $\textsc{triggers}(A) \neq \{\}$ and $\textsc{triggers}(B) = \{\}$, $\textsc{merge}(A, B) = \{A, B\}$.

[4] Chomsky [9], p. 280: "Erasure is a 'stronger form' of deletion, eliminating the element entirely so that it is inaccessible to any operation, not just to interpretability at LF."

The idea is that two items may only merge when one has remaining trigger features, and the other does not. If defined, the trigger features of $\{A, B\}$ are just those of the triggering object A with the triggering feature removed. Notice, however, that TRIGGER keeps track of the feature changes externally, in that no features of heads contained in A or B are changed. Under such a method, the set-theoretic structure of syntactic objects alone does not encode the featural changes. We want to 'internalize' the feature calculus so that MERGE actually results in changes in the structure of the objects it combines.

We have at least two reasonable options for formally realizing these notions of erasure/deletion within a syntactic object itself: by removing the element in question from the syntactic object, or by changing the element in some way which marks it as inoperative. We will show that either method results in an object which the $\in$ relation and its transitive closure both fail to treat as related to the original object in any straightforward way. We will extend the argument to cases of AGREE.

6.2.1 Method one: removal of the feature

For any sets A and B, we can construct a set $A - B = \{a \in A : a \notin B\}$, their *difference*, which removes B-elements from A.

Let A be a lexical item and X and Y be syntactic objects (lexical items or otherwise). We treat lexical items as in CT, where A is literally a *set* of features. Take the syntactic object MERGE$(A, Y) = \{A, Y\}$.[5] Suppose that when this object

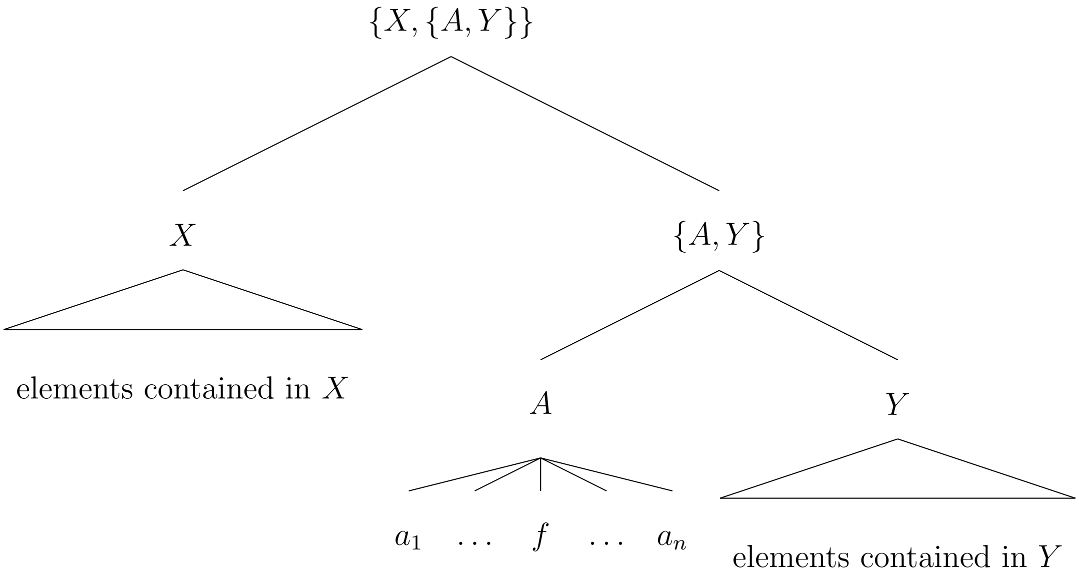
[5] For simplicity, we delete no features in the first step, though the argument still holds if we do remove a feature of A (or Y) during this first step.

is merged with X, a feature of the head A is checked, removing $f \in A$, resulting in the object $\{X, \{A - \{f\}, Y\}\}$. Alternatively, if features are not deleted in syntax, we may say that some interface only sees the structure $\{X, \{A - \{f\}, Y\}\}$, which should be a substructure of $\{X, \{A, Y\}\}$. In the first case, we should like to describe in what sense $\{A - \{f\}, Y\}$ is a substructure of $\{A, Y\}$ in that they have the same phrase structure, with the former simply missing a feature of the latter. In the second case, we should like to describe how $\{X, \{A - \{f\}, Y\}\}$ is a substructure of $\{X, \{A, Y\}\}$.

