1. Introduction

Linguists are often interested in derivations of syntactic objects. The derivations of minimalist grammars such as [Boston et al., 2010], [Stabler, 2011], or [Stabler and Keenan, 2003] are sequences of operations applying iteratively to tuples of syntactic objects. The objects that these operate over are often ‘structured sets’ in some way - linear orders of features or phonetic symbols, (labeled) graphs or orders, etc. However, unlike in the graph grammars of [Ehrig et al., 1997], the structural changes induced by operations are not part of the formal structure of these models. We generalize the intuition in Ehrig et al. that the ‘rewrite rules’ or ‘structural changes’ associated to grammatical rules are morphisms in some category. We give a construction of a category of ‘derivations’ - sequences of objects connected by structural changes - $\mathcal{D}(A)$ for any category $A$ of ‘derived objects and structural changes between them’. We then present models of derivations which are concrete - i.e. ‘categories of structured sets’ - and give a theory of structural similarity of derivations at many levels (isomorphism of derivations, derived objects, equivalence of languages, subderivations, etc.). This approach gives an alternative to [Stabler and Keenan, 2003]. It uses the ‘usual’ definitions of these notions from category theory, so we can take advantage of many traditional methods to define constructions on derived objects and derivations which respect structure.

The main goals of the paper are to present a very general calculus for describing structural changes over dependency structures (possibly with ‘extra’ data) and the rules that generate them. Series of applications of such structural changes over tuples of syntactic objects should lead to derivations of the appropriate sort. While not discussed in this paper, this generality allows for precise formulations of more subtle dependencies and rules beyond selection, such as flavors of agreement, concord, and licensing, which may be defined over feature geometries and lead to feature-sharing structures. The calculus of rules given here also allows for a theory of constraining application of rules based on the structure of the syntactic objects involved, ranging over many formal models of syntactic objects.
§2 gives a general theory of derivations for any category. We first motivate the use of categories for modeling syntactic objects and structural changes, then move on to propose a model of derivations of such objects. §3 describes derivations which are concretes in a particularly nice way which admits good theories not only of isomorphisms of derivations, but also embeddings. It also allows us to give recursive constructions on derivations by reasoning about relations between elements in a set, which leads to a theory of grammars of such derivations, described in §4. We also describe how an assignment of structural changes can be generated by a set of basic structural changes, translated along restricted contexts. This is done with a general categorical construction called a pushout, similar to the technique in [Ehrig et al., 1997], amenable for handling many different kinds of models.

We then give a very general method for extending these results to derivations with more structure, giving a model of [Boston et al., 2010]. We conclude with a discussion comparing this ‘structured’ approach to ‘algebraic’ approaches in §5.

2. General derivations

2.1. Categories of syntactic objects

In this section, we motivate the use of (concrete) categories for describing syntactic objects. Syntactic objects are often represented as trees, possibly adorned with other information, such as a (PF) linear order, feature structures, projection/headedness information, etc. We give some examples of mathematical structures (graphs, directed graphs (digraphs), preorders) which could be used to model these objects in Table 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 Table 2. 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 $\text{Hom}(A,B)$ of homomorphisms, together with for each triple $A$, $B$, and $C$ a composition function $\circ : \text{Hom}(A,B) \times \text{Hom}(B,C) \to \text{Hom}(A,C)$ constitutes a category.

Many basic facts about morphisms can be proven in any category.

Claim 1. If $x, y \in C(A, A)$ 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 $\text{Grph}$, $\text{DGrph}$, and $\text{Proset}$. 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 $\gamma$, or finite partial order $P$ as
<table>
<thead>
<tr>
<th>Example</th>
<th>Definition</th>
<th>$V = {a, b, c, d}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>A set $V$ of vertices together with a set $E$ of 2-element subsets of $V$. $E = {{a, b}, {a, c}, {c, d}}$</td>
<td></td>
</tr>
<tr>
<td>Directed Graph</td>
<td>A set $V$ of vertices together with a subset $E \subseteq V \times V$ of edges (i.e. a binary relation on $V$). $E = {(a, b), (a, c), (c, d)}$</td>
<td></td>
</tr>
<tr>
<td>Preorder</td>
<td>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 $a \leq c$ $a \leq a$, $a \leq b$, $a \leq c$, $a \leq d$, $b \leq b$, $c \leq c$, $c \leq d$, $d \leq d$</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Mathematical structures used to model dependencies
Given two graphs \((V_G, E_G)\) and \((V_H, E_H)\), a morphism \(f\) is given by a function \(f : V_G \rightarrow V_H\) such that for each \(\{a, b\} \in E_G\), we have \(\{f(a), f(b)\} \in E_H\).

Given two digraphs \((V_G, E_G)\) and \((V_H, E_H)\), a morphism \(f\) is given by a function \(f : V_G \rightarrow V_H\) such that for each \((a, b) \in E_G\), we have \((f(a), f(b)) \in E_H\).

Given two partial orders \((P, \preceq_P)\) and \((Q, \preceq_Q)\), a morphism \(f\) is given by a function \(f : P \rightarrow Q\) such that if \(a \preceq_P b\) in \(P\), then \(f(a) \preceq_Q f(b)\).

<table>
<thead>
<tr>
<th>Morphism</th>
<th>Composite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Given two graphs ((V_G, E_G)) and ((V_H, E_H)), a morphism (f) is given by a function (f : V_G \rightarrow V_H) such that for each ({a, b} \in E_G), we have ({f(a), f(b)} \in E_H).</td>
</tr>
<tr>
<td>Directed Graph</td>
<td>Given two digraphs ((V_G, E_G)) and ((V_H, E_H)), a morphism (f) is given by a function (f : V_G \rightarrow V_H) such that for each ((a, b) \in E_G), we have ((f(a), f(b)) \in E_H).</td>
</tr>
<tr>
<td>Preorder</td>
<td>Given two partial orders ((P, \preceq_P)) and ((Q, \preceq_Q)), a morphism (f) is given by a function (f : P \rightarrow Q) such that if (a \preceq_P b) in (P), then (f(a) \preceq_Q f(b)).</td>
</tr>
</tbody>
</table>

Table 2: Homomorphisms between objects

A preorder \(\preceq\) on its underlying set of nodes, such that (1) \(a \preceq a\), (2) \(a \preceq b\) and \(b \preceq c\) imply \(a \preceq c\), and (3) for any \(a, b\), either \(a \preceq b\) or \(b \preceq a\). We can construct morphisms as graph, digraph, or order-preserving morphisms \(\phi\), 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.

2. For any set \(L\), we can construct categories of (directed) graphs or preorders (partially) labeled by \(L\). Concretely, we add (partial) labelling data as a (partial) function \(f : V \rightarrow L\) from the underlying set of nodes. A morphism \(\phi\) 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 a predicate \(\alpha : V \rightarrow \{\top, \bot\}\) on the underlying nodes. For example, we could say that if \(\alpha(a) = \top\), then \(a\) is ‘inactive’. If \((A, \alpha)\) and \((B, \beta)\) are (directed) graphs or preorders with predicates, we can define a morphism to be a morphism \(\phi\) of the relevant type, such that if \(\alpha(a) = \top\), then \(\beta(\phi(a)) = \top\).

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.
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 1.** A morphism \( f : X \to Y \) in a category is called an **isomorphism** 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.

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 \( L \), with a unary predicate \( (P, \leq_P, \preceq_P, f_P : P \to L, \pi) \) and \( (Q, \leq_Q, \preceq_Q, f_Q : Q \to L, \kappa) \) is a bijection \( \phi : P \to Q \) such that (1) \( a \leq_P b \) iff \( \phi a \preceq_Q \phi b \); (2) \( a \preceq_P b \) iff \( \phi a \leq_Q \phi b \); (3) \( a \) has label \( f_P(a) \) iff \( \phi(a) \) has label \( f_Q(\phi(a)) = f_P(a) \); (4) \( \pi(a) = \top \) iff \( \kappa(\phi(a)) = \top \).

Just by observing the examples, it appears that the categories are related to each other. We can describe these relations precisely as **functors** between categories. A functor \( F : \mathcal{C} \to \mathcal{D} \) is a mapping from objects and morphisms of \( \mathcal{C} \) to those of \( \mathcal{D} \) which is compatible with composition. We have the following examples of functors:

1. \( i : \text{FProset} \to \text{DGrph} \). 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 \( \phi \) to the graph homomorphism acting the same way on underlying sets of vertices.

\[
\begin{align*}
\{a, b, c, d\} \\
a \leq a, a \leq b, a \leq c, \\
a \leq d, b \leq b, c \leq c, \\
c \leq d, d \leq d
\end{align*}
\]

2. \( j : \text{Grph} \to \text{DGrph} \). 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. $\rho : \text{DGrph} \to \text{Proset}$. To each digraph $G$, we can associate to it a preorder with underlying set of nodes $V_G$, and we take the smallest preorder $\leq_G \subseteq V_G \times V_G$ containing $E_G$, sometimes called the \textit{transitive closure} of the relation $E_G$, or the \textit{reachability relation} generated by $E_G$. If $f : G \to H$ is a morphism of digraphs, then $f$ is order-preserving. That is, if $a \leq_G b$ is in the transitive closure of $E_G$, then $f(a) \leq_H f(b)$ is in the transitive closure of $E_H$, so $f$ is order-preserving.

4. $u : \text{DGrph} \to \text{Grph}$. $u$ sends a digraph $G$ to a graph with the same underlying set of nodes, such that if $(a,b) \in E_G$, then $\{a,b\} \in E_{uG}$, unless $a = b$. It again takes a digraph homomorphism $f$ to the graph homomorphism whose underlying function on vertices is $f$.

Composition of functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{E}$ is given by $(G \circ F)(C) = G(F(C))$ for each object of $\mathcal{C}$ and $(G \circ F)(f) = G(F(f))$ for each morphism of $\mathcal{C}$.

The category $\text{Set}$ of sets and set-functions gives a category using regular function composition.

**Definition 2.** A category $\mathcal{C}$ together with a functor $U : \mathcal{C} \to \mathcal{D}$ is said to be \textbf{concrete over} $\mathcal{D}$ if $U$ is faithful. A category $U : \mathcal{C} \to \text{Set}$ concrete over $\text{Set}$ is called a \textbf{construct}. [Adámek et al., 2004], Ch. 5.

The intuition behind a construct $U : \mathcal{C} \to \text{Set}$ is that objects $C$ can be thought of as sets $UC$ ‘with extra structure’. This is because a morphism $f : C \to C'$ is totally determined
by a set function $Uf : UC \to UC'$. That $\mathcal{C}(C, C') \to \text{Set}(C, C')$ 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 $Uf$. An iso $f : C \to C'$ will always be totally determined by a bijection.

$i : \text{FProset} \to \text{DGrph}$ and $j : \text{Grph} \to \text{DGrph}$ are isomorphic to full subcategory inclusions. Each of the functors above are faithful. The functor $V_{(-)} : \text{Grph} \to \text{Set}$ taking a graph to its underlying set of vertices and a graph homomorphism to the function on vertices is faithful, making $\text{Grph}$ into a construct. Similarly, the functors mapping partial orders or digraphs to their underlying sets of vertices turn them into constructs.

In any construct, we can define an embedding. An embedding $f : A \to B$ is a morphism whose underlying set function $Uf$ is injective, such that any morphism into $B$ whose image is contained in $f(UA)$ factors through $f$ as a morphism. It formalizes ‘restricting structure to a subset’. We give examples of embeddings in the concrete categories above.

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 FPos, 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 partial orders with a precedence order, partially labeled by $L$ with a single unary predicate on the vertices. The functor $U : A \to \text{Set}$ taking $(P, \leq_P, \preceq_P, f_P : P \to L, \pi)$ to $P$ turns $A$ into a construct. A morphism $i : (P, \leq_P, \preceq_P, f_P : P \to L, \pi) \to (Q, \leq_Q, \preceq_Q, f_Q : Q \to L, \kappa)$ is an embedding iff (1) $i : P \to Q$ is an injective function; (2) $a \leq_P b$ iff $ia \preceq_Q ib$; (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) $\kappa(ia) = \top$ iff $\kappa(ia) = \top$.

