STRING DIAGRAMS FOR TEXT

VINCENT WANG-MAŚCIANICA

ST. CATHERINE'S COLLEGE
THE UNIVERSITY OF OXFORD
DEPARTMENT OF COMPUTER SCIENCE

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
2023

Contents

(Acknowledgements will go in a margin note here.)

0

Internal wirings: what, why, and where from?

Chapter summary: Speakers *produce*, and listeners *parse* sentences. If we believe that semantics is compositional according to syntax, because communication is possible, these two ways of conceiving of grammar (e.g. string-rewrite systems and typelogical grammars, respectively) must be mathematically related: the semantics of a sentence ought to be "the same" in either grammar. Using string diagrams as semantics allows us to formalise "sameness" as "up to topological deformation", and internal wirings constitute a shared representation strategy on words between speaker and listener that witnesses this topological equivalence for any sentence that both grammars can produce. However, internal wirings are not functorially determined by syntax: the same word may have different internal wirings in a way that depends systematically on context. We can capture this systematicity as a span of functors, for which we develop a diagrammatic technique that works like functor boxes, along with the attendant mathematics.

0.1 How do we communicate using language?

Speakers and listeners understand one another. Obviously, natural language involves communication, which involves at minimum a speaker and a listener, or a producer and a parser. The fact that communication happens at all is an everyday miracle that any formal understanding of language must account for. The miracle remains so even if we cautiously hedge to exclude pragmatics and context and only encompass small and boring fragments of factual language. At minimum, we should be able to model a single conversational turn, where a speaker produces a sentence, the listener parses it, and both agree on the semantics. Here is a sequence of diagram equations that demonstrates mathematically how the miracle works for two toy grammars, for the sentence `Alice sees Bob quickly run to school`. On the left we have a grammatical structure obtained from a context-free grammar, and we have equations from a discrete monoidal fibration all the way to the right, where we obtain a pregroup representation of the same sentence. Going from right to left recovers the correspondence in the other direction.

Here are some naïve observations on the nature of speaking and listening. Let's suppose that a speaker, Charlie, wants to communicate a thought to Dennis. Charlie and Dennis cooperate to achieve the miracle; Charlie encodes his thoughts – a structure that isn't a one-dimensional string of symbols – into a one-dimensional string of symbols. And then Dennis does the reverse, turning a one-dimensional string of symbols into a thought-structure like that of Charlie's. It may still be that Charlie and Dennis have radically different internal conceptions of what `FLOWERS` or `GIVING` or `BEETLES IN BOXES` are, but that is alright: we only care that the *relational structure* of the thought-representations in each person's head are the same, not their specific representations.

The nature of their challenge can be summarised as an asymmetry of information. The speaker knows the structure of a thought and has to supply information or computation in the form of choices to turn that thought into text. The listener knows only the text, and must supply information or computation to deduce the thought behind it. By this perspective, language is a shared and cooperative strategy to solve this (de/en)coding interaction.

Speakers choose. The speaker Charlie must supply decisions about phrasing a thought in the process of speaking it. At some point at the beginning of an utterance, Charlie has a thought but has not yet decided how to say it. Finding a particular phrasing requires choices to be made, because there are many ways to express even simple relational thoughts. For example, the relational content of our running example might be expressed in at least two ways (glossing over determiners):

`Alice likes flowers that Bob gives Claire.`

`Bob gives Claire flowers.  Alice likes (those) flowers.`

Whether those decisions are made by committee or coinflips, they represent information that must be supplied to Charlie in the process of producing language. For this reason, we consider context-free-grammars (and more generally, other string-rewrite systems) to be *grammars of the speaker*, or *productive grammars*. The start symbol S is incrementally expanded and determined by rule-applications that are selected by the speaker. The important aspect here is that the speaker has an initial state of information S that requires more information as input in order to arrive at the final sentence. Note that the concept of productive grammars are not exhausted by string-rewrite systems, merely that string-rewrite systems are a prototype that illustrate the concept well.

LISTENERS DEDUCE. The listener Dennis must supply decisions about which words are grammatically related, and how. Like right-of-way in driving, sometimes these decisions are settled by convention, for example, subject-verb-object order in English. Sometimes sophisticated decisions need to be made that override or are orthogonal to conventions, as will be illustrated in the closing discussions and limitations section of this chapter. Since Dennis has to supply information in the form of choices in the process of converting text into meaning, we consider *parsing grammars* – such as all typelogical grammars, including pregroups and CCGs – to be *grammars of the listener*.

THE SPEAKER'S CHOICES AND THE LISTENER'S DEDUCTIONS MUST BE RELATED. The way the speaker decomposes the thought into words in text in the speaker's grammar must allow the listener to reconstruct the thought in the listener's grammar. Even in simple cases where both parties are aiming for unambiguous communication, the listener still must make choices. This is best illustrated by introducing two toy grammars – we pick a context-free grammar for the speaker and a pregroup grammar for the listener, because they are simple, planar, and known to be weakly equivalent.