As expected, a subset relation fails to hold in both cases: $\{X, \{A-\{f\}, Y\}\} \not\subset \{X, \{A, Y\}\}$, and $\{A - \{f\}, Y\} \not\subset \{A, Y\}$. However, there is also no containment relation between the syntactic objects. In fact, there is no straightforward set-theoretic relation between these objects. While a subset relation $A - \{f\} \subset A$ does hold, $\{A - \{f\}, Y\} \not\subset \{A, Y\}$. More generally, for any constituent M containing a head A from which we remove a feature, the resulting constituent M' will simply be a distinct set from M (often with the same number of elements as M). In this example, $\{X, \{A - \{f\}, Y\}\}$ and $\{X, \{A, Y\}\}$ have the same number of elements, though A and $A - \{f\}$ do not, assuming A is finite.

On the other hand, there are canonical ways to draw graph-theoretic objects from well-founded sets. One method produces trees: draw a set X as a root, and write all of its elements as immediate daughters. We repeat the process at each child, writing the same element multiple times if necessary. This process is described in Aczel [48].

(24)


$\{X, \{A, Y\}\}$
X
$\{A, Y\}$
elements contained in X
A
Y
a_1 . . . f . . . a_n
elements contained in Y

For syntactic objects K, we can define a set X of occurrences of contained elements, with $R \subset X \times X$ being the immediate containment relation between the appropriate occurrences; see C&S, §4, Def. 18 for a formal treatment. We can define a *subgraph* relation between two graphs (X, R) and (Y, S) if $X \subset Y$ and we have a relation xRx' for $x, x' \in X$ if and only if xSx' in Y. We can then form the graph-theoretic tree associated to $\{X, \{A - \{f\}, Y\}\}$, which is clearly a subgraph of the graph in (24). We could similarly use the containment relation in place of the immediate containment relation, which would describe the syntactic objects as partially ordered sets, with the substructure relation being a subspace inclusion of finite partial orders.

6.2.2 Method two: changing (the value of) a feature

Changing the 'value' or otherwise adding diacritical marks to an element is another way to formally represent the status of a feature in a syntactic object. In this case, suppose that we have again constructed $\{A, Y\}$ which we intend to merge with X in a way which will alter a feature $f \in A$. This alteration could be realized as a bijection $m : A \to A'$, where A' is the same set as A, except the feature f has been replaced by $\bar{f}$, the 'inoperative' form of f.

However, $\{A, Y\}$ is not a subset of $\{X, \{A', Y\}\}$, nor do we have a containment relation between the two sets. Much like subsets are not the relevant notion of substructure for BPS sets, neither will bijection be the appropriate notion of isomorphism of underlying hierarchical structure, nor will functions be the appropriate notion of homomorphism. For, depending on whether we allow MERGE to combine identical sets or not, every BPS set will have cardinality 1 or 2, and hence be in a bijection with the set $1 = \{0\}$ or $2 = \{0, 1\}$. So while $\{A', Y\}$ and $\{A, Y\}$ are 'isomorphic' in that there is a bijection between them, so are they both isomorphic to $\{X, \{A, Y\}\}$, showing that this is not the correct notion of 'isomorphism' between the objects, in that it totally ignores constituency.

Again, we may convert $\{A, Y\}$ and $\{X, \{A', Y\}\}$ into graph- or order-theoretic trees. We can define an isomorphism between graphs (X, R) and (Y, S) as a bijection $m : X \to Y$ such that xRx' in X if and only if $(mx)S(mx')$ in Y (or similarly, an isomorphism of partial orders as a bijection $m : P \to Q$ such that $x \leq x'$ in P if and only if $m(x) \preceq m(x')$ in Q). Using these definitions, two graph- or order-theoretic trees (X, R) and (Y, S) will be isomorphic if and only if they have the same number of nodes with the same constituency relations.[6] Using this definition, the graphs

[6] Though, this ignores the 'occurrence' relations which indicate which nodes are 'copies' of others. On the other hand, the multidominant picture of a tree, called the *canonical picture* in

associated to $\{A, Y\}$ and $\{A', Y\}$ will be isomorphic, such that $\{A, Y\}$ is isomorphic to a subgraph of $\{X, \{A', Y\}\}$ in the appropriate way.