The collection of sets and partial functions forms a category PSet. If $f : A \to B$ and $g : B \to C$ are partial functions, then the composite $g \circ f : A \to C$ is the partial function which is defined at $a \in A$ only if $g$ is defined at $f(a)$, in which case it maps $a$ to $g(f(a))$. In each case above, we can weaken the notion of homomorphism to one which must only be a partial function $f$ between sets of vertices, such that $f$ preserves edge relations between the vertices where $f$ is defined. We denote these categories PGrph, PProset, etc. In each case, the functor $U : A \to \text{PSet}$ is faithful, so that we can think of $A$ as concrete over PSet. Embeddings can be generalized to any concrete category, and the embeddings in PGrph and PProset are exactly the same as those in Grph and Proset.

The previous examples are intended to show that the theory of concrete categories is very general, and many models of linguistic objects can be represented as categories concrete over Set or PSet, and even over (P)(D)Grph or (P)Proset.
2.2. Structural Changes

If we think about objects of a category $A$ as syntactic objects, we may think of morphisms of $A$ as structural changes between them. In the categories of §2.1, we think of them as carrying nodes of the input object into nodes of the output object, while preserving dependency relations, precedence, etc., possibly with deletion (in the case of underlying partial functions). This approach to ‘rewrite rules’ in the case of (labeled, multi-) digraphs with partial functions between sets of vertices and edges is described in [Ehrig et al., 1997], going back to methods in [Ehrig et al., 1973]. In [Ehrig et al., 1997], a direct derivation from one graph with structure to another is a morphism $f : A \rightarrow B$ arising in a systematic way from basic production rules. We will show how morphisms can arise from generating productions in §4.

We give an example a structural change. We say that a digraph $G$ is rooted if its reachability preorder $\rho(G)$ has unique minimal element $r$, which we call its root. Let $G$ and $H$ be any digraphs with roots $r$ and $s$. We construct a digraph $\text{merge}(G,H)$ whose set of vertices is the disjoint union $V_G + V_H$ of vertices of $G$ and $H$. If $g, g'$ are vertices of $G$ with edge $(g, g')$, then $(g, g')$ is an edge in $\text{merge}(G,H)$, and similarly for edges $(h, h')$ of $H$. Additionally, we add an edge $(r, s)$, so that $\text{merge}(G,H)$ is a rooted digraph. We have a pair of digraph morphisms $f : G \rightarrow \text{merge}(G,H)$ and $f : H \rightarrow \text{merge}(G,H)$, assigning the nodes in the input graphs to nodes in the output graphs while preserving edges.

![Figure 1: A pair of graph homomorphisms attaching the root of one to the other](image)

Similarly, let $R$ and $S$ be any preorders with least elements (roots) $r$ and $s$. We take the disjoint union with ordering as above on $\text{merge}(R,S)$, but now adding all relations $r \leq x$ where $x \in S$.

Such assignments of syntactic objects to pairs appear in grammars manipulating dependency trees, such as in [Boston et al., 2010]. We add to these assignments data associating the structure of the inputs to the output as in [Ehrig et al., 1997]. However, it may seem that we have generalized to tuples of morphisms. In fact, in the above cases, the pairs of maps can be represented by a single map - the sum or coproduct of the two maps. If a pair of objects $(A, B)$ in a category has a sum, we write this sum as $A + B$. A sum comes with two ‘coprojections’ $A \rightarrow A + B$ and $B \rightarrow A + B$.

A sum of sets is given by disjoint union $A \rightarrow (A \times \{0\}) \cup (B \times \{1\}) \leftarrow B$ together with
injections \( a \mapsto (a, 0) \) and \( b \mapsto (b, 1) \). A sum of digraphs \( G + H \) is given by the digraph with set of vertices \( V_G + V_H \) with an edge \( (x, y) \) iff \( (x, y) \) is an edge of \( G \) or \( H \) with coprojections given by inclusion of vertices. A sum of preorders with precedence order,\(^2\) partially labeled by \( L \), with single unary predicate on the vertices \( (P, \preceq_P, \preccurlyeq_P, f_P : P \rightarrow L, \pi) + (Q, \preceq_Q, \preccurlyeq_Q, f_Q : Q \rightarrow L, \kappa) \) is given by the set \( P + Q \) such that (1) \( x \preceq y \) iff the relation holds in \( P \) or \( Q \); (2) identically for \( \preccurlyeq \); (3) \( x \) has label \( f(x) = f_P(x) \) iff \( x \) has label \( f_P(x) \) in \( P \) or \( f(x) = f_Q(x) \) iff \( x \) has label \( f_Q(x) \) in \( Q \); (4) \( (\pi + \kappa)(x) = \top \) iff \( \pi(x) = \top \) or \( \kappa(x) = \top \).

If a category has sums for every pair of objects, the pair of maps \((s_1, s_2)\) can be represented by the single map \( u \), written \( s_1 + s_2 \), which we also call their sum. In Fig. 1, the pair of maps can be summed in the obvious way, and hence represented by a single map \( f + g : A + B \rightarrow \text{MERGE}(A, B) \). Analogous results hold in other categories.

2.3. Example from literature

We conclude this section with a ‘categorification’ of a proposed model of syntactic objects from the literature. In [Stabler, 1996], trees are formalized as expressions. We simplify his definitions here. Given vocabulary items \( V \) and a set of base syntactic types \( \text{base} \), Stabler constructs a collection \( L \) of labels. The syntactic features are partitioned as \( F = (\text{base} \cup \text{select} \cup \text{licensors} \cup \text{licensees}) \), defined as follows.

\[ \text{select} = \{ -x \mid x \in \text{base} \}; \text{licensees} = \{ +x \mid x \in \text{base} \}; \text{licensors} = \{ +x \mid x \in \text{base} \}. \]

The set of labels are the regular expressions \( L = F^* \cdot V^* \). Expressions over \( L \) are defined as trees \( \tau = (N_\tau, \prec_\tau, \preceq_\tau, \prec_\tau, \text{label}_\tau) \) such that

1. \( \prec_\tau \) is a preordering on \( N_\tau \) ‘dominates’
2. \( \preceq_\tau \) is a preordering on \( N_\tau \) ‘precedes’
3. \( \prec_\tau \) is a preordering on \( N_\tau \) ‘projects over’
4. \( \text{label}_\tau : L_\tau \rightarrow L \) is a function, where \( L_\tau \) are the leaves of \( N_\tau \) (nodes which only dominate themselves).

Such that

1. For any \( x \in N_\tau \), the set of nodes \( \{ y \in N_\tau \mid y \prec_\tau x \} \) is a linear order, and there is a unique element \( r \) such that \( r \prec_\tau a \) 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. \( (\forall w, x, y, z) \cdot (x \preceq_\tau y \land x \prec_\tau w \land y \prec_\tau z) \rightarrow (w \preceq_\tau z) \)

\(^2\)If we allow \( \preceq \) to be an arbitrary preorder.
4. \((\forall x) : ((\exists y)(x < y)) \rightarrow ((\exists y)(\forall z \neq y)(x < z \rightarrow y < z))\), where \(<\) and \(<\) are the immediate ‘dominates’ and ‘projects over’ relations.

Morphisms \(f : \tau \rightarrow \sigma\) should be functions \(f_N : N_\tau \rightarrow N_\sigma\) which at least have the following properties: (1) \(a <^*_\tau b\) implies \(f_Na <^*_\sigma f_Nb\); (2) \(a \leq^*_\tau b\) implies \(f_Na \leq^*_\sigma f_Nb\); and (3) \(a <^*_\tau b\) implies \(f_Na <^*_\sigma f_Nb\), 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 \(\tau\) and \(\sigma\) in that precedence order, where the root of \(\tau\) immediately projects over that of \(\sigma\), with all of the relevant relations added to meet the axioms. We define a partial binary operation on expressions using Stabler’s terminology.\(^3\)

\[
\text{merge}(\tau, \sigma) =
\]

\[i. \ [<_\tau', \sigma'] \text{ if } \tau \text{ is a head with initial feature } = x \text{ and } \sigma \text{ has feature } x. \text{ Here, } \tau' \text{ is } \tau, \text{ but where the string of features } \alpha \text{ on the head of } \tau \text{ has had } = x \text{ removed from the front, and similarly } x \text{ is removed from the front of the string of features } \beta \text{ on the head of } \sigma. \text{ (complement merger)}
\]

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

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

For \(\text{merge}\), the inclusions of unlabeled trees \(\tau \rightarrow \text{merge}(\tau, \sigma)\) and \(\sigma \rightarrow \text{merge}(\tau, \sigma)\) preserve \(<^*_\tau\), \(\leq^*_\tau\), and \(<^*_\sigma\), and in fact their immediate variants. A leaf \(x\) with label \(\alpha \cdot \pi \in F^* \cdot V^*\) may have its its initial feature removed. With \(\text{move}\), we have an map of unlabeled trees \(\tau \rightarrow \text{move}(\tau)\), sending all nodes in \(\tau_0\) to a dummy ‘trace’ node; however, all features in \(\tau_0\) are removed under this map. We also have a map \(\tau_0 \rightarrow \text{move}(\tau)\) embedding \(\tau_0\) into the moved position, with \(-x\) removed.

We define a morphism \(f : \tau \rightarrow \sigma\) to be a map of unlabeled trees such that if \(x \in L_\tau\) has label \(\alpha \cdot \pi\), then \(f(x)\) is in \(L_\sigma\) with label \(\alpha' \cdot \pi'\), where \(\alpha'\) is \(\alpha\) with some initial substring removed, and \(\pi'\) is \(\pi\) with some initial substring removed. This will allow composites.

\(^3\) Given a tree \(\tau, x, y \in N_\tau, x\) is a head of \(y\) iff either (1) \(y\) is a leaf and \(x = y\); or (2) \((\exists z)(y < z \land (\forall w)(y < w \rightarrow z <^*_w \land x \text{ is a head of } z))\). Every tree \(\tau\) has a root \(\tau\), and \(\tau\) has a unique head \(y\), which is a leaf. This leaf is labeled by \(\text{label}_\tau(y) \in \mathcal{L}\), which has as first part a string of features \(\alpha \in F^*\). A tree \(\tau\) has feature \(f\) if the first symbol in the string \(\alpha\) is \(f\). A maximal projection \(m\) in \(N_\tau\) is an element minimal (with respect to \(<^*_\tau\)) amongst the nodes with head equal to the head of \(m\). \(\tau\) is a head if it consists of one node, otherwise it is complex.
of structural changes under 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 \( \tau \) with a trace node while deleting features to be a morphism. We call this category \((\text{Strict})\text{MGExp}\). The functor \( N_{(-)} : \text{SMGExp} \to \text{Set} \) turns \( \text{SMGExp} \) 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 \( \pi \) and specific features \( \alpha \) 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.

2.4. Derivations over a category \( A \)

[Ehrig et al., 1997] 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. These morphisms also give relations such as isomorphisms and embeddings between syntactic objects. We want to consider derivations on categories \( A \). 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 could formalize a derivation using the following definitions.

**Note 1.** Any preorder \((P, \leq)\) 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.** A diagram of shape \( P \) is a functor \( F : P \to A \). 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 \( !_{q,p} \circ !_{p,r} = !_{q,r} \).

If \( p \leq q \) and \( q \leq p \), then \( !_{p,q} \) and \( !_{q,p} \) are mutually inverse isomorphisms. If \( p \leq q \) and \( q \leq p \), we call them equivalent and write \( p \approx q \). We think of \( P \) as a ‘state space’ of a derivation, and \( F \) assigns to each state an \( A \)-object or ‘derived syntactic object (DSO)’ and to each ‘transition’ in \( P \) a ‘net structural change morphism’. Then, if two states are equivalent, their associated DSOs are isomorphic, and the structural changes are mutually inverse isomorphisms. Further, every map into or out of \( p \) in the derivation corresponds to an isomorphic map into or out of the other. So, we may as well not distinguish them.

**Definition 4.** A partial order (poset) is a preorder \((X, \leq)\) such that \( x \approx y \) implies that \( x = y \). An fposet is a poset with finite underlying set.