We assume Charlie and Dennis speak the same language, so both know how words in their language correspond to putative building blocks of thoughts, and how the order of words in sentences and special grammatical words affect the (de-/re-)construction procedures. Now we have to explain how it is that the two can do this for infinitely many thoughts, and new thoughts never encountered before. Using string diagrams, this is surprisingly easy, because string diagrams are algebraic expressions that are invariant under certain topological manipulations that make it easy to convert between different shapes of language.

Example 0.1.1 (`Alice likes flowers that Bob gives Claire.`)**.** Let's say Charlie is using a context-free grammar to produce sentences, and Dennis a pregroup grammar.

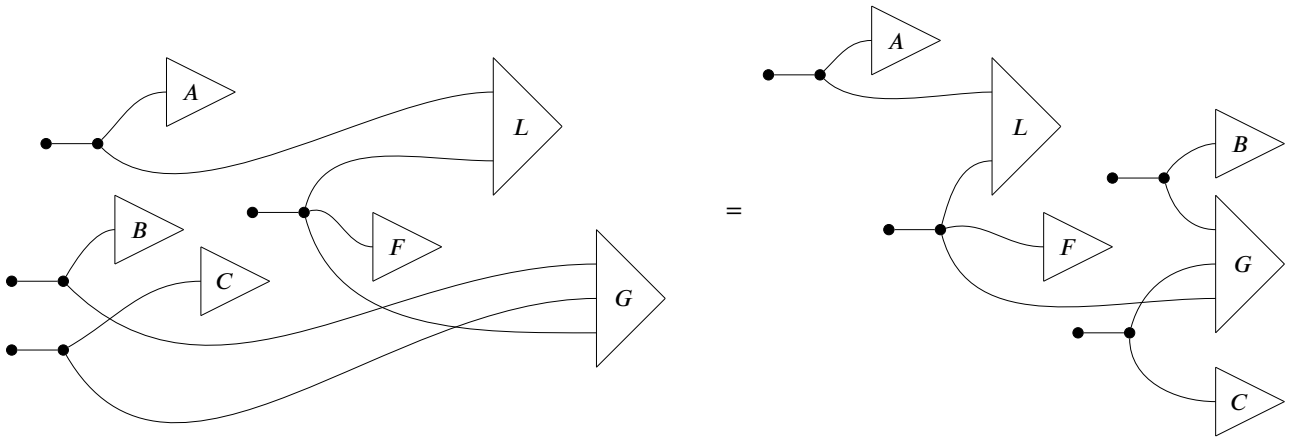


Figure 1: Charlie and Dennis agree on the conceptual organisation entities and relations up to the words for those entities and relations. Just as a running example that does not affect the point, let's say we can gloss a thought in first order logic as $\exists a \exists b \exists c \exists f \ : \ A(a) \wedge B(b) \wedge C(c) \wedge F(f) \wedge L(a,f) \wedge G(b,c,f)$. In diagrammatic first order logic [], this is equivalently presented as the following diagrams (and any other diagram that agrees up to connectivity.) For example, Charlie could ask Dennis comprehension questions such as `WHO GAVE WHAT? TO WHO?`, and if Dennis can always correctly answer – e.g. `BOB GAVE FLOWERS. TO CLAIRE.` – then both Charlie and Dennis agree on the relational structure of the communicated thought to the extent permitted by language.