Alternatively, we might think of this 'value' or 'activity' as a property of a feature which is explicitly part of its structure. This again has a straightforward formalization when the syntactic objects are graphs: we define a graph-with-value as a graph (X, R) together with a function $\nu : X \to \{\text{true}, \text{false}\}$ where we interpret $\nu(x) = \text{true}$ as meaning 'x is inactive'. We define a homomorphism between graphs-with-values $f : (X, R, \nu) \to (X', R', \nu')$ as a graph homomorphism such that if $\nu(x) = \text{true}$, then $\nu'(f(x)) = \text{true}$, i.e. inactive features stay inactive, but active features may be deactivated. Using this structure, the inclusion of an operand A into larger object X, while deactivating a feature in A, would be a homomorphism.

6.2.3 Agree

The above examples showed that the feature-deletion and feature-valuation methods of modeling MERGE do not lead to substructure embeddings or homomorphisms between BPS sets in any obvious sense. In contrast, relations between derived syntactic objects are straightforward when represented as graphs (possibly with extra structure). Chomsky [49] has a 'valuation' version of agreement, which is subject to similar analysis as the valuation case for selection above. We look now at a feature-sharing approach to agreement, and similarly show that the structural relation between the input structures and output structures is given straightforwardly by graph homomorphisms, while there is no clear associated notion for sets.

Azcel [48], and given in Fig. 3 in C&S, would not have this issue, and could be used instead.

We recall the architecture for agreement as feature-sharing from Frampton & Gutmann [12] discussed in §4.3, which is couched in the set-theoretic model of BPS. The arrow 'Agree' in Frampton & Gutmann's figure in §4.3 can clearly be viewed as a pair of graph homomorphisms from each graph on the lefthand side to the graph on the righthand side, or as a single graph homomorphism from the 'structured disjoint union' of the graphs on the lefthand side to the graph on the righthand side. If we view the valuations as properties attached to the nodes of the graph, then we can additionally view this map Agree as a graph homomorphism which preserves those properties (e.g. a pl node gets taken to a pl node). However, it is again difficult to describe the relationship above when we view the objects as BPS sets. Usually at least one of A or B above will be in a phrase when agreement is applied. Suppose it is B, and we have $B \in \ldots \in X$. We intend to construct from A and X an object $\{A', X'\}$, where A' and X' are exactly A and X, but where the number and case features have been replaced accordingly. Again, we will have no subset, containment, or other obvious set-theoretic relation between A or X and $\{A', X'\}$.

We mentioned in §4.3 that isomorphisms appear implicitly here. Frampton & Gutmann note that the specific index for the element representing the shared feature does not matter, so long as it is suitably distinguished. Again, while the set-theoretic statement of this is somewhat complex (and relies on knowing the specific indices used elsewhere in syntactic objects contained in the current one), the graph-theoretic notion is quite elegant: the righthand side above is determined up to isomorphism of graphs (possibly with values assigned to nodes).

6.2.4 Graph structure

We finally note that it is really the graph structure which we may *associate* to a BPS set which is the object of interest for reasons other than just statement of formal mathematical relations like substructures, isomorphisms, and general homomorphisms. §4 of C&S describes many of the important syntactic relations through *paths* associated to a BPS set. Specifically, a path is a sequence of constituents contained in a BPS set $\langle X_1, \ldots, X_n \rangle$ such that $X_i \in X_{i+1}$. For a BPS set X, the paths from constituents to X determine a graph structure. The *positions* that a constituent occurs in in X are stated with respect to these paths, and hence this is also how relations like c-command are stated, as well as chains, similar to the method described by Chomsky [9], Ch. 4. Importantly, these properties like c-command and minimality/locality which are relevant for describing interpretations of and application of operations to syntactic objects then care about this path (i.e. graph) structure. The BPS formalism is then largely used insofar as it encodes graph structures. This is reenforced by the preceding discussions, where we showed that graph morphisms naturally described structural similarity between different objects, not set functions or morphisms between models of set theory.

6.3 Conclusions

The previous sections show that while on an intuitive level, the models presented here have many connections with other mainstream formalizations like formal Minimalist Grammars or Bare Phrase Structure, at a technical level they are quite different.