**Definition 5.** Given a category \( A \), an \( A \)-derivation is a functor \( F : P \to A \), where \( P \) is a finite partial order.
We can define a language to be a subclass consisting of just one step - the object is isomorphic to a subcategory of \( A \):

\[
\text{Definition 6.}
\]

Two languages are equivalent iff for every derivation \( \Delta \) and \( \delta \) in \( A \), we can define \textit{strong extensional equivalence} between languages. Two languages are equivalent iff for every derivation \( \Delta \) in \( A \) and conversely.

\[
\text{A is isomorphic to a subcategory of } \mathcal{D}(A), \text{ taking each DSO } a \text{ in } A \text{ to the derivation consisting of just one step - the object } a \text{ itself.}
\]

\[4\]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 \( n \) argument slots, but the slots are not linearly ordered.

\[5\]Identically, an iso of preorders
Claim 5. The map \( i : A \to \mathcal{D}(A) \) sending \( a \mapsto (*,!_a : * \to A) \), where \(*\) is a one-point poset, and \(!_a\) is the functor 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 \( a \).

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 \(A\) 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 \(\top : \mathcal{D}(A) \to A\), which has the property that derivation morphisms from \((P,F)\) to \((*,!_a)\) are determined totally by \(A\)-morphisms \(\top(P,F) \to a\).

Definition 7. \(i : A \to \mathcal{D}(A)\) has a left adjoint if and only if \(A\) has all finite colimits,\(^6\) in which case we denote it \(\top\), and we say it is the \textit{yield functor} of \(\mathcal{D}(A)\).

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 \(F_r + G_s\). We now investigate ‘underlying sets’ of derivations.

For any category \(A\) and object \(a\), we define the functor \(y(a) \equiv A(a,-) : A \to \textbf{Set}\). \(y(a)\) returns for each object \(b\) of \(A\) the set \(A(a,b)\). We say that a functor \(U\) is \textit{representable by} \(a\) if it is naturally isomorphic to \(y(a)\).

1. The forgetful functor \(N(\_ \to \_ ) : \textbf{DGrph} \to \textbf{Set}\) is represented by \(1\), the digraph with one node and no edges.

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

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

In each case, the set of maps \(\bullet \to A\) from some fixed object \(\bullet\) into \(A\) gives the underlying set of \(A\). Since \(\bullet\) is itself also a derivation, we can extend the construction to get a functor \(U : \mathcal{D}(A) \to \textbf{Set}\), where \(U\) is the functor \(\mathcal{D}(A)(i(\bullet),- )\). In each case, a ‘point’ of a derivation \(x : \bullet \to (P,F)\) picks out a stage \(p \in P\) along with an element \(x\) of \(F_p\). Whenever \(U : A \to \textbf{Set}\) is concrete and represented by \(\bullet\), we can call a map \(x : \bullet \to (P,F)\) a point. \(\mathcal{D}(A)(\bullet, (P,F))\) is the set of points of \((P,F)\), and consists of the disjoint union of all \(F_p\). Whenever \(U : A \to \textbf{Set}\) is concretely representable, it makes sense to define a \textit{projection relation} on the points of \((P,F)\).

---

\(^6\)The standard definition of colimits can be found in [Mac Lane, 1971] III. 3 and [Borceux, 1994a] 2.6.6.
Definition 8. Let $U : A \to \text{Set}$ be a construct represented by $\bullet$. 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 derived spaces preserved under iso (and in fact arbitrary morphisms). We can also 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 iso of derivations.

However, $\mathcal{D}(A)$ is rarely representably concrete, even when $A$ is. For example, when $A = \text{Grph}$, 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 $\Box$ 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.\footnote{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 $x, y : 1 \to G$, the functor represented by the empty graph is not faithful. We can embed $\text{Grph}$ in $\mathcal{D}(\text{Grph})$ and recreate the problem.} However, if $\Box$ has any nonempty DSO, then there are no maps from $\Box$ 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.

In the next section, we construct a category $\text{Der}$ similar to $\mathcal{D}(\text{FPos})$ such that the functor taking each derivation to its set of points turns it into a construct. Notably, this means that we discard empty states, since they will not have any points associated to them. We will otherwise generalize the category slightly so that it has well-behaved concrete properties.

3. A representably concrete category of derivations

The previous section showed that categories $A$ (concrete over $\text{Set}$ or $\text{PSet}$) make for good models of syntactic objects and structural changes between them. We constructed a category $\mathcal{D}(A)$ of derivations over $A$ which are diagrams of objects from $A$ connected by structural changes from $A$. However, even when $A$ is a representable construct, $\mathcal{D}(A)$ need not be. We will construct certain categories of derivations which are representably concrete by relating the state diagram to the derived objects.

We denote by $\text{FPos}$ the category of fposets and order-preserving functions. For any partial order $(X, \leq)$ and $x \in X$, let $U_x = \{ y \in X \mid x \leq y \}$.
**Definition 9.** A **derivation** $\Delta$ 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 $!_x : U_x \rightarrow \top_x$. The assignments in (3) must meet the condition: if $y \leq x$, then there exists an order-preserving function $f_{y,x} : \top_x \rightarrow \top_y$ 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 6.** Define a **morphism of derivations** $\phi : \Delta \rightarrow \Gamma$ to be (1) a function $|\phi| : |\Delta| \rightarrow |\Gamma|$; (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 $\phi_x : \top_x \rightarrow \top_{\phi(x)}$ such that $!_{\phi(x)} \circ \phi = \phi_x \circ !_x : U_x \rightarrow \top_{\phi(x)}$. The class of derivations with these morphisms forms a category $\mathbf{Der}$.

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

**Claim 7.** The functor $|\cdot| : \mathbf{Der} \rightarrow \mathbf{Set}$ is faithful (hence $\mathbf{Der}$ is a construct). It is represented by $1$, 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 $\Delta$, and each derived object is associated to an embedding of fposets.

**Claim 8.** 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 $\top_a^S = \{ k \in \top_a^\Delta : (\exists x \in S) : (a \leq x) \wedge (!_a(x) = k) \}$, with the ordering $k \leq c$ in $\top_a^S$ iff the relation holds in $\top_a^\Delta$. Construct the map $i_a^S : \{ x \in S : a \leq x \} = S_a \rightarrow \top_a^S$ sending each $x \in S_a$ to $!_a(x)$. This defines a derivation $S$ with underlying set $S$, and the function $S \hookrightarrow |\Delta|$ underlies a morphism $i : S \hookrightarrow \Delta$, and this morphism is an embedding. Furthermore, any embedding $j : \sigma \hookrightarrow \Delta$ is isomorphic to one of this form.

**Claim 9.** Define **projection** as a relation on the points of $\Delta$: $x \sqsubseteq y$ if and only if $f_{x,y} : \top_y \rightarrow \top_x$ maps the root of $\top_y$ to the root of $\top_x$. Any morphism $\phi : \Delta \rightarrow \Gamma$ preserves projection, in the sense that $x \sqsubseteq \Delta y$ implies $\phi x \sqsubseteq \Gamma \phi y$. For a subderivation $S \hookrightarrow \Delta$, and points $x, y \in S$, $x \sqsubseteq \Delta y$ additionally implies $x \sqsubseteq_S y$.

**Claim 10.** Let $\phi : \Delta \rightarrow \Gamma$ be a morphism of derivations.

1. $\phi$ 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 $\phi_x : \top_x \rightarrow \top_{\phi x}$ is an isomorphism; (4) inducing isomorphisms of structural changes as in the diagram:
2. $\phi$ is a monomorphism iff its underlying set function is an injection

3. $\phi$ is an epimorphism iff its underlying set function is a surjection

Claim 11. Der is finitely complete and cocomplete. In particular, finite completeness 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 $!:\Delta \to 1$ sending all points to the single point of 1.

2. For any pair of derivations $(\Delta, \Gamma)$, a product derivation $\Delta \times \Gamma$
   A product of $\Delta$ and $\Gamma$ 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 $(d, g)$, $T_{(d,g)} = T_d \times T_g = \{(k, c) \mid k \in T_d, c \in T_g\}$ with the ordering $(k, c) \leq (k', c')$ iff $k \leq_{\Delta} k'$ in $T_d$ and $c \leq_{\Gamma} c'$ in $T_g$. Note that $U_{(d,g)} = U_d \times U_g$, and we give the map $!_{(d,g)} : U_d \times U_g \to T_d \times T_g$ 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 $a, b : \Delta \Rightarrow \Gamma$, an equalizer $s : S \to \Delta$
   The equalizer $S$ is constructed as the subderivation on $S = \{x \in \Delta \mid a(x) = b(x)\}$, with $s$ the associated embedding. In fact, every subderivation embedding arises as an equalizer.

while finite cocompleteness means that Der has

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

2. For any pair of derivations $\Delta, \Gamma$ 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 $\Delta$ or $\Gamma$, and we associate $T_x$ with the derived object in $\Delta$ or $\Gamma$. The subset inclusions into the disjoint union $\Delta, \Gamma \hookrightarrow \Delta + \Gamma$ give coproduct inclusions.
3. For any pair of morphisms \(a, b : \Delta \rightarrow \Gamma\) a \textbf{coequalizer} \(s : \Gamma \rightarrow \tilde{\Gamma}\)

Construction of coequalizers is somewhat more involved, and will not be used in this paper, so we omit it. We have another result which shows that embeddings lead to the ‘right’ notion of subderivation.

\textbf{Claim 12.} The classes \textbf{Epi} (equivalently, surjections) and \textbf{Emb} (embeddings) form a factorization system for \textbf{Der}. That is, every morphism \(\phi : \Delta \rightarrow \Gamma\) factors as an epimorphism followed by a regular monomorphism \(\Delta \rightarrow \text{im}(\phi) \hookrightarrow \Gamma\) up to unique isomorphism of embeddings, where \(\text{im}(\phi) = \{g \in \Gamma \mid (\exists d \in \Delta) : \phi(d) = g\}\).

Finally, we want to show that \(\mathbf{FPos} \hookrightarrow \mathbf{Der}\) has an obvious inclusion, akin to \(\mathbf{A} \hookrightarrow \mathcal{D}(\mathbf{A})\), and a left adjoint \textit{yield} functor.

\textbf{Claim 13.} The map \(i : \mathbf{FPos} \hookrightarrow \mathbf{Der}\) (1) sending \((P, \leq) \mapsto (P, \leq_P, id)\) where for each \(x \in P\), \(id_x : U_x \rightarrow \top_x\) is an iso; and (2) sending \(f : P \rightarrow Q\) 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 \(\top : \mathbf{Der} \rightarrow \mathbf{FPos}\), and we denote \(\top(\Delta)\) as \(\top\Delta\). When \(\Delta\) has root \(r\), then \(\top\Delta \cong \top_r\). Similarly, if \(\Delta\) is the sum of rooted derivations \(\Delta_1 + \ldots + \Delta_n\) with roots \(r_1, \ldots, r_n\), then \(\top\Delta \cong \top_{r_1} + \ldots + \top_{r_n}\).

\textbf{3.1. Constituency}

We have seen many nice properties of \(\mathbf{Der}\) as a concrete category: isos 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 \(\mathbf{Der}\)-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 had 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. Even when they are, we usually want to group some points together into those which we think of as being ‘part of the same step’. Consider the following derivation.
Let $X = \{1, 2, 3, 4\}$ together with the partial ordering in Fig. 2. The maps $!_1 : U_1 \rightarrow \top_1$ and $!_2 : U_2 \rightarrow \top_2$ must be isos since they must be surjective. We let $!_3 : U_3 \rightarrow \top_3$ be the map sending both 3 and 2 to a single point. Finally, $!_4 : U_4 \rightarrow \top_4$ 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 $\mathcal{O}(\mathbb{FPos})$ in Fig. 2. We intend to 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 10.** 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 $\phi : \Delta \rightarrow \Gamma$ is **coherent** if $x \preceq y$ implies that $\phi x \preceq \phi y$. We say that $\Delta$ is **transitive** if $\preceq$ is transitive.

**Definition 11.** Let $X$ be any finite set and $\leq$ any preordering on it. Call a subset $U \subset X$ **open** if $x \in U$ and $x \leq y$ imply $y \in U$. Call a subset $S \subset X$ with $\leq$ restricted to $S$ **connected** if $S = U \cup V$ with $U, V$ open in $S$ implies that $U \cap V \neq \emptyset$. Call $K \subset X$ a **connected component** if it is maximal amongst connected subsets with respect to subset inclusions.