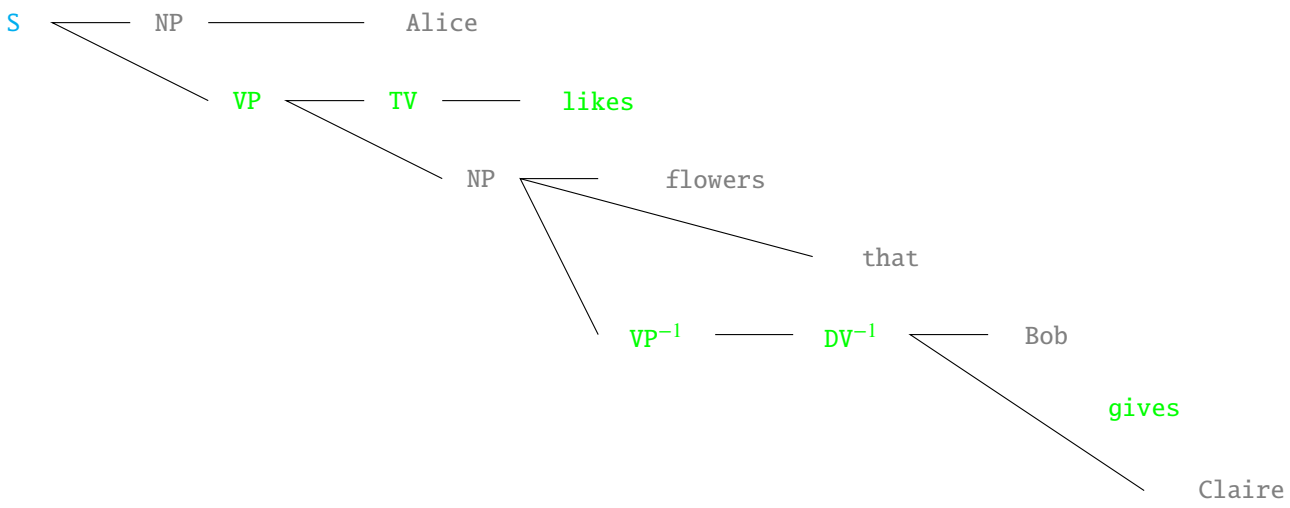


Figure 2: The rule of the game is that Charlie and Dennis can agree on a string-diagrammatic encoding strategy before having to communicate with each other. Here is one such strategy. Charlie might generate the example sentence as depicted.

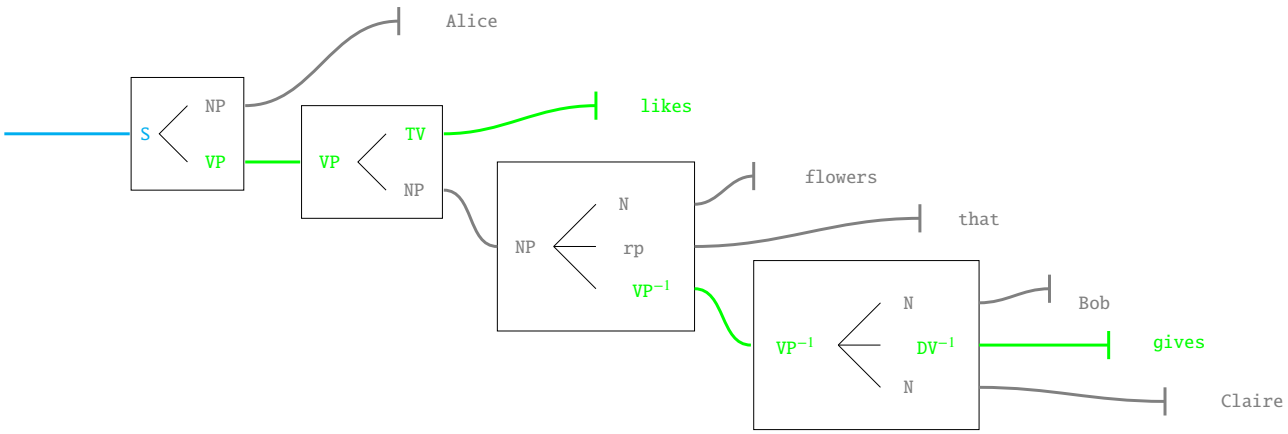


Figure 3: Mathematically, it makes no difference if we take the Poincaré dual of the tree, so that zero-dimensional nodes become one-dimensional wires, and branchings become zero-dimensional points linking wires – but we can just as well depict those points as boxes to label them more clearly.