First turning to DSOs, objects in our models are 'structured sets' - formally, representably concrete categories. Isomorphisms between objects are then induced, and are special bijections which preserve the relations between and properties of the nodes occurring in a structure. However, in formal Minimalist Grammars, DSOs are *not* structured sets (or at least this is not part of the formalization of them), and hence 'isomorphisms' between objects are determined combinatorially. In Bare Phrase Structure, there is a tempting notion of isomorphism between structures given by putting SOs contained in two SOs X and Y in correspondence, though exactly how one should do this is dependent on how we correspond the objects with graph structures. For example, how we count occurrences of a SO which is contained in X and integrate them into the hierarchical structure associated to X will affect which structures are isomorphic. Different canonical ways to do this are given in Azcel [48], showing that the correspondence between set and graph structures is not unique. In any event, the relevant notion seems to boil down to isomorphisms of the more traditional sort used in the theory presented here. There are other related issues using a purely set-theoretic characterization of SOs, such as how to represent ordered pairs versus unordered pairs as in Chomsky [21]. Since the $\in$ relation is used to describe the structure of a syntactic object, using it to also encode ordered pair structure can lead to ambiguity as to what object a set is to represent. We avoid such issues by describing the structure we care about directly in the categorical data.

As concrete categories, DSOs in our model have a natural notion of substructure which can be defined in general for many different models. In contrast, most formal Minimalist Grammars have no such notion of substructure on the DSOs. For example, while it is often 'obvious' that two DSOs combining to form a third correspond to 'parts' of the output, these parts are not part of the formal theory. Similarly, §6.2 showed that the $\in$ relation is not sufficient to describe general substructures in BPS, such as those obtained by deleting a feature, and similarly feature checking/replacement does not arise as a homomorphism. More generally, being able to identify substructures and containment relations between them is often part of how we intuitively view the targeting of substructures for copying, description of minimality or binding theory configurations, probe-goal relations, etc. It was the existence of these substructures which also allowed for our formalization of 'online copying' in §5.2.

The 'structure' we endow sets with to describe DSOs goes hand-in-hand with the notion of 'morphism' preserving that structure. This is how we formalized structural changes, which makes all other constructions presented in the thesis possible. Namely, being able to talk about what structure a rule 'targets', as in §4.2, and how it can be generated from a 'basic' SC, as in §4.3, required both that DSOs are structured and that SCs relate the structures of inputs and outputs. Our construction of maximal condition categories allows us to perform standard constructions like finding the repletion of the category, thereby extending a structural rule to be able to apply to all tuples of objects isomorphic to ones it was originally defined on. Description of when a rule is associated with SCs meeting versions of Inclusiveness or Extension relied on properties of the underlying functions or morphisms. In formal Minimalist Grammars, only the assignments of outputs to inputs is formalized, making such constructions and analyses impossible. The 'inverse' of introduction of structure is the measurement of what structure was introduced, leading to the theory of grammatical relations in §3.2.1. In Minimalist Grammars, these grammatical relations must be stated by proxy, associating a particular operand of a particular operation with a particular kind of grammatical relation - associations which do not follow from general principles about the structural relations introduced. Stemming from our ability to formalize basic SCs which are applied 'in some context' is the analysis of deconstruction of rules into component SCs in §4.4, which is again unstateable without data of this sort. Similarly, it is from these deconstructions that we are able to talk about what aspects of a SC are similar between different rules, such as sharing a phrasal attachment or agreement component, as well as demonstrate how movement involves similar SCs as MERGE-operations after using the FORM-CHAIN functor. In Minimalist Grammars, at a formal level, all the grammatical operations are simply distinct, as are all instances of their application.

That the structure of DSOs leads immediately to a theory of SCs aligns with the category-theoretic motto that it is the 'morphisms which matter'. This extended to other parts of the theory, such as the connection between constituent-preserving maps and description of c-command as negation in §2.5.2. Similarly, we used unary algebras to keep track of chain data on derivations. Universal constructions which must preserve chain data (i.e. be unary algebra homomorphisms) automatically induced the desirable property that applying a SC to one element in a chain spreads to its copies. Being able to consider SCs as morphisms and knowing the isomorphisms between DSOs allowed us to construct a more robust theory of language equivalence in §3.3.3.3 which, while still caring about combinatorics of the grammar, ignores structurally irrelevant properties like cardinality, the number or indexation of rules used to induce the SCs, or particular notation/indexation used to describe copies, occurrences, feature identification, extensions, etc. §6.1 showed that isomorphisms of languages in S&K's sense is combinatorial and very rigid. We gave examples showing that there are readily constructed examples of languages which we would like to be equivalent, despite neither being isomorphic, having the same symmetries, nor being in a free lexical extension relation.