**Claim 14.** Let $(X, \leq)$ be any finite preorder, and let $\kappa(X) = \{K_1, \ldots, K_n\}$ be its set of connected components. Then the $K_i$ give a partition of $X$, and each $K_i \subset X$ is open.

If $\Delta$ is transitive, then closeness is a partial ordering on $|\Delta|$. In this case, we denote the set of components with respect to $\preceq$ as $T(\Delta)$. For any transitive derivation, there is a function $q_\Delta : |\Delta| \rightarrow T(\Delta)$ taking a point $x$ to the $\preceq$-component it belongs to. The ordering $\preceq$ on $|\Delta|$ induces a preordering on $T(\Delta)$: if $x \preceq y$ in $\Delta$, then we declare $q_\Delta(x) \preceq q_\Delta(y)$ on $T(\Delta)$, and we take the smallest preordering on $T(\Delta)$ containing these relations.
Definition 12. If $\Delta$ is transitive, and the induced ordering on $T(\Delta)$ is a partial order, then we say that $\Delta$ is separated. Let $K$ be an element of $T(\Delta)$, ordered by $\leq$. If $K$ has a $\leq$-least element (root) $t$, we say that $t$ is a term. If $\Delta$ is separated and each component $K$ of $T(\Delta)$ has a root, we say that $\Delta$ is separated by terms.

Claim 15. Let $\Delta$ and $\Gamma$ be separated by terms. Then $\Delta \times \Gamma$ and $\Delta + \Gamma$ are separated by terms. Furthermore, $T(\Delta + \Gamma) = T(\Delta) + T(\Gamma)$ and $T(\Delta \times \Gamma) = \{ K \times C \mid K \in T(\Delta), C \in T(\Gamma) \}$.

Claim 16. Let $\phi: \Delta \rightarrow \Gamma$ be a coherent morphism between transitive derivations. Then there is a unique function $T(\phi): T(\Delta) \rightarrow T(\Gamma)$ such that $q_\Gamma \circ \phi = T(\phi) \circ q_\Delta$, and it is order-preserving with respect to the induced preorders on $T(\Delta)$ and $T(\Gamma)$.

We can associate subsets of derived objects to subderivations, and closeness is well-behaved on those subderivations.

Claim 17. Let $\mathcal{K} \hookrightarrow \Delta$ be any coherent embedding. If $\Delta$ is transitive, then $\mathcal{K}$ is transitive. Let $\Delta$ 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 $K^{-1} = \{ y \in \Delta \mid (x \leq y) \land (!x(y) \in K) \}$ is always coherent.

We seek to characterize fposet trees and forests, constituent-preserving maps, and factorization of forests into trees. We then generalize these results to derivations.

Definition 13. Call a partial order $X$ rooted if it has a unique minimum element. $X$ is a forest if every connected open subset of $X$ is rooted. A forest is a tree if it is connected. Call an open connected subset of a tree $T$ a constituent.

Claim 18. For any forest $X$, each component $K_i \in \kappa(X)$ is a tree. Furthermore, $X$ factors uniquely as the sum of trees $X = K_1 + \ldots + K_n$.

Definition 14. A function $f: X \rightarrow Y$ between preorders is called an open map if for every open subset $U \subset X$, $f(U) = \{ y \in Y \mid (\exists x \in U) : f(x) = y \}$ is an open subset of $Y$.

Claim 19. Let $f: X \rightarrow Y$ be an order-preserving function between trees. Then $f$ is an open map iff for each constituent $K \subset X$, $f(K)$ is a constituent of $Y$.

For any subset $S \subset X$ of a preorder $(X, \leq)$, denote $(S) = \{ y \in X \mid (\exists x \in S) : x \leq y \}$

Definition 15. Let $\Delta$ be separated (by terms). We say that $\Delta$ is a forest (of terms) if for any components $K, C \in T(\Delta)$, we have $(K) \cap (C) = (K), (C), \text{ or } \emptyset$. 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 $K \in T(\Delta)$.

Definition 16. Call $\phi: \Delta \rightarrow \Gamma$ open if $|\phi|: (|\Delta|, \leq_\Delta) \rightarrow (|\Gamma|, \leq_\Gamma)$ is an open map.

Claim 20. Let $\phi: \Delta \rightarrow \Gamma$ be an open morphism between arbitrary derivations. Then the induced map $\top_x \rightarrow \top_{\phi x}$ is surjective for each $D \in \mathcal{O}(\Delta)$.
Definition 17. Let \( \phi : \Delta \to \Gamma \) be a morphism between separated derivations. We say that \( \phi \) is **constituent-preserving** if (1) \( \phi \) is coherent; (2) \( \phi \) 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 \( \Gamma \) containing the image of \( K \).

Claim 21. If \( \Delta \) is a forest, then \( T(\Delta) \) is a forest. If \( \Delta \) is a tree, then \( T(\Delta) \) is a tree. 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.

Claim 22. Let \( \Delta \) be separated (by terms), and let \( U \subset \Delta \) be an open subset such that \( \Delta/U \hookrightarrow \Delta \) is a constituent-preserving embedding. Then \( \Delta/U \) is separated (by terms). Furthermore, if \( \Delta \) is a forest (of terms), then so is \( \Delta/U \).

It is an immediate corollary that when \( \Delta \) 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 \( \Delta \). In this way, each constituent-preserving embedding corresponds uniquely to a family of derivational constituents of \( \Delta \). 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 \( \Delta \).

3.2. Adding structure to \textbf{Der}

We are often interested in adding data like precedence, syntactic type, etc. to DSOs. We want to know conditions under which we can add data to DSOs of objects of \textbf{Der} such that the associated category is still concrete. Let \( U : A \to \text{FPos} \) be any faithful functor. We construct \( \text{ADer} \) as the category whose objects are partial orders \((|\Delta|, \leq_\Delta)\) together with for each point \( x \in \Delta \) an \( A \)-object \( T_x \), together with a surjective order-preserving function \( U_x : T_x \to U(T_x) \). Additionally, the induced order-preserving maps \( f_{y,x} : U(T_x) \to U(T_y) \) must exist and underlie \( A \)-morphisms \( T_x \to T_y \). A morphism of \( \text{ADer} \) is just like a morphism in \textbf{Der}, except the local maps \( \phi_x : U(T_x) \to U(T_{\phi x}) \) must underlie \( A \)-morphisms. Hence, \( \text{ADer} \) is also a construct.

\( \text{ADer} \) always has coproducts, computed exactly like those in \textbf{Der}. We can put various other ‘niceness’ constraints on \( A \) and \( U \) to allow us to perform certain constructions on \( \text{ADer} \), or to ensure that \( \text{ADer} \) is representably concrete. We look at a method for recursively constructing derivations and how to generalize the method to other categories in the next section.

---

\(^8\)Where \( \Delta/U \) denotes the subderivation on \( U \).
4. Operations, rules, extensions, and grammars

Consider an binary algebraic operation \( \otimes : A \times A \to A \). An algebraic operation assigns to each pair \((a, b)\) of elements of \(A\) an element \(c = a \otimes b\) of \(A\). For example, the \textsc{merge} operation on dependency trees \(R\) and \(S\) could return the tree \textsc{merge}(\(R, S\)) as defined in §2.2. However, we are not only interested in the object \textsc{merge}(\(R, S\)) itself: we want to view \(R, S, \) and \textsc{merge}(\(R, S\)) as ‘structured objects’ in some category \(A\), and we would like a pair of morphisms \(f : R \to \textsc{merge}(R, S)\) and \(g : S \to \textsc{merge}(R, S)\) associating the structure of the input objects with the structure of the output object. When \(A\) has coproducts, this can be represented with a single map \(R + S \to \textsc{merge}(R, S)\).

Grammatical operations are not algebraic operations in at least two other ways: (1) grammatical operations may not be defined for every tuple of objects; and (2) for a fixed tuple \(a = (a_1, \ldots, a_n)\) of syntactic objects and grammatical operation \(G\), there may be multiple distinct objects which we think of as arising from applying \(G\) to \(a\). Consider the following scenario: \(X^0\) is merging with \(YP\), such that \(X^0\) has a probe, and \(YP\) has multiple equidistant goal features which are all closest to \(X^0\) amongst potential goals. Distinct SOs can be formed by \textsc{agree} for each goal. The goal of this section is to find a systematic assignment of structural changes to tuples of syntactic objects, which we call a \textit{rule}. We then construct languages recursively.

4.1. Rules and operations

Let \(A^n\) be the category of \(n\)-tuples of objects from \(A\), together with \(n\)-tuples of \(A\)-morphisms \((\phi_1, \ldots, \phi_n) : (A_1, \ldots, A_n) \to (B_1, \ldots, B_n)\) with \(\phi_i : A_i \to B_i\). Suppose that \(A\) has coproducts. For a tuple \(A = (A_1, \ldots, A_n)\), denote \(\overline{A} = A_1 + \ldots + A_n\), and for a map \(\phi = (\phi_1, \ldots, \phi_n)\), denote \(\overline{\phi} = u\) for the unique map from the coproduct \(\overline{A}\) to \(\overline{B}\) induced by the maps \(\kappa_i \circ \phi_i\), where \(\kappa_i : B_i \to \overline{B}\) is the coprojection of the \(i^{th}\) coordinate into the coproduct \(\overline{B}\). See Fig. 3
We are then interested in subcategories $\mathcal{C} \subset \mathbf{A}^n$ and assignments $G : \mathcal{C} \rightarrow \mathbf{Set}$ which return for each tuple $A$ a set $G(A)$ of $\mathbf{A}$-morphisms $h : \overline{A} \rightarrow Z$. We are only interested in the structural changes up to iso. Two operations $h : \overline{A} \rightarrow Z$ and $g : \overline{A} \rightarrow Y$ are isomorphic if there is an iso $k : Z \rightarrow Y$ such that $kh = g$. Given an operation $h : \overline{A} \rightarrow Z$ on $A$ and a $\mathcal{C}$-morphism $\phi : A \rightarrow B$, we have a natural translation from $h$ to an operation on $B$ called the pushout of $h$ along $\phi$.

**Claim 23.** We construct a pushout in $\mathbf{FProset}$. Let $h : A \rightarrow Z$ and $u : A \rightarrow B$ be order-preserving functions between preorders. Take the coproduct of preorders $B + Z$. We construct a binary relation on $B + Z$: if there exists an $a \in A$ such that $u(a) = h(a)$, then we add the relation $(u(a))R(h(a))$ under the coprojections $u(a) \in B \rightarrow B + Z$ and $h(a) \in Z \rightarrow B + Z$. We take the smallest equivalence relation $E$ containing $R$. We take the underlying set of the pushout to be $(B + Z)/E$, the set of $E$-equivalence classes of $B + Z$.

We have a pair of set-functions $B \rightarrow (B + Z)/E$ and $Z \rightarrow (B + Z)/E$, where $B + Z \rightarrow (B + Z)/E$ is the function sending each element to its equivalence class. We add order relations to $(B + Z)/E$: for each $b \leq b'$ in $B$, we demand that $[b] \leq [b']$ in $(B + Z)/E$, where $[x]$ is the equivalence class containing $x$. Similarly for pairs $z \leq z'$ in $Z$. We then take the smallest preorder $\leq_{A^+}$ containing these relations. The maps $B \rightarrow ((B + Z)/E, \leq_{A^+})$ and $Z \rightarrow ((B + Z)/E, \leq_{A^+})$ are order-preserving, and give a pushout. The map from $B$ to the pushout is the pushout of $h$ along $u$.

**Claim 24.** We construct a pushout in $\mathbf{FPos}$. Let $h : A \rightarrow Z$ and $u : A \rightarrow B$ be order-preserving functions between partial orders. For any preorder $P$, we call two elements equivalent if $x \leq y$ and $y \leq x$, and write $x \approx y$. $\approx$ is an equivalence relation on $P$. The
function $P \to P/ \approx$ sending each element $x$ to its equivalence class induces an ordering on $P/ \approx$: for each $x \leq y$ in $P$, then we add the relation $[x] \leq [y]$ in $P/ \approx$. We take the smallest preorder containing these relations, call it $\leq$. $\leq$ is a partial ordering on $P/ \approx$, and we call this partial order the soberification of $P$ and denote it $[P]$. The pushout of $h$ and $u$ in $\text{FPos}$ is the pushout of them in $\text{FProset}$, followed by composing the pushout $P$ with the map to its soberification $P \to [P]$.