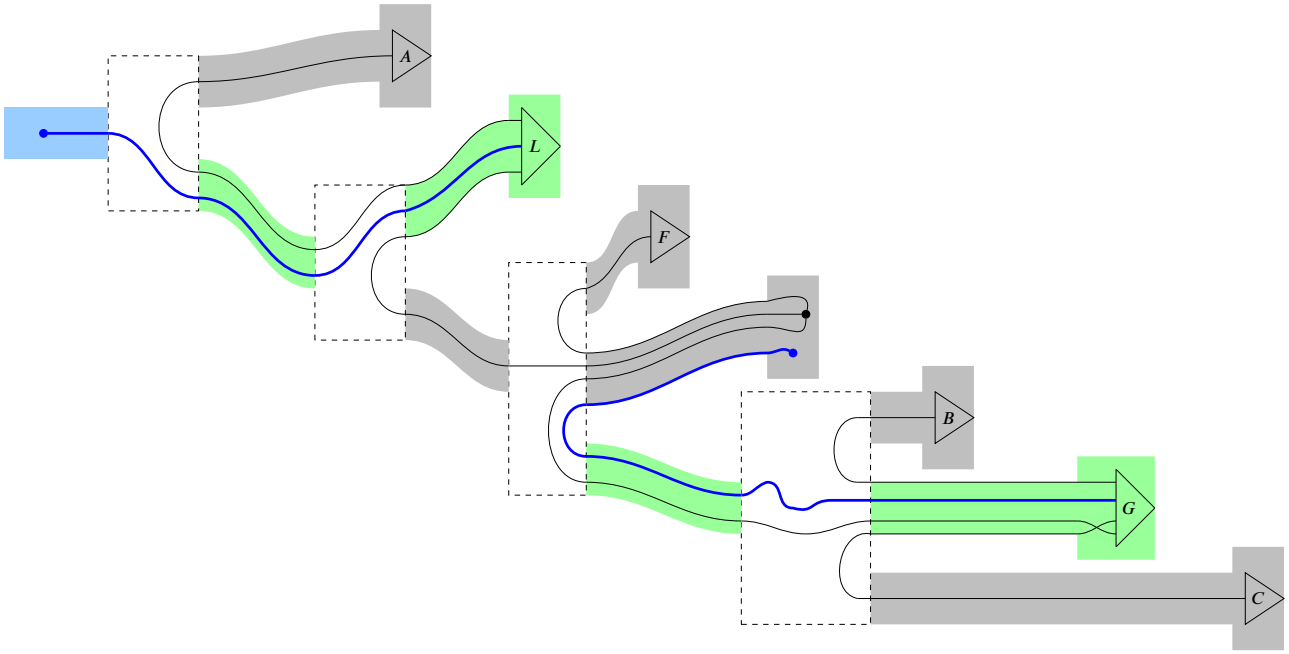


Figure 4: Now that Charlie can express their grammatical structure string-diagrammatically, they can try to deform their first-order-logic diagram – representing what they mean to communicate – subject to the constraint that every one of their branchings (the structure of the CFG) is something recoverable by Dennis using just pregroup reductions. To do so, Charlie introduces a formal blue wire to mimic Dennis's sentence-type, and stuffs some complexity inside the labels in the form of internal wirings: a multiwire configuration for `that`, and a twist for `gives`. Those internal wirings are the content of Charlie and Dennis's shared strategy. In passing, I'll remark that by the outside-in convention for functor boxes 13, this diagram constitutes a monoidal functor from this particular CFG to pregroup diagrams, where nonlabel tree-nodes are partial monoidal closure evaluators. Replacing rigid autonomous closure with cartesian closure and n, s with e, t recovers montague semantics for CFGs (c.f. Curry-Howard-Lambek correspondence for the case of typed lambda-calculus and cartesian closed categories, and all of Heim and Kratzer (**?**)), and interpreting the closure in a compact closed setting recovers montague semantics for CCGs (**?**).

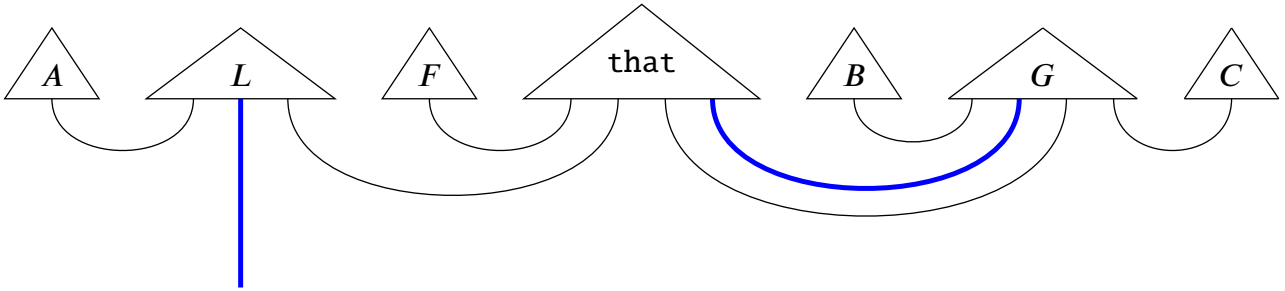


Figure 5: So, when Dennis receives the sentence, Dennis's pregroup derivation yields a pregroup diagram that is connectively equivalent to what Charlie stuffed inside the context-free grammar structure. So now the two have strong equivalence between their grammars in the sense that every one of Charlie's branches is resolved by one of Dennis's reductions. As is convention for pregroup diagrams, we only use types n and s – the latter denoted by a blue wire here – and we'll leave the directionality (rigid autonomous turning number) of wires implicit, so you can either trust me that everything typechecks or do it yourself.