One tenet of traditional minimalist syntax we have rejected is that operations do not union structure. Chomsky argues as follows in *Categories & Transformations*:

> The label γ must be constructed from the two constituents α and β. Suppose these are lexical items, each a set of features. Then the simplest assumption would be that γ is either
>
> (6) a. the intersection of α and β
>
> b. the union of α and β
>
> c. one or the other of α, β
>
> The options (6a) and (6b) are immediately excluded: the intersection of α, β will generally be irrelevant to output conditions, often null; and the union will be not only irrelevant but "contradictory" if α, β differ in value for some feature, the normal case. We are left with (6c): the label γ is either α or β; one or the other *projects* and is the *head* of K. If α

> projects, then K = $\{\alpha, \{\alpha, \beta\}\}$
>
> Chomsky [9], p. 244

We, however, reject that (6b) is automatically excluded: the pushout constructions of §4.3 in many ways act like a 'structured union' of DSOs, capturing the intuition that a DSO formed out of A and B does in fact consist of parts of both A and B. In fact, this is essential: it is exactly the 'inclusions' into the 'unioned structure' which give information about how the two inputs formed and are related to the output object.

6.3.1 Generality

The technique was also developed in great generality, making only enough assumptions about the DSOs to guarantee the existence of certain constructions. We have given examples throughout which illustrate how feature-sharing is easily admitted into the model. Similarly, nothing we have done excludes using feature geometric structures [50], and in fact many of our proposed SCs such as for head-adjunction will automatically keep track of that geometry when 'unioning'. Similarly, models like the one presented in Bye & Svenonius [15] which combines aspects of Mirror Theory [14] and nanosyntax [51] are easily modeled. Both models which allow and models which restrict feature movement [9] can easily be modeled as well. In all cases, the theories of structural analyses, structural changes, copies, etc. can still be implemented straightforwardly. We take this to be a benefit of the methods presented here - they show that the constructions we use to model copying, constituency, projection, etc. are very simple abstract properties, and do not care about idiosyncratic aspects of the notation or formalization used.

Appendix A: Proofs

Lemma 3.5.8.1. Let X be any finite partial order and $i : S \to X$ any injective order-preserving function considered as a subset, and take the pushout *of preorders* of i with itself to obtain $X +_S X$. Then, $X +_S X$ is already antisymmetric, and hence the forgetful functor from **FPos** to **Set** preserves this pushout, and $s(x) = t(x)$ if and only if $x \in S$.

Proof. We identify S with a subset of X. The pushout of sets $X +_S X$ is effectively the set X 'doubled' where we have identified points from S in each copy. As a set pushout, the functions $s, t : X \rightrightarrows X +_S X$ have the property that $s(x) = t(x)$ iff $x \in S$. To complete the theorem, we must simply show that the induced preordering on $X +_S X$ is already antisymmetric.

If we look in each 'copy' of X in $X +_S X$, the ordering looks just as it does on X. We transitively add relationships $a \leq b$ between elements in different copies of $X - S$ if there is an element $y \in S$ such that $a \leq y \leq b$. If $a \leq b$ and $b \leq a$ in a single copy of X, then $a = b$ by virtue of antisymmetry there. If a and b are in different copies of $X - S$ such that $a \leq b$ and $b \leq a$ in $X +_S X$, then we must have elements $y, y' \in S$ such that $a \leq y \leq b$ and $b \leq y' \leq a$. Viewing all of these as elements of X, this means that b and a are equally ordered in X, so the only elements y between them are representatives of $a = b$. But by assumption $a, b \notin S$, a contradiction. So distinct copies of a point in $X - S$ cannot be ordered as such, and the ordering is antisymmetric. □

Claim 3.5.10. If $\phi : \Delta \to \Gamma$ is a constituent-preserving morphism between separated derivations, then $T(\phi) : T(\Delta) \to T(\Gamma)$ is an open map.

Proof. The theorem can be proven using an alternate characterization of open maps, and then using a back-and-forth method to construct blockwise order sequences in the domain and codomain.

The following are equivalent:

(1) $f : X \to Y$ is an open map of finite preorders

(2) For every $x \in X$, if $y \geq f(x) \in Y$, then there exists some $z \in X$ such that $f(z) = y$ and $z \geq x$.

If f is an open map, and $U \subset X$ an open subset, then $f(U) \subset Y$ is open. By finiteness, U is generated by some finite set $\{x_1, \ldots, x_n\} \subset X$ such that for all $x \in U$, $x \geq x_i$ for some x_i, a generator. Then for any $f(x_i) \leq y$, by openness, some element of U must map to y. Since this holds for all open sets, and for any $x \in X$ the set $\{x' \in X : x' \geq x\}$ is open, (1) $\Rightarrow$ (2). If (2) holds, then for any U we can use (2) on the generators to produce the necessary elements of U to hit all points greater than some element in the image $f(U)$, so that this map is surjective, and then f is open, so that (2) $\Rightarrow$ (1).