The underlying functions $h$ and $u$ give instructions about elements of $B$ and $Z$ which should be identified. The pushout of fposets is the ‘simplest’ fposet which identifies the requisite points with the induced order relations, then closes the ‘proto-partial order’ under transitivity, reflexivity, and antisymmetry. Any object isomorphic to the pushout above is a pushout.

We give an example. Take the fposets $\{a\}$ and $\{b\}$ and function $h : \{a\} + \{b\} \to \{a < b\}$. Let $R$ and $S$ be any fposets with roots $r$ and $s$, and let $u : \{a\} + \{b\} \to R + S$ be the map sending $a$ to $r$ and $b$ to $s$. The pushout of $h$ along $u$ is the function $h' : R + S \to P$, where the set underlying $P$ is the disjoint union of the sets underlying $R$ and $S$, with $x \leq y$ in $R + S$ if the ordering holds between them in $R$ or $S$ together with order relations $r \leq x$ for each $x \in S$. The pushout of $h$ along the sum of any maps sending $a$ and $b$ to the roots of $R$ and $S$ (if they exist), gives the desired ‘root attachment’ operation.

Definition 18. Let $A$ be a category with finite coproducts. Let $C \subset A^n$ be a category of tuples of syntactic objects. A rule on $C$ is a functor $G : C \to \text{Set}$ such that:

1. For each $A \in C$, $G(A)$ is a set of isomorphism classes of $A$-morphisms $f : \overline{A} \to Z$.
2. Let $\phi : A \to B$ be a $C$-morphism and let $f : \overline{A} \to Z$ be an element of $G(A)$. Then there is a (necessarily unique) element $f' : \overline{B} \to Z'$ in $G(B)$ such that there is a pushout diagram

$$
\begin{array}{ccc}
\overline{A} & \xrightarrow{\overline{\phi}} & \overline{B} \\
\downarrow f & & \downarrow f' \\
Z & \xrightarrow{p} & Z'
\end{array}
$$

and $G(\phi) : G(A) \to G(B)$ maps $f$ to $f'$. Note that $\overline{\phi}$ and $f$ determine $f'$ and $p$.

$A = \text{FPos}$ is one such category. We immediately generalize the case of rules on $\text{FPos}$ to $\text{Der}$. Recall that there is an inclusion $i : \text{FPos} \to \text{Der}$ with left adjoint $\top$. This means that for each $\Delta$, there is a morphism $\text{yield}_\Delta : \Delta \to \top_\Delta$ such that any operation $h : \Delta \to Z$ from a derivation to a DSO factors uniquely through the yield $\Delta \to \top_\Delta \to Z$.

Claim 25. Let $h : \Delta \to Z$ be any operation, and let $\phi : \Delta \to \Gamma$ be any morphism. Then there is a partial order $P$ 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 partial order and $k : \Gamma \to Q$ and $g : Z \to Q$ any pair of derivation morphisms such that $gh = k\phi$, then there is a unique order-preserving $u : P \to Q$ such that $k = uh'$ and $g = uf$.

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 partial order, 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 **fposet-universal** or just **universal** with respect to $\phi$ and $h$. We often abuse terminology and call $h'$ the pushout of $h$ along $\phi$.

We can give a concrete construction. $h$ and $h'$ are totally determined by maps $\top : \Delta \to Z$ and $\top : \Gamma \to P$, so we may just apply $\top$ to the legs of the diagram and take the pushout in $\textbf{FPos}$.

**Claim 26.** The universal construction above has a ‘pushout lemma’. That is, given commuting squares as below with the left square universal, the right square is universal if and only if the whole square is universal.

\[
\begin{array}{ccc}
\Delta & \xrightarrow{\phi} & \Gamma \\
\downarrow{f} & & \downarrow{g} \\
X & \xrightarrow{a} & Y
\end{array} \quad \begin{array}{ccc}
\Gamma & \xrightarrow{\psi} & \Sigma \\
\downarrow{h} & & \\
Y & \xrightarrow{b} & Z
\end{array}
\]

Fix a subcategory $\mathcal{C} \subset \textbf{Der}^n$ thought of as **conditions** on the arguments of the rule $G$. Denote objects $\Delta \equiv (\Delta_1, \ldots, \Delta_n) \in \mathcal{C}$ and morphisms $\phi \equiv (\phi_1, \ldots, \phi_n) : (\Delta_1, \ldots, \Delta_n) \to (\Gamma_1, \ldots, \Gamma_n)$. For each $\mathcal{C}$-object, denote $\overline{\Delta} \equiv \Delta_1 + \ldots + \Delta_n$, and for each morphism $\phi : \Delta \to \Gamma$, denote $\overline{\phi} : \overline{\Delta} \to \overline{\Gamma}$ for the sum of maps $\kappa_i \phi_i$, where $\kappa_i : \Gamma_i \to \overline{G}$ are the coprojections.

**Definition 19.** A **$\mathcal{C}$-rule** $G : \mathcal{C} \to \textbf{Set}$ is a functor with the following properties:

1. For each $\Delta \in \mathcal{C}$, $G(\Delta)$ is a set of isomorphism classes of maps $f : \overline{\Delta} \to Z$ from $\overline{\Delta}$ to some partial order $Z$.

2. Let $\phi : \Delta \to \Gamma$ be a $\mathcal{C}$-morphism and let $f : \overline{\Delta} \to Z$ be an operation in $G(\Delta)$. Then there is a (necessarily unique) element $f' : \overline{\Gamma} \to Z'$ in $G(\Gamma)$ such that there is an fposet-universal diagram.
\[
\begin{array}{c}
\Delta \\ \downarrow f \\
Z \\ \downarrow p \\
\end{array}
\xrightarrow{\overline{\phi}}
\begin{array}{c}
\Gamma \\ \downarrow f' \\
Z' \\
\end{array}
\]

and \( G(\phi) : G(\Delta) \to G(\Gamma) \) maps \( f \) to \( f' \). Note that \( \overline{\phi} \) and \( f \) determine \( f' \) and \( p \).

We usually only consider subcategories \( C \subset \text{Der}^n \) which are \textit{replete} in that for any object \( \Delta \) of \( C \) and iso \( \phi : \Delta \to \Gamma \) in \( \text{Der}^n \), then \( \Gamma \) and \( \phi \) are in \( C \). Given a rule \( G : C \to \text{Set} \) on \( C \subset \text{Der}^n \) and a pair \( (\Delta, \Gamma) \) of objects of \( C \), there may be \( \text{Der}^n \) morphisms \( \phi : \Delta \to \Gamma \) not in \( C \) which ‘don’t change \( G \)', in the sense that given any \( h : \Delta \to Z \in G(\Delta) \), the pushout of \( h \) along \( \overline{\phi} \) is already in \( G(\Gamma) \). We can construct the maximal such ‘condition category' by forming \( E \) consisting of the same objects as \( C \), with \( E(\Delta, \Gamma) = \{ \phi \in \text{Der}^n(\Delta, \Gamma) \mid (\forall h \in G(\Delta)) : \text{the pushout of } h \text{ along } \overline{\phi} \text{ is in } G(\Gamma) \} \). \( G \) then naturally extends to a rule \( G' : E \to \text{Set} \), where \( G'(\Delta) = G(\Delta) \). We call \( E \) the \textit{maximal condition category} for \( G \), and just denote the extension to \( E \) as \( G \) when it is unambiguous.

Given an element \( r \in G(M) \) of \( G : C \to \text{Set} \), there is an associated natural transformation \( y_r : yM \to G \). For each object \( \Delta \) of \( C \), the coordinate of the natural transformation is the function \( yM(\Delta) = C(M, \Delta) \to G(\Delta) \) sending \( \phi \in C(M, \Delta) \) to \( G(\phi)(r) \in G(\Delta) \). This is the natural transformation associated to the operation under the Yoneda lemma.

**Definition 20.** Let \( C \subset \text{Der}^n \) and let \( G : C \to \text{Set} \) be a rule. Take any \( M \in C \) and operation \( r : \overline{M} \to Z \in G(M) \). We say that \( r \) and \( C \) \textit{generate} \( G \) if the associated natural transformation \( y_r : yM \to G \) is epimorphic; that is, if each coordinate of the natural transformation is a surjective set function. If \( r_i \in G(M_i) \) is a set of \( G \)-operations on a family of \( C \)-objects, we say that the family \textit{generates} \( G \) if the sum \( y_{r_1} + \ldots + y_{r_n} : yM_1 + \ldots + yM_n \to G \) is epimorphic.

This just says that \( G \) is generated by \( r \) and \( M \) if every operation on \( \Delta \) arises as the pushout of \( r \) along \( \overline{\phi} \) for some \( C \)-morphism \( \phi : M \to \Delta \). \( G \) is generated by a family if every operation on \( \Delta \) arises as the pushout of \( r_i \) along \( \overline{\phi} \) for some \( C \)-morphism \( \phi : M_i \to \Delta \) for some \( M_i, r_i \) in the family. Clearly, if a set of operations generates an operation \( G : C \to \text{Set} \), then it generates the extension of \( G \) to its maximal condition category \( E \).

**Claim 27.** A rule \( G \) generated by \( f \) is completely determined by \( f \). That is, if \( G, H \) are any two \( C \)-rules both generated by \( f \), then \( G = H \).

We give an example. Let \( C \) be the category of pairs \( (\Delta, \Gamma) \) where \( \Delta \) and \( \Gamma \) have roots \( r \) and \( s \), together with pairs of morphisms \((\phi, \psi) : (\Delta, \Gamma) \to (\Delta', \Gamma') \) which take the root of \( \Delta \) to the root of \( \Delta' \) and the root of \( \Gamma \) to the root of \( \Gamma' \). Take the \( C \) object \((\{a\}, \{b\})\) and operation \( r : \{a\} + \{b\} \to \{a < b\} \). Then the \( C \)-operation generated by \( r \) and \((\{a\}, \{b\})\) is the rule \( G : C \to \text{Set} \) sending a pair \((\Delta, \Gamma)\) to the singleton set \( \{h : \Delta + \Gamma \to Z\} \) where \( Z \) is \( \top_r + \top_s \) with the relations \( !_r(r) \leq x \) added for \( x \in \top_s \).
The case of rules on $\text{FP}os$ generated by a finite set of operations is very similar to the single pushout construction (SPO) of [Ehrig et al., 1997]. We have generalized their construction in many ways: we allow maps to take in tuples of objects, and we allow arbitrary assignments of rules to tuples in principle if $G$ does not have to be generated by a finite set of operations. However, by restricting to operations generated by a finite family of operations, rules effectively reduce to pushouts of basic operations along specified ‘contexts’, much like the SPO. In other ways, we have restricted the operations. We only consider total functions, without deletion. We also restrict application of rules by manipulating condition categories $C$.

4.2. Extensions and grammars

Suppose that $\mathcal{D}(A)$ is a category of derivations such that the inclusion $A \to \mathcal{D}(A)$ has a left adjoint. Then a morphism from $\top(P,F)$ to a DSO $a$ is in correspondence with a morphism $(P,F) \to a$, and hence a family of compatible maps $\mu_p : F_p \to a$. There is an obvious way to ‘extend’ the derivation $(P,F)$ to include this new stage $a$: we add a point $*$ to $P$ and order relations $* \leq p$ for all $p \in P$. The morphism associated to each such order relation is just $\mu_p$. Given operations $h : \top(P_1,F_1) + \ldots + \top(P_n,F_n) \to a$ from the sum of yields of derivations to DSO $a$ (equivalently, a tuple of maps $(P_i,F_i) \to a$), we can recursively construct larger derivations by ‘extending’ them ‘along’ these operations, adding new stages and structural changes according to the operation.

The case of derivations is somewhat more subtle. Given an operation $h : \Delta \to Z$, we will want to add many points to $\Delta$, one for each 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 $\Delta$.

**Definition 21.** 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 $x \in \Delta$, $k(h(x)) \sqsubseteq k(x)$, that is, $kx$ projects to $k(hx)$ in $E$.