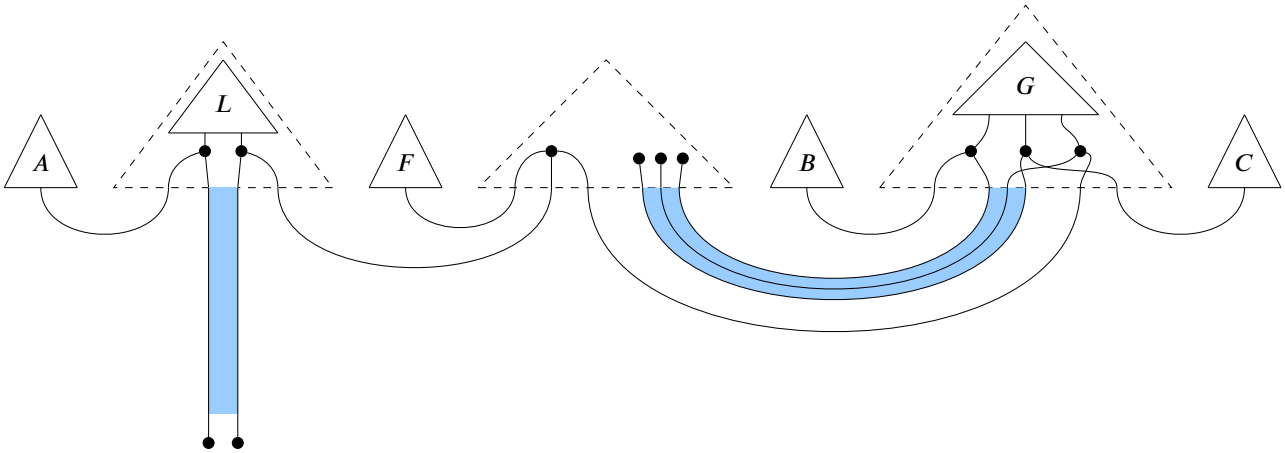


Figure 6: Now to fully recover Charlie's intended FOL-diagram, Dennis refers to the internal wirings from their shared strategy, and fills those in.

Example 0.1.2 (`Bob gives Claire flowers. Alice likes flowers.`)**.** Now we try the same content as the previous example but presented as a text with two sentences.

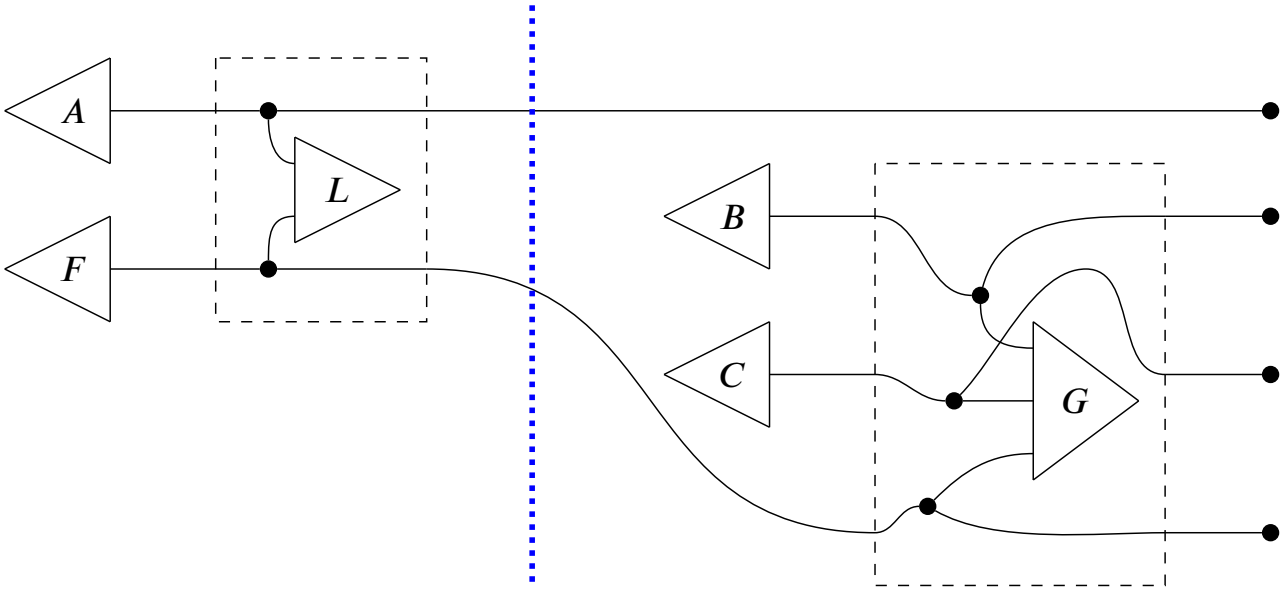


Figure 7: Charlie's diagram morphed to fit a text circuit. The dotted blue line is a formal mark to indicate a sentential boundary. Observe how new discourse elements are introduced as states, and how open wires correspond to ongoing discourse and deletions mark completed discourse. This diagram also indicates that text circuits can be given semantics in FOL.