We now introduce terminology related to the partition into close-components. On a finite partial order X, an equivalence relation $\sim$ is said to be *regular* or a *regular partition* of X if the induced quotient on preorders $q : X \to \tilde{X}$ is actually a map of partial orders. [52] gives the following results for regular morphisms in the category **FPos** (Codara uses 'poset' to mean 'finite poset').

Definition 2.7 (Blockwise order). Let $(P, \leq)$ be a poset and let $\pi = \{B_1, B_2, \ldots, B_k\}$ be a partition of the set P. For $x, y \in P$, x ***is blockwise under*** y ***with respect*** π, written

$$x \lesssim_\pi y,$$

if and only if there exists a sequence

$$x = x_0, y_0, x_1, y_1, \ldots, x_n, y_n = y \in P$$

satisfying the following conditions:

(1) for all $i \in \{0, \ldots, n\}$, there exists j such that $x_i, y_i \in B_j$,

(2) for all $i \in \{0, \ldots, n-1\}$, $y_i \leq x_{i+1}$.

Codara [52]

In other words, the blockwise ordering on a poset P with partition π is just the (pre)ordering between elements of P, such that we are allowed at any point to 'jump' to any other element in the same block while zig-zagging from x to y.

Definition 2.8 (Fibre-coherent map). Consider two partially ordered sets $(P, \leq_P)$ and $(Q, \leq)$. Let $f : P \to Q$ be a function, and let $\pi_f =$

$\{f^{-1}(q) \mid q \in f(P)\}$ be the set of fibres of f. We say f is a ***fibre-coherent map*** whenever for any $p_1, p_2 \in P$, $f(p_1) \leq f(p_2)$ if and only if $p_1 \lesssim_{\pi_f} p_2$.

Codara [52]

Codara gives the following result using these definitions.

Proposition 2.4 In **FPos**, regular epimorphisms are precisely fibre-coherent surjections.

Codara [52]

Codara characterizes regular partitions as follows.

Definition 3.4 A ***regular partition*** of a poset P is a poset $(\pi_f, \preceq)$ where π_f is the set of fibres of a fibre-coherent surjection $f : P \to Q$, for some poset Q, and $\preceq$ is the partial order on π_f defined by

$$f^{-1}(q_1) \preceq f^{-1}(q_2) \text{ if and only if } q_1 \leq q_2,$$

for each $q_1, q_2 \in Q$.

Codara [52]

Finally, giving the following result:

Theorem 3.2. If P is a poset, $(\pi = \{B_1, B_2, \ldots, B_k\}, \preceq)$ is a regular partition of P if and only if π is a partition of the underlying set P, and $\preceq$ is a partial order on π such that for each pair B_i, B_j of blocks of π, and for all $x \in B_i$, $y \in B_j$,

$$x \lesssim_\pi y \text{ if and only if } B_i \preceq B_j,$$

where $\lesssim_\pi$ is the blockwise quasiorder induced by π.

Codara [52]

And by Def 3.4, this blockwise order is actually the partial order $P/\sim$. Summarizing, $f : P \to Q$ is a regular epimorphism of finite posets if and only if for the partition π on P of fibres of f, the blockwise ordering of elements in P coincides with the ordering of the image of the blocks they are contained in under f.

Now suppose that $a \in \Delta$ has equivalence class $[a]$ in $T(\Delta)$ and $\phi : \Delta \to \Gamma$ is a constituent-preserving map. By coherence, it induces a map f between the quotients of points $f : T(\Delta) \to T(\Gamma)$, taking $f([a]) = [b]$. To show that f is open, for any $[c]$ such that $[b] \leq [c]$ in $T(\Gamma)$, we must find some $[z]$ such that $[z] \geq [a]$ in $T(\Delta)$ and $[z]$ maps to $[c]$.

We denote the equivalence relation 'is in the same close-component' by $\sim$. If $f([a]) = [b]$, that means there is some point x such that $x \sim a$, such that $\phi(x) = y$ and $y \sim b$. Now, if we have $[c] \geq [b]$, by the regularity of the partition, we must have that c is blockwise greater than b in $T(\Gamma)$. Choose some sequence satisfying the blockwise ordering requirements, so that $c = x_0 \sim y_0 \geq x_1 \sim y_1 \geq \ldots \geq x_n \sim y_n = b$. We construct a blockwise path in Δ mapping to this blockwise path by a back-and-forth method. Start with $b = y_n$.