**Claim 28.** For any operation $h : \Delta \to Z$, there is a universal derivation $\text{ext}(h)$ together with a map $k : \Delta + Z \to \text{ext}(h)$ 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 $u : \text{ext}(h) \to E$ make the following diagram commute.

$$
\begin{array}{ccc}
\Delta + Z & \xrightarrow{k} & \text{ext}(h) \\
\downarrow \uparrow & & \downarrow \uparrow \\
E & \xrightarrow{i} & E \\
\text{ext}(h) & \xrightarrow{k'} & E \\
\end{array}
$$

For an operation $h : \Delta \to Z$, we will often denote the extension as $\Delta^h$. We now show many
ways in which this functor acts intuitively like an ‘extension of $\Delta$ along $h$’ on operations $h : \Delta \to Z$.

**Definition 22.** Define $\text{Der}/\text{FPos}$ to be the comma category defined by the following equation.

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

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

Concretely, an object of $\text{Der}/\text{FPos}$ 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.

$$
\begin{array}{ccc}
\Delta & \xrightarrow{f} & P \\
\phi \downarrow & & \downarrow h \\
\Gamma & \xrightarrow{g} & Q
\end{array}
$$

**Claim 29.** The assignment $\text{ext} : \text{Der}/\text{FPos} \to \text{FPos}$ sending $h : \Delta \to Z$ to $\Delta^h$ 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 $\phi^h$. This functor is determined uniquely up to natural iso.

**Claim 30.** 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) $\phi^h$ restricted to the subderivation $\Delta$ factors through $\Gamma \to \Gamma^g$ as $\phi$:
(b) \( \phi^h \) restricted to the subderivation \( Z \) factors through \( Z' \hookrightarrow \Gamma \) as \( h \):

\[
\begin{array}{ccc}
\Delta^f & \xrightarrow{\phi^h} & \Gamma^g \\
\uparrow & & \uparrow \\
Z & \xrightarrow{h} & Z'
\end{array}
\]

(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 \( \top_{\Gamma^f} \approx Z' \).

We can now construct a grammar. A grammar \( G \) is a (typically finite) set of rules

\[
\text{Rules} = \{ G_i : \mathcal{C}_i \to \text{Set}, \mathcal{C}_i \subset \text{Der}^n \}_{i \in I}
\]

together with a (typically finite) set of base items \( \text{LEX} \) of fposets.\(^9\) We describe the language derived by \( G \) as the class of derivations \( \mathcal{L}_G \):

**Definition 23.** If \( G \) is a grammar, then we define a language \( \mathcal{L}_G \) recursively as follows.

1. If \( X \in \text{LEX} \), then \( X \in \mathcal{L}_G \)

2. Let \( G : \mathcal{C} \to \text{Set} \) be a rule on \( \mathcal{C} \subset \text{Der}^n \) in \( \text{RULES} \). If \( (\Delta_1, \ldots, \Delta_n) \in \mathcal{C} \), \( h : \Delta_1 + \ldots + \Delta_n \to Z \in G(\Delta_1, \ldots, \Delta_n) \), and \( \Delta_i \in \mathcal{L}_G \), then \( (\Delta_1 + \ldots + \Delta_n)^h \in \mathcal{L}_G \)

We consider \( \mathcal{L}_G \) a category by taking it to be a full subcategory of \( \text{Der} \). That is, the \( \mathcal{L}_G \)-maps between a pair of objects \( \Delta \) and \( \Gamma \) is just the set of \( \text{Der} \)-maps between them.

However, such a grammar is usually too ‘weak’ for linguistic purposes, as it can only talk about dependencies, and not things like feature or label ‘type’ (N, V, T, C, etc.), feature ‘activity’ (feature \( f \) needs to be checked or has already been checked), and does not order the features (feature \( f \) must be checked before feature \( g \)). Before describing how to add these data, we make some notes about particularly ‘nice’ operations and how they lead to ‘nice’ derivations.

**Claim 31.** Let \( h : \Delta \to Z \) be an operation, and suppose that \( Z \) is rooted.

1. If \( \Delta \) is separated, then \( \Delta^h \) is separated

2. If \( \Delta \) is the sum of trees of terms (i.e. a forest of terms), then \( \Delta^h \) is a tree of terms

For this reason, we usually only want to consider \( G \)'s where each lexical item is rooted, and each rule produces only operations to rooted spaces \( Z \). When this is the case, the separation of each derivable \( \Delta \) into points close to some term corresponds to points coming from the same step. If we wish to use more general objects and rules, this partition must be extrinsically specified. Similar results to those in §3.1 hold for extrinsic specification of stephood.

\[^9\]However, not much changes if we allow more general base derivations.
4.3. Example grammar: Boston, et al. 2010

We give an example of how to add data to derivations which leaves much of the general theory intact. We demonstrate the main properties of ‘nice’ additions of structure to DSOs by sketching a model based on [Boston et al., 2010] (BHK), from which we can generalize. We could reconstruct their assignments of tuples of expressions to expressions as morphisms, much as we did for [Stabler, 1996] nearly verbatim. However, we would like to put features in our dependency trees explicitly, in that features in $\gamma_i$ should correspond to nodes in our DSOs.

We then construct a more general category of expressions than those in Boston, et al. Define an expression to be any fposet $s$ whose ‘dominance/dependency’ order is written $\preceq$, together with (1) a partition $S = \{s_1, \ldots, s_n\}$ and a ‘chain’ preordering $\rightarrow$ on $S$; (2) for each $s_i$ a subset $\pi_i \subset s_i$ of pronounceable nodes and a ‘precedence’ preordering $\preceq_i$ on $\pi_i$; (3) a unary predicate $\delta$ ‘derived’ on the set $S$; and (4) a subset $\gamma_i \subset s_i$ of ‘active’ features on each partition, together with a preordering $\ll_i$ on each set $\gamma_i$ representing ‘checking order’. Usually, $\rightarrow$ 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 $\sigma$, a morphism $f : \sigma \to \tau$ between derivations should be a function $f : s \to t$ between fposets, preserving $\preceq$ such that (1) if $x, y \in s$ are in the same partition $s_i$, then $f(x), f(y)$ are in the same partition $t_j$ such that $\hat{f}$ preserves $\rightarrow$, where $\hat{f} : \{s_1, \ldots, s_n\} \to \{t_1, \ldots, t_m\}$ is the map taking $s_i$ to $t_j$ if there exists $x \in s_i$ such that $f(x) \in t_j$; (2) if $x \in s_i$ is such that $x \not\in \pi_i$, then $f(x) \not\in \pi_j' \subset \hat{f}(s_i) = t_j$, and if $x \preceq_i y$ in $\pi_i$ and $f(x), f(y) \in \pi_j' \subset \hat{f}(s_i) = t_j$ are still pronounceable, then $f(x) \preceq_j' f(y)$; (3) if $\delta(s_i) = \top$ is ‘derived’, then $\delta(\hat{f}(s_i)) = \top$ is ‘derived’; (4) if $x \in s_i$ is such that $x \not\in \gamma_i$, then $f(x) \not\in \gamma_j' \subset \hat{f}(s_i) = t_j$, and if $x \ll_i y$ in $\gamma_i$ and $f(x), f(y) \in \gamma_j' \subset \hat{f}(s_i) = t_j$ are still active, then $f(x) \ll_j' f(y)$.

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 $\sigma$ to $s$ 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 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 $\sigma$ with predicates $\sigma_N, \sigma_V$, 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 $\sigma_X(a) = \top$ for some type $X \in \mathbf{B}$, then $\tau_X(f(a)) = \top$. Call this category $\mathbf{A}^\mathbf{B}$ or simply $\mathbf{A}$ when $\mathbf{B}$ is obvious. It is again
representably concrete by the same object \( \bullet \) with \( \bullet_X(*) = \bot \) for all \( X \in B \).

We can represent the syntactic objects of [Boston et al., 2010] with objects of \( A \), and we can represent their structural changes as morphisms of \( A \). We let \( B \) have at least types \( \text{base} \) (selectee), \( = \) (selector), \( + \) (licensor), and \( - \) (licensee), together with the base feature types \( F = \{ V, N, T, \text{wh}, \ldots \} \). 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 \( f_0, \ldots, f_n \). To each lexical item \( s :: f_1 \ldots f_n \), we associate the \( A \) object \( \sigma \) depicted in the table below.

<table>
<thead>
<tr>
<th>pset</th>
<th>( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( f_1 ) ... ( f_n )</td>
</tr>
<tr>
<td>partition</td>
<td>( { S } )</td>
</tr>
<tr>
<td>(chains)</td>
<td>( S = { s, f_1, \ldots, f_n } )</td>
</tr>
<tr>
<td></td>
<td>( \delta(S) = \bot ) (( S ) is underived)</td>
</tr>
<tr>
<td>( \pi ) sets</td>
<td>( \pi_S = { s } )</td>
</tr>
<tr>
<td>( \gamma ) sets</td>
<td>( \gamma_S = f_1 \ll \ldots \ll f_n )</td>
</tr>
</tbody>
</table>

We implicitly assume that each feature \( f_i \) is also ‘true’ for certain \( B \)-predicates. For example, if a feature \( f_i \) is \( =n \) or ‘select noun’ type, then \( \sigma_n(f_i) = \top \) and \( \sigma_(f_i) = \top \), meaning that \( f_i \) is a noun (\( n \)) feature, and a selector (\( = \)) feature.

A general expression in BHK is essentially a tree \( t \) with a partition \( \{ t_1, \ldots, t_n \} \) 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 \text{merge} \( 1 \) 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 \( e = (t_1, \preceq_1) \cdot g_0, \alpha_1, \ldots, \alpha_k \). BHK say that \text{merge} \( 1 \) 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 \text{merge} \( 1 \) 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 relations, as well as a relation from \( s \) to the root of \( e \). The first chain is the partition block \( \{ s \} \cup t_1 \) of derived type, with string of features \( f_1, \ldots, f_n \).

---

\( ^{10} \)This is not quite true: while BHK require that the \( t_i \) union to \( t \) and are pairwise disjoint, they leave open the possibility that a particular segment \( t_i \) is empty. However, for us, to be empty, you must not only be ‘phonologically empty’, but also have no syntactic features, since we 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.

\( ^{11} \)Here, \( \alpha_i \) 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.
This partition has precedence order $s \preceq x$ for all $x \in t_1$, as well as any precedence relations from $t_1$. 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 $(\sigma, \tau)$. If $(X, \prec)$ is any preorder and $x \in X$ is a unique least element in that $x \prec y$ for all $y \in X$ and and if $z$ also has this property, then $x = z$, we call it a root and write $x \prec X$. We write minimal assumptions about the DSOs we want to apply $\text{merge1}$ to in the table below.

<table>
<thead>
<tr>
<th>fposet</th>
<th>$\sigma$ has a (unique) root $s \leq \sigma$</th>
<th>$\tau$ has a root $t \leq \tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition (chains)</td>
<td>$\text{Ch}_\sigma = {S}$, $\delta S = \bot$</td>
<td>$\text{Ch}<em>\tau$ has root $T \rightarrow \text{Ch}</em>\tau$</td>
</tr>
<tr>
<td>$\pi$ sets</td>
<td>$s \in \pi_S$</td>
<td>$\gamma_T$ has root $g \ll \gamma_T$</td>
</tr>
<tr>
<td>$\gamma$ sets</td>
<td>$\gamma_S$ has root $f \ll \gamma_S$</td>
<td></td>
</tr>
</tbody>
</table>

We additionally assume that $f$ is of type $\text{=}$ and $g$ of type $\text{base}$, and that both have the same syntactic type, say $n$. 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 $h : \sigma \rightarrow E$ and $k : \tau \rightarrow E$ meet the $\text{merge1 condition}$ if its images meet certain conditions. For an element $x$ of $\sigma$ or $\tau$, we write $\pi$ for its image under $h$ or $k$ in $E$. Similarly, for any partition $P$ of $\sigma$ or $\tau$, we write $\overline{P}$ for the partition containing $\pi$ for any $x \in P$. $h$ and $k$ meet the $\text{merge1 condition}$ if (1) $\pi \leq \overline{t}$; (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 ($\delta(ST) = \bot$). There is a universal such expression $E$ which meets the $\text{merge1 condition}$ determined up to isomorphism, which we write $\text{merge1}(\sigma, \tau)$. By this, we mean that there is a pair of morphisms $m_\sigma : \sigma \rightarrow \text{merge1}(\sigma, \tau)$ and $m_\tau : \tau \rightarrow \text{merge1}(\sigma, \tau)$ such that for any $h$ and $k$ meeting the $\text{merge1 condition}$, there is a unique $u : \text{merge1}(\sigma, \tau) \rightarrow E$ 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, $\pi$, and $\gamma$ 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.