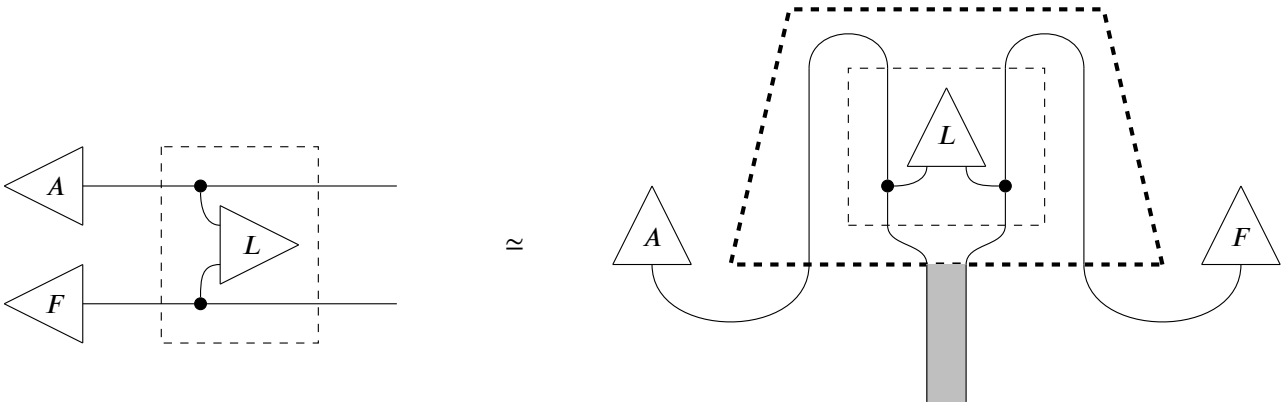


Figure 8: Dennis already knows how to parse individual sentences to extract the FOL using internal wirings. Observe there is a mathematical complication that arises in determining how many noun-wires should go into the sentence wire-bundle; we need to account for this later.

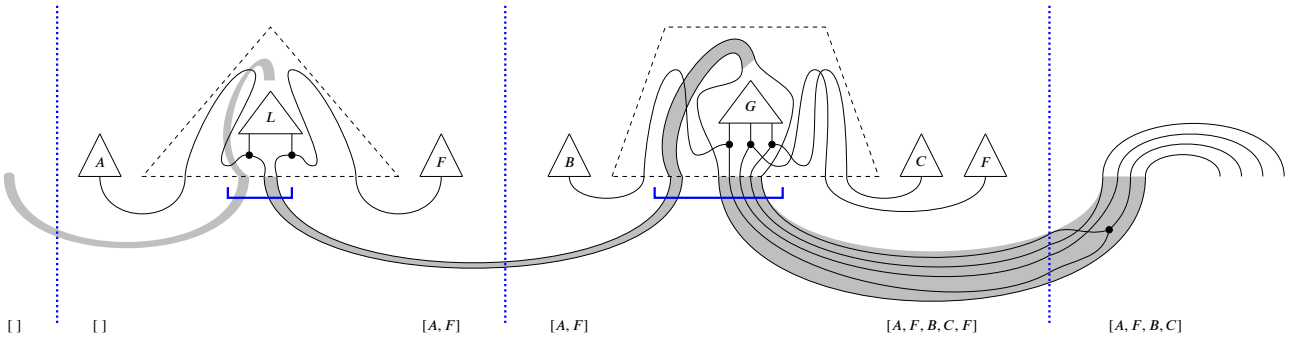


Figure 9: To deal with text, Dennis can pass a growing bundle of sentence wires along horizontally.

The essence of internal wirings: The relational content that is communicated by language is not inherently one-dimensional, but must be encoded in and decoded from the one-dimensional strings of language. Internal wirings CITES provide a way to approach this coding problem topologically: while productive and parsing grammars have different topologies, by choosing internal wirings for individual words, the speaker and listener can obtain topologically equivalent representations.

0.1.1 *An issue with functorial semantics of internal wirings*

There is a mathematical issue: the "filling in" of internal wirings is not *in general* functorial, for either speaker or listener. The issue in both cases is that sometimes the particular internal wiring depends on what words are around it.

Figure 10:

Example 0.1.3 (Nonfunctoriality of internal wirings for productive grammars)**.**

Let's consider an easy context-free grammar, with just four types and three rules apart from labels. The types are: `S` for sentences, `N` for nouns, `ADV` for adverbs, and `V` for verbs. There is a single adverb introduction rule, and two verb introduction rules for intransitive and transitive verbs.