Since we have established that there is some point x mapping to $y \sim b$ under ϕ, there is a component K containing x, a component C containing b and a surjective map $\phi : K \to C$. By surjectivity, there is some $\widehat{y_n} \sim a$ mapping to y_n and $\widehat{x_n} \sim a$ mapping to x_n. In the blockwise sequence, we assume that there is some step $x_n \leq y_{n-1}$. But since $\widehat{x_n}$ hits the point x_n, and ϕ is open, there must be some point $\widehat{x_n} \leq \widehat{y_{n-1}}$ of Δ which hits y_{n-1}.

Now, for the inductive step, suppose there is some point $\widehat{y_i}$ hitting y_i from the sequence. There must be some $\widehat{x_i} \sim \widehat{y_i}$ hitting $x_i \sim y_i$ since there are components K' and C' containing $\widehat{y_i}$ and y_i such that $\phi : K' \to C'$ is surjective. From the blockwise ordering sequence, we have $x_i \leq y_{i-1}$. But ϕ is open, and $\widehat{x_i}$ maps to x_i, so there is some element $\widehat{x_i} \leq \widehat{y_{i-1}}$ mapping to y_{i-1}.

Continuing, we obtain a blockwise ordering $\widehat{y} \lesssim \widehat{y_{n-1}} \lesssim \widehat{y_{n-2}} \ldots \lesssim \widehat{y_0}$ in Δ, where $\widehat{y_i}$ hits y_i in the sequence. In particular, this final $\widehat{y_0}$ hits $y_0 \sim c$. Since equivalence classes in $T(\Delta)$ are ordered iff elements in their class are blockwise ordered, we have that $[a] \leq [\widehat{y_0}]$. Since $\widehat{y_0}$ hits $y_0 \sim c$, $[\widehat{y_0}]$ must be mapped to $[c]$, so we can always produce the necessarily element, and the map is open.

□

Bibliography

[1] Noam Chomsky. *Syntactic Structures.* Mouton, 1957.

[2] Edward P Stabler. Computational perspectives on minimalism. *Oxford handbook of linguistic minimalism*, pages 617–643, 2011.

[3] Edward L Keenan and Edward P Stabler. Language variation and linguistic invariants. *Lingua*, 120(12):2680–2685, 2010.

[4] Edward P Stabler and Edward L Keenan. Structural similarity within and among languages. *Theoretical Computer Science*, 293(2):345–363, 2003.

[5] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation: part ii: single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars*, pages 247–312, 1997.

[6] Noam Chomsky. A minimalist program for linguistic theory. In Kenneth Hale and Samuel J. Keyser, editors, *The view from Building 20: Essays in linguistics in honor of Sylvain Bromberger*, pages 1—52. Cambridge, Mass.: MIT Press, 1993. [Reprinted in Noam Chomsky, *The minimalist program*, 167-217. Cambridge, Mass.: MIT Press, 1995].

[7] Luigi Rizzi. *Relativized minimality.* The MIT Press, 1990.

[8] Tim Hunter. Deconstructing merge and move to make room for adjunction. *Syntax*, 18(3):266–319, 2015.

[9] Noam Chomsky. *The minimalist program.* Cambridge, Mass.:MIT Press, 1995.

[10] Pieter Muysken. Parametrizing the notion head. *Journal of Linguistic Research*, 2:57–76, 1982.

[11] Gregory Michael Kobele. *Generating Copies: An investigation into structural identity in language and grammar.* PhD thesis, Citeseer, 2006.

[12] John Frampton and Sam Gutmann. Agreement is feature sharing. *Ms., Northeastern University*, 2000.

[13] David Pesetsky and Esther Torrego. The syntax of valuation and the interpretability of features. *Phrasal and clausal architecture: Syntactic derivation and interpretation*, pages 262–294, 2007.

[14] Michael Brody. Mirror theory: Syntactic representation in perfect syntax. *Linguistic Inquiry*, 31(1):29–56, 2000.

[15] Peter Svenonius and Patrik Bye. Non-concatenative morphology as epiphenomenon. 2011.

[16] Marisa Ferrara Boston, John T Hale, and Marco Kuhlmann. Dependency structures derived from minimalist grammars. In *The Mathematics of Language*, pages 1–12. Springer, 2010.

[17] Noam Chomsky. The logical structure of linguistic theory. 1975.