In simple cases, we can generate instances of $\text{merge1}$ and the structural change maps as pushouts from a finite set of basic structural changes. We make the following extra assumptions, still assuming that $f$ and $g$ have matching syntactic type, and $f$ is a $\text{=}$ feature and $g$ a $\text{base}$ feature.
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 $\leq$, $\rightarrow$, and $\ll$ roots of $\sigma$ and $\tau$ to the $\leq$, $\rightarrow$, and $\ll$ roots of $\sigma'$ and $\tau'$. For example, if $s'$ and $t'$ are the roots of $\sigma'$ and $\tau'$, then for any $k : \sigma' + \tau' \to E$, if there exists a morphism $v : \text{MERGE1}(\sigma, \tau) \to E$ such that for all $x \in \sigma$, $k(u_1(x)) = v(m(x))$ and for all $x \in \tau$, $k(u_2(x)) = v(m(x))$, 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 $(\sigma, \tau)$, the merge1 operations on it are generated by a single operation, 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 the generating operations.

We denote these expressions as $A$, $B$, and $C$, respectively. 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 $xy$. We take the pushout of $r$ along the map $u_1 + u_2 : A + B \to \sigma + \tau$ which takes the root $a$ to the root of $\sigma$, the element $x$ to $f$, the root $b$ to the root of $\tau$, 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. Assuming all elements in the left and righthand sides of the rule $r$ are untyped, the pushout does not change the B-typing of any element. An identical variant lacking the node $k$ handles the case when $T$ has no pronounceable elements, since no precedence order needs to be added. These two base generating rules handle all of the original case under the assumption that $\pi_S = \{s\}$, which is general enough to handle all of the BHK-like DSOs.

While writing out the structural changes as pushouts is a bit cumbersome, it is straightforward, though some special care should be taken when adding precedence ordering $\preceq$.
or chain ordering $→$. We must make sure we have ‘generic representatives’ for $≤$-initial and final pronounced elements and $→$-initial and final chains so that we can specify when ordering is added between them.\(^{12}\) For example, $k$ above was a ‘dummy’ element which represents the $≤$ initial node in the first chain of some expression $e$, and adding a relation $x ≤ k$ will then add $x ≤ y$ for all $\pi$ 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 $A$ comparable to those in [Boston et al., 2010].

We finally must generalize the rule and extension constructions to $\mathcal{A}\text{Der}$ (§3.2). We first note many good properties of $A$. The functor $U : A \to \mathcal{FPos}$ sending $σ$ to its underlying fposet $\langle |σ|, ≤_{σ} \rangle$ is faithful, so that $A$ is concrete over $\mathcal{FPos}$. We have an inclusion $i^{A} : A \to A\text{Der}$ sending $σ$ to the derivation with underlying fposet $\langle |σ|, ≤_{σ} \rangle$ with the isomorphism $U_x : U(σ_x)$ at each point $x ∈ σ$, where $σ_x$ is the $A$-substructure on $U_x$ (e.g. two nodes are in the same partition of $σ_x$ iff they are in the same partition in $σ$, have $≤$ relation in $σ_x$ iff in $σ$, etc.). We can then form the comma category over $1_{A\text{Der}}$ and $i^{A}$ to construct $A\text{Der}/A$. $i^{A}$ has a left adjoint $\top^{A}$, which is unique up to iso, and the counit $\top^{A}i^{A} \to 1_{A}$ is an iso (the yield of a DSO $σ$ is $σ$). For any operation $h : Δ \to i^{A}σ$ in $A\text{Der}/A$, there is a universal $A\text{Der}$ morphism $k : Δ + i^{A}σ \to \text{ext}^{A}(h)$ taking $h$-images to projection, which determines a functor $\text{ext}^{A} : A\text{Der}/A \to A\text{Der}$ up to iso. (Ambigously) writing $U$ for all of the forgetful functors $U : A \to \mathcal{FPos}$, $U : A\text{Der} \to \text{Der}$, and $U : A\text{Der}/A \to \text{Der}/\mathcal{FPos}$, we have natural isos $U \circ \text{ext}^{A} \approx \text{ext} \circ U$, $U \circ i^{A} \approx U \circ i$, and $U \circ \top^{A} \approx \top \circ U$. Finally, $A \to \mathcal{FPos} \to \text{Set}$ is representable (say by $\bullet$) such that $U(\bullet) \cong \mathbf{1}$, which implies that $U : A\text{Der} \to \text{Der}$ is representable by $i^{A}\bullet$. We can axiomatize triples $(A, U, i^{A})$ with these properties, in which case we call $A$ extendable. These requirements guarantee that we can emulate all of the main constructions for recursively building derivations concretely.

We seek a straightforward ‘lifting’ of operations on $A$ to operations on $A\text{Der}$. The left adjoint property $\top^{A} \vdash i^{A}$ comes with a unit $η_{Δ} : Δ \to i^{A}\top^{A}(Δ)$ for each $Δ$ in $A\text{Der}$ (which we simply write $Δ \to \top^{A}_A$). This map has the property that any map $h : Δ \to i^{A}Z$ factors uniquely through $Δ \to \top^{A}_A \to Z$. Additionally, as a left adjoint, $\top^{A}$ ‘preserves coproducts’ in the sense that $\top^{A}_{Δ_1 + \ldots + Δ_n} \approx \top^{A}_{Δ_1} + \ldots + \top^{A}_{Δ_n}$. Then, for any $n$-ary operation $\top^{A}_{Δ_1 + \ldots + Δ_n} \to Z$, we get an associated $A\text{Der}$ morphism $Δ_1 + \ldots + Δ_n \to Z$. Additionally, given an operation $h : Δ \to Z$ and morphism $ϕ : Δ \to Γ$, the universal $A$-object $Y$ with morphisms $f : Z \to Y$ and $g : Γ \to Y$ such that $fh = gϕ$ can be computed by applying $\top^{A}$ to the whole diagram and taking the pushout in $A$. In this way, we can ‘freely’ lift a rule $G : C \to \text{Set}$, $C ⊂ A^n$ to a rule $G' : D \to \text{Set}$, $D ⊂ A\text{Der}^n$, by declaring $G'(Δ) = \{ Δ \to \top^{A}_A \to Z \mid \top^{A}_A \to Z \in G(\top^{A}_A) \}$, and taking the maximal condition category. There is a $G'$ operation on a tuple $Δ$ if and only if there is a $G$ operation on the tuple of their yields using this method.

\(^{12}\)None of the operations in [Boston et al., 2010] add ordering relations between active features, so we do not need to worry about similar constructions with $≪$. 

33
Definition 24. If $G$ is a grammar of $\textsf{ADer}$ rules with a lexicon of $A$ objects, then we define a language $L_G$ recursively as follows.

1. If $X \in \text{Lex}$, then $X \in L_G$

2. Let $G : C \to \text{Set}$ be a rule on $C \subset \text{ADer}^n$ in $\text{Rules}$. If $(\Delta_1, \ldots, \Delta_n) \in C$, $h : \Delta_1 + \ldots + \Delta_n \to Z \in G(\Delta_1, \ldots, \Delta_n)$, and $\Delta_i \in L_G$, then $(\Delta_1 + \ldots + \Delta_n)^h \in L_G$, where $(\Delta_1 + \ldots + \Delta_n)^h = \text{ext}^A(h)$.

It is then straightforward to translate other formal grammars, such as BHK, into $\textsf{ADer}$ derivations whenever $A$ admits the requisite constructions. In particular, for each BHK grammar, we can construct a grammar $G$ with an assignment of BHK lexical items to $A$ objects and BHK operations to $\textsf{ADer}$ rules, bijectively. We can assign derivations of BHK expressions to $\textsf{ADer}$ objects such that $G$ derives $\Sigma$ from $\Delta$ and $\Gamma$ using the operation $G$ iff there are derivations of expressions $e$, $f$, and $g$ associated to them where BHK can derive $e$ from $f$ and $g$. The difference is now our objects are structured sets, and we can apply all of the techniques from the preceding sections to them. For example, we may describe a strong extensional equivalence between languages as in Def. 6, just replacing $\mathcal{D}(A)$ with $\textsf{ADer}$ for extendable $(A, U, i^A)$.

5. Differences from ‘algebraic’ approach

The defining property of $\textsf{Der}$ is that it emphasizes descriptions of derivations as ‘structured sets’. We compare our notions of isos and substructures to those in other theories.

We start by looking at the relevant notions for DSOs. Stabler & Keenan [Stabler and Keenan, 2003, Keenan and Stabler, 2010] view a grammar as a set of expressions, together with partial operations. Their method is general, accounting for many variants of formal MGs. We look at their methods, and show by example that their notions of isomorphism and constituent are about the combinatorics of the grammar, while our notions are about the structure of the derivations and DSOs. That is, the combinatorial method formalizes properties related to ‘when operations are defined’, while the categorical approach is about ‘what operations do to structure’.

Fix a grammar $(G, f_1, \ldots, f_n)$, where $G$ is a set of expressions, and $f_i : G^i \to G$ are partial functions taking finite tuples $(g_1, \ldots, g_k) \in G^k$ to elements of $G$. Two objects $x, y \in G$ are isomorphic in their sense if there is an automorphism $\pi$ of the $G$ 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 $\pi(f_i(g_1, \ldots, g_k))$. It should be clear from the definition that this notion is not with respect to the structure of the SOs (i.e. is not an invertible structure-preserving map), but about what other elements an expression $x$ can combine with for each $f_i$. 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 \( A \), where two objects are isomorphic
iff they have the same dependency structure, feature typing, etc.

This difference extends to ‘higher order’ comparisons like equivalence of languages and
grammars. The automorphisms of a grammar as defined do not allow us to interchange any
operations, showing that they cannot detect information about similar structural changes
induced by those operations. The ‘structure of a grammar’ in this sense is about how many
operations there are, and for what tuples they are defined, not the structural changes they
induce. For example, consider the grammar \( G \). Let \( T \) be the set of binary unlabeled
trees, where each pair of sisters has a precedence ordering \( \preceq \) or \( \succeq \), and let
\( G = T_1 + T_2 + T_3 \), all 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 \( t, s \in T_1 \). Let \( f_2 \) be
an identical function, but defined only on \( T_2 \). Finally, let \( f_3 \) be the same operation on
\( T_3 \), but \( t \succeq s \) in \( f_3(t, s) \). Without allowing permutation of the operations, there is no
formal way to say that \( f_1 \) and \( f_2 \) induce the same structural change. There is a reasonable
weakening of automorphism: let an automorphism \( (\pi, \kappa) \) of \( (G, f_1, \ldots, f_n) \) be a pair where
\( \pi \) is a permutation of \( G \) and \( \kappa \) 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 \( \pi \) interchanging the versions of each \( s \) in \( T_1 \) and \( T_2 \) and \( \kappa \)
interchanging \( f_1 \) and \( f_2 \) is one such automorphism, but so is \( \pi \) interchanging \( s \) and \( s' \) in \( T_1 \)
and \( T_3 \), where \( s' \) is just \( s \) with reversed \( \preceq \) ordering, while \( \kappa \) 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 iso between structural changes requires that it ‘do isomorphic
things’ to the structures in question.

Similarly, take any a partial function \( f : G^+ \to G \), and write \( \text{dom}(f) \) for the subset of \( G^+ \)
where it is defined. We can always take some partition of \( G^+ \) into \( \{S_1, \ldots, S_n\} \) and define
partial functions \( f_i : G^+ \to G \) where \( f_i \) is defined only on \( S_i \cap \text{dom}(f) \), and where defined
takes exactly the same values as \( f \). Intuitively, a grammar with \( f \) and a grammar with
the collection \( f_i \) instead are ‘similar’ in that their rules can induce the same structural
changes for the same objects. More simply, we might hope for some automorphism of
\( (G, f, f_1, \ldots, f_n) \) which interchanges certain algebraic expressions in \( f \) and \( f_i \), but that
would require another weakening of ‘similarity’ of grammars. Such a ‘division of a partial
function’ is common in MGs, where \text{MERGE} breaks down into different cases depending on
the inputs. This shows that whether we treat \text{MERGE} as separate partial functions or a
single one leads to nonisomorphic grammars.