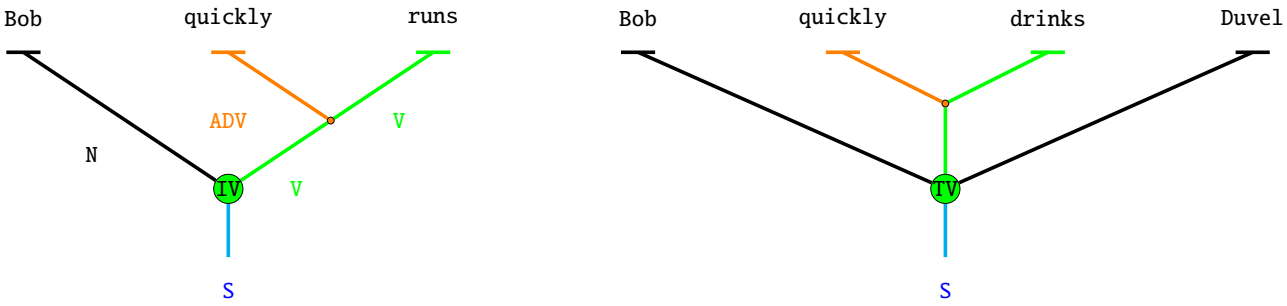


Figure 11: Now suppose we want to describe a functor from this context free grammar to a pregroup grammar with just types n and s. We know how verb states ought to look, and we know that adverbs ought to modify a verb. We can get pretty close with a first sketch, depicting the desired action of the functor using the outside-in convention for functor boxes, and we can slim them down to tubes. Now the simplicity of the CFG reveals a complication. Since there are two possible kinds of verbs, there are two possible kinds of adverbs, and accordingly two possible kinds of adverb introduction rules. A functor from the CFG to a pregroup diagram can't send the single adverb introduction rule to two different things at the same time.

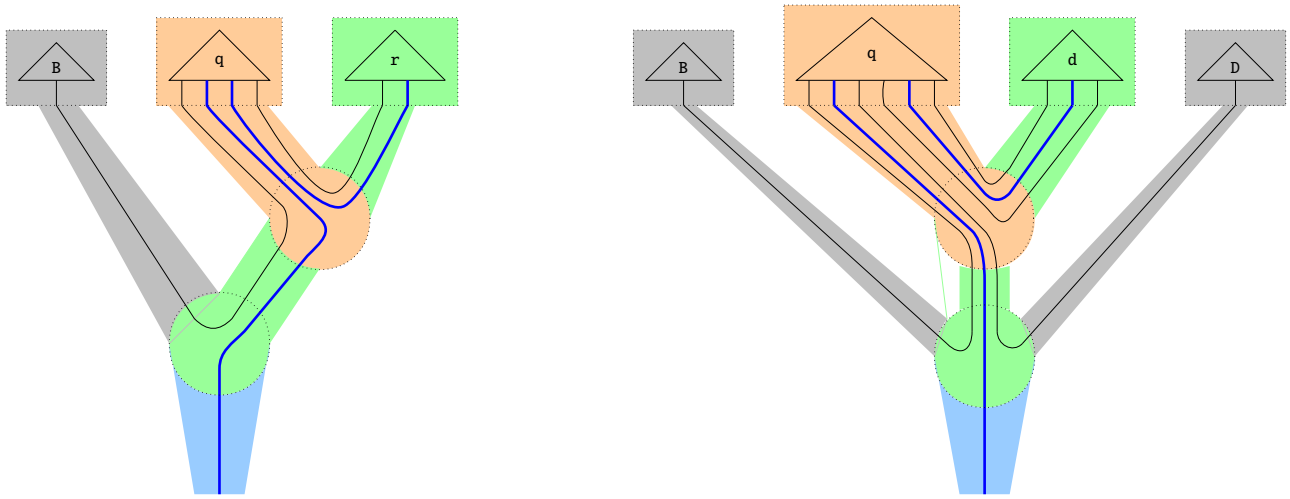
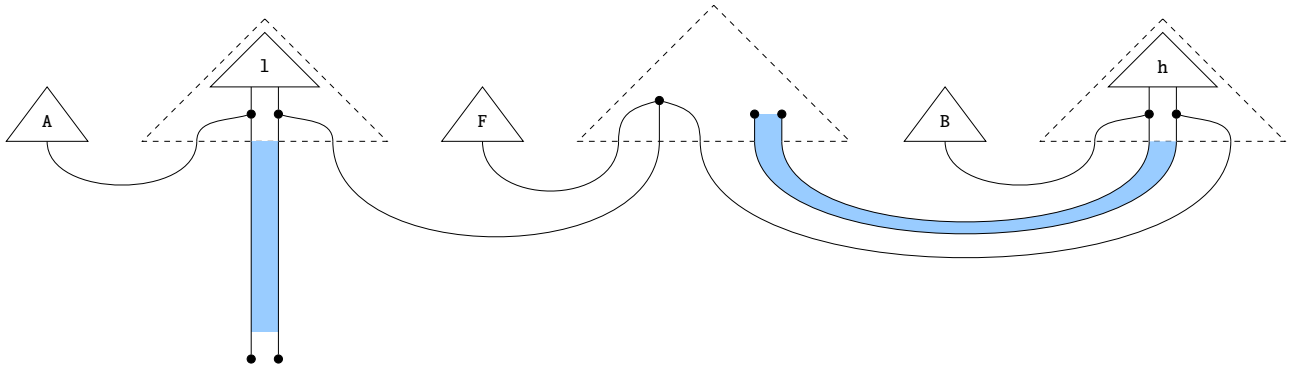


Figure 12:

Example 0.1.4 (Nonfunctoriality of internal wirings for parsing grammars)**.**

Compare `Alice likes flowers that Bob hates` to the sentence in Figure 6; here the object relative pronoun `that` is connected to a transitive verb `hates` rather than a ditransitive `gives`. The internal wirings work fine in this example, but now `that` deletes two wires instead of three; a functor can't map the same word-state to two possible instantiations.