[18] Omer Preminger. How can feature-sharing be asymmetric? valuation as over geometric feature structures. 2017.

[19] Chris Barker and Geoffrey K Pullum. A theory of command relations. *Linguistics and Philosophy*, 13(1):1–34, 1990.

[20] Richard S Kayne. *The antisymmetry of syntax*. Number 25. mit Press, 1994.

[21] Noam Chomsky. On phases. *Current Studies in Linguistics Series*, 45:133, 2008.

[22] John Torr and Edward P Stabler. Coordination in minimalist grammars: Excorporation and across the board (head) movement. In *The 12th International Workshop on Tree Adjoining Grammars and Related Formalisms*, page 1, 2016.

[23] Francis Borceux. *Basic category theory*. Cambridge Univ. Press, 1994.

[24] Bodo Pareigis. Category theory. 2018.

[25] Jiří Adámek, Horst Herrlich, and George E Strecker. Abstract and concrete categories. the joy of cats. 2004.

[26] Saunders Mac Lane. *Categories for the working mathematician*. Springer-Verlag, 1971.

[27] Edward Stabler. Derivational minimalism. In *Logical aspects of computational linguistics*, pages 68–95. Springer, 1996.

[28] JP May. Finite topological spaces. *Notes for REU*, 2003.

[29] Peter T Johnstone. Complemented sublocales and open maps. *Annals of Pure and Applied Logic*, 137(1):240–255, 2006.

[30] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in geometry and logic: a first introduction to topos theory.* Springer-Verlag, 1992.

[31] Jonathan David Bobaljik. Distributed morphology. *Ms., University of Connecticut*, 2015.

[32] Norbert Hornstein, Jairo Nunes, and Kleanthes K. Grohmann. *Understanding minimalism.* Cambridge University Press, 2005.

[33] Steve Awodey. *Category theory.* OUP Oxford, 2010.

[34] Naoki Fukui. Merge and bare phrase structure. *The Oxford handbook of linguistic minimalism*, pages 73–95, 2011.

[35] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamentals of algebraic graph transformation. volume xiv of monographs in theoretical computer science. an eatcs series, 2006.

[36] Noam Chomsky. *Minimalist inquiries: The framework.* Number 15. MIT Working Papers in Linguistics, MIT, Department of Linguistics, 1998.

[37] Maria Polinsky and Eric Potsdam. Long-distance agreement and topic in tsez. *Natural Language & Linguistic Theory*, 19(3):583–646, 2001.

[38] Rajesh Bhatt. Long distance agreement in hindi-urdu. *Natural Language & Linguistic Theory*, 23(4):757, 2005.

[39] Annie Zaenen, Joan Maling, and Höskuldur Thráinsson. Case and grammatical functions: The icelandic passive. *Natural Language & Linguistic Theory*, 3(4):441–483, 1985.

[40] Carson T Schütze. Towards a minimalist account of quirky case and licensing in icelandic. *MIT working papers in linguistics*, 19:321–375, 1993.

[41] Danny Fox and David Pesetsky. Cyclic linearization of syntactic structure. *Theoretical linguistics*, 31(1-2):1–45, 2005.

[42] Chen Chung Chang and H Jerome Keisler. *Model theory.* Elsevier, 1990.

[43] Jean Bénabou and Bruno Loiseau. Orbits and monoids in a topos. *Journal of Pure and applied algebra*, 92(1):29–54, 1994.

[44] Garrett Birkhoff. *Lattice theory.* American Mathematical Society, 1967.

[45] Avery D Andrews. Case agreement of predicate modifiers in ancient greek. *Linguistic Inquiry*, 2(2):127–151, 1971.

[46] Danny Fox. Antecedent-contained deletion and the copy theory of movement. *Linguistic Inquiry*, 33(1):63–96, 2002.

[47] Chris Collins and Edward Stabler. A formalization of minimalist syntax. *Syntax*, 2016.

[48] Peter Aczel. Non-well-founded sets. 1988.

[49] Noam Chomsky. *Derivation by phase.* Number 18. MIT, Department of Linguistics, 1999.

[50] Heidi Harley and Elizabeth Ritter. Person and number in pronouns: A feature-geometric analysis. *Language*, 78(3):482–526, 2002.

[51] Michal Starke. Nanosyntax: A short primer to a new approach to language. *Nordlyd*, 36(1):1–6, 2010.

[52] Pietro Codara. *Partitions of a finite partially ordered set.* Springer, 2009.