Finally, equivalences of languages defined algebraically will be ill equipped to talk about
structural relations in the same manner. [Stabler and Keenan, 2003] note ‘we would like to
say that adding a single name to a language does not (significantly) change the structure
of the language.’ They formalize this as follows:

When grammars \( G, G' \) are identical except that \( \text{Lex}_G \subset \text{Lex}_{G'} \), we say that

35
$G'$ is a lexical extension of $G$. A lexical extension of $G'$ of $G$ is \textit{free} iff for ever $s \in \text{Lex}_{G'} - \text{Lex}_G$, there is a $t \in \text{Lex}_G$ such that $s \simeq_{G'} t$.

where $\simeq_{G'}$ means that there is an automorphism of $G'$ taking $s$ to $t$. We have already described why $\simeq$ is not always well-behaved with respect to the internal structure of a lexical item or DSO, while our notion of ‘equivalence of languages’ implies isomorphic $A$ structures of DSOs. Further, consider the case when there is a common language $\mathcal{N}$, and $\mathcal{L}$ and $\mathcal{M}$ are two ‘free lexical extensions’, each adding a different \textit{number} of words, but each word (intuitively) isomorphic to one added in the other language (for example, speaker $\mathcal{L}$ adds 3 names, while speaker $\mathcal{M}$ adds 6). Neither of $\mathcal{L}$ nor $\mathcal{M}$ will be contained in the other, but they seem just as ‘equivalent’ as each of them is to $\mathcal{N}$. Our notion of equivalence of languages has the property that they are all equivalent, though they are not all in a lexical extension relation, as desired. Of course, a similar equivalence should (and, using our definition, does) hold even in circumstances when the languages may not share a common sublanguage, so long as they have isomorphic DSOs which undergo isomorphic structural changes.

There is no corresponding notion of substructure of a DSO, though [Stabler and Keenan, 2003] do give a ‘constituency’ relation between DSOs. $A$ is a constituent of $B$ in their sense if there is some sequence of operations taking in an $A$ and eventually producing a $B$. Since this quantifies over the whole grammar, items not occurring in particular derivations of $B$ may still be constituents of it; e.g., if $\lambda ::= yx$ and $\lambda :: y$ are items of the grammar, they will be constituents of $\lambda : x$, despite not occurring in the derivation in Fig. 4.

![Figure 4: Example of ambiguous constituency](image)

This again shows that these definitions are about the combinatorial relations between objects under operations, not about the structure of the objects involved. In contrast, a subderivation embedding must include the diagram of steps that have occurred in the subderivation into the superderivation.

However, since it is easily possible, we may want to use both structural and algebraic methods to compare grammars and describe grammatical operations. One reason is that an iso of $A\text{Der}$ is totally insensitive to the specific names of the rules used to build it: if two rules $G$ and $H$ happen to induce the same structural change for a given tuple, the two derivations in which they are applied will be isomorphic.


6. Conclusion

We proposed general models $\mathcal{D}(A)$ of derivations of $A$, combining the ‘derivation tree of DSOs’ of models like [Boston et al., 2010] with the ‘explicit structural change information’ of [Ehrig et al., 1997]. We then specialized to categories $\text{ADer}$ whose properties as constructs were particularly well-behaved. We demonstrated that the induced notions of subderivations and isomorphic derivations provided the ‘correct’ definitions, in that structural properties of DSOs and the structural changes between them remain invariant under iso. We then gave applications like describing rules in terms of the structural changes they induce, describing how to construct languages from grammars, and how to determine strong extensional equivalences between these languages.

As a general introduction, there are many theoretical and empirical applications of having all of this structure which were left out for space. More precise descriptions of grammatical relations as a ‘typology’ of dependencies introduced between DSOs over the course of some structural change can be given, and will be preserved under iso. Similarly, having given a general calculus for rules generated from basic operations, it becomes easy to test hypotheses about what structural changes are used by natural language. For example, it becomes easy to model dependencies like feature-sharing and feature geometry, and give rules which can manipulate these structures. We can also give a theory of how certain rules are related in terms of the structural changes they introduce - e.g. $\text{MERGE}$ might be a combination of $\text{ROOT-ATTACHMENT}$ plus $\text{SELECT}$, or in other cases $\text{ROOT-ATTACHMENT}$ plus $\text{LICENSE}$ or $\text{AGREE}$. Having structured DSOs also allows us to constrain applications of a rule based on the structure of the DSOs which may serve as arguments to it. Hopefully, such a robust toolkit can lead to more precise formulations of the constraints on natural language and understanding of the properties of the structures it generates.

References


A. Category Theory

We give definitions of basic constructions from category theory used throughout the paper.

Definition 25. A category is a class of objects \( \mathcal{C} \), together with a set \( \mathcal{C}(A, B) \) of morphisms for each pair of objects \((A, B)\), together with a composition function \( \circ : \mathcal{C}(A, B) \times \mathcal{C}(B, C) \to \mathcal{C}(A, C) \) for each triple \((A, B, C)\). 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:
   \[
   f \circ 1_A = f, \quad 1_A \circ g = g
   \]

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, 1994a] 1.2.1

Definition 26. ([Borceux, 1994a] 1.2.2) 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 \( f \in A(A, A'), g \in A(A', A'') \): \( F(g \circ f) = F(g) \circ F(f) \)
2. for every object \( A \in A \): \( F(1_A) = 1_{FA} \)

**Definition 27.** A functor \( F : C \to D \) is an **isomorphism of categories** if there is a functor \( G : D \to C \) such that \( G \circ F = 1_C \) and \( F \circ G = 1_D \).

**Definition 28.** A functor \( F : C \to D \) is **faithful** if every induced set-function \( C(C, C') \to D(FC, FC') \), taking \( CF \) to \( FC' \), is injective - i.e. if \( F(f) = F(g) \) iff \( f = g \).

**Definition 29.** A functor \( F : C \to D \) is **full** if every induced set-function \( C(C, C') \to D(FC, FC') \), taking \( CF \) to \( FC' \), is surjective, - i.e. if \( FC' \) is any morphism in \( D(FC, FC') \), then there is some morphism \( C \to C' \) in \( C(C, C') \) such that \( F(f) = h \).

**Definition 30.** A **subcategory** of a category \( C \) is a category \( D \) whose objects are objects of \( C \), and whose morphisms are morphisms of \( C \), such that if \( x \) is an object of \( D \), then \( 1_x : x \to x \) is a \( D \)-morphism.

A subcategory is written \( D \subset C \), and is associated to a functor \( i : D \Rightarrow C \).

This inclusion functor is always faithful.

If this inclusion functor is full, then we say that \( D \) is a **full subcategory** of \( C \).

**Definition 31.** Let \( U : A \to Set \) be a construct. An \( A \)-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 \( A \)-object \( C \), and 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 \( A \)-morphism.

([Adámek et al., 2004] 8.6)

**Definition 32.** ([Borceux, 1994a] 1.7.1) A morphism \( f : A \to B \) in a category \( C \) is called a **monomorphism** when, for every object \( C \in C \) and every pair of morphisms \( g, h : C \Rightarrow A \), the following property holds: \((f \circ g = f \circ h) \Rightarrow (g = h)\).

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. A partial function \( f : A \to B \) in \( PSet \) is a monomorphism iff it is an injective total function.
3. An order-preserving function \( f : P \to Q \) in \( Proset \) is a monomorphism iff it is injective on the underlying sets.
We can now generalize embeddings.

**Definition 33.** ([Adámek et al., 2004], 8.6) Let $U : A \to X$ be a faithful functor.

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

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

The above definition specializes to Def. 31 in the case $X = \text{Set}$.

An **epimorphism** is the dual of a monomorphism: $f : A \to B$ is an epimorphism in $C$ when, for every object $C \in C$ and every pair of morphisms $g, h : B \Rightarrow C$, the following property holds: $(g \circ f = h \circ f) \Rightarrow (g = h)$.

**Definition 34.** Let $(A, B)$ be a pair of objects of a category $C$. A triple $(C, k_1 : A \to C, k_2 : B \to C)$ of object and morphisms is a **coproduct** of $(A, B)$ in $C$ if for any triple $(D, s_1 : A \to D, s_2 : B \to D)$, there is a unique map $u : C \to D$ such that $s_1 = k_1 \circ u$ and $s_2 = k_2 \circ u$. See diagram. Often we just refer to $C$ as the coproduct or **sum** and write it as $A + B$, and morphisms $k_1$ and $k_2$ as the **coprojections**.

**Claim 32.** If $(C, k_1, k_2)$ and $(C', k_1', k_2')$ are coproducts of $(A, B)$, then there is a unique iso $C \cong C'$ commuting with the coprojections. That is, coproducts are unique up to unique iso.

**Definition 35.** Given functors $F, G : C \Rightarrow D$, a **natural transformation** $\eta : F \to G$ from $F$ to $G$ is a collection of $D$-morphisms $\eta_C : FC \to GC$, one for each object of $C$, which meet the condition: for any $C$-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 $(\beta \circ \eta)_C = \beta_C \circ \eta_C$.

**Definition 36.** Given categories and functors $F : C \Rightarrow D : G$, an **adjunction** from $F$ to $G$ is a pair of natural transformations $\eta : 1_C \Rightarrow GF$ and $\epsilon : FG \Rightarrow 1_D$ called the **unit** and **counit** meeting the two equalities: for every object $c$ of $C$, $1_{Fc} = \epsilon_{Fc} \circ F(\eta_c) : Fc \to FGFc \to Fc$, and for every object $d$ of $D$, $1_{Gd} = G(\epsilon_d) \circ \eta_{Gd} : Gd \to GFGd \to Gd$, summarized by the diagrams:
If there is an adjunction from $F$ to $G$, we write $F \dashv G$. 

**Claim 33.** Adjoint determine each other up to isomorphism. That is, if $F, F' \dashv G$, then there is a natural isomorphism $F \approx F'$, and if $F \dashv G, G'$, then there is a natural isomorphism $G \approx G'$.

**Definition 37.** For any category $A$ and any object $a$ of $A$, we define the functor $y(a) \equiv A(a, -) : A \to \text{Set}$. $y(a)$ returns for each object $b$ of $A$ the set $A(a, b)$ and for each morphism $f : b \to c$ the function $y(f) : y(a)(b) \to y(a)(c)$ taking $g : a \to b$ to $f \circ g : a \to c$. We say that a functor $F : A \to \text{Set}$ is **representable** if it is naturally isomorphic to $y(a)$ for some $a$ in $A$.

**Definition 38.** In any category, for any pair of arrows $f : A \to B$ and $g : A \to C$, an object $D$ together with maps $k : C \to D$ and $j : B \to D$ such that $jf = kg$ is a **pushout** if it is universal with respect to this property. This means that for any object $E$ and pair of maps $m : C \to E$ and $n : B \to E$ such that $nf = mg$, there is a unique map $u : D \to E$ such that $uk = m$ and $uj = n$. See picture below.

Given a morphism $f : A \to B$ and morphism $g : A \to C$, we call $k : C \to D$ the **pushout of $f$ along $g$**. This can be thought of as finding the ‘best’ translation of $f$ into $C$ in the context $g$. As with all universal constructions, $D$, $k$, and $j$ are all unique up to unique isomorphism. This means that if $D'$, $k'$ and $j'$ also give a pushout, then there is a unique isomorphism $i : D \cong D'$ such that $ik = k'$ and $ij = j'$.

**Claim 34.** The most fundamental property of pushouts is that they paste in any category.

The usual statement of the pushout lemma is that if the left square is a pushout square, the right square is a pushout square if and only if the outer rectangle is a pushout. That is, we could push $k$ out along $f$ to obtain $j$. We could then push $j$ out along $g$ to obtain $c$. On the other hand, we could start by composing $f$ and $g$, then push $k$ out along $gf$. The pushout lemma states that $c$ is also this pushout.

**Definition 39.** ([Mac Lane, 1971], II.6) Given categories and functors
the comma category $(T \downarrow S)$, also written $(T, S)$, has as objects all triples $\langle e, d, f \rangle$ with $d \in \text{Obj } D$, $e \in \text{Obj } E$ 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,

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.