So we're in a situation where internal wirings are conceptually nice and work well *within* examples, but are not described *across* examples by functorial semantics. It turns out that the right kind of ...

0.2 *Discrete Monoidal Opfibrations*

To capture the kinds of diagrammatic correspondences we have just sketched, we will develop monoidal cofunctors diagrammatically. The first step is introducing the concept of a discrete monoidal fibration[1] : a mathematical bookkeeping tool that relates kinds of choices speakers and listeners make when generating and parsing text respectively. This in turn will require introducing *monoidal functor boxes*.

[1] Expressing the coherence conditions of monoidal functors using equations involving functor boxes as below is not new CITE . The idea of a functor being simultaneously monoidal and a fibration is not new CITE . What is new is minor: the express requirement that the lifts of the fibration satisfy interchange, which is (due to a personal communication with Fosco Loregian) in general not implied when a functor is both monoidal and a discrete fibration.

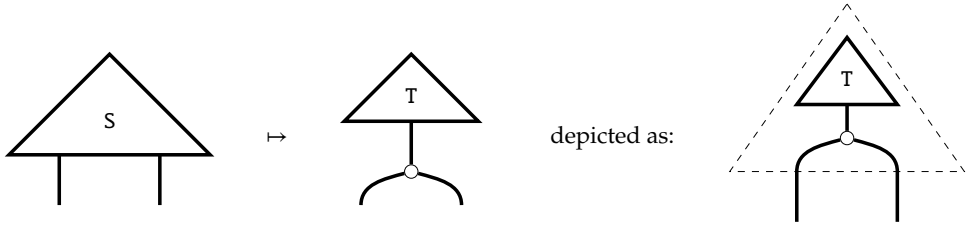


Figure 13: There are two conventions for depicting the action of a monoidal functor on parts of a string diagram. The first follows source-to-target *outside-in*. This convention is used for other work in internal wirings, since it is well-suited for describing functors that send atomic generators in their domain to more complex diagrams in their domain.

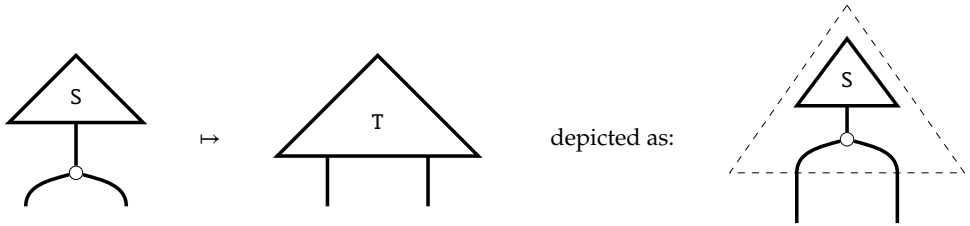


Figure 14: The other convention, following CITE , is *inside-out*. For the following section, we will define the coherence conditions of discrete monoidal fibrations using this convention.

Figure 15: Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \to \mathcal{D}$. Then we have this diagrammatic representation of a morphism $\mathbf{F}A \xrightarrow{\mathbf{F}f} \mathbf{F}B$ in $\mathcal{D}$.

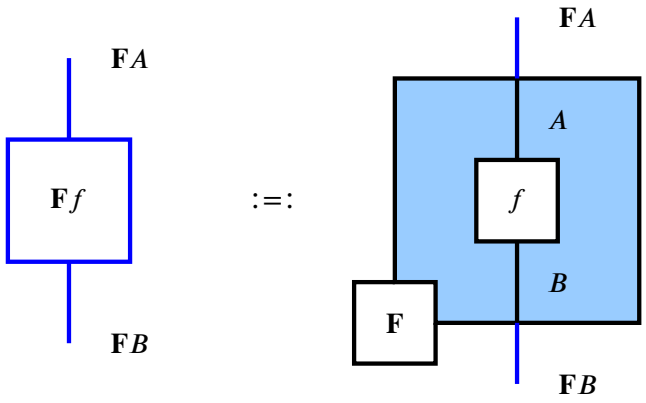


Figure 16: The use of a functor box is like a window from the target category $\mathcal{D}$ into the source category $\mathcal{C}$; when we know that a morphism in $\mathcal{D}$ is the image under $\mathbf{F}$ of some morphism in $\mathcal{C}$, the functor box notation is just a way of presenting all of that data at once. Since $\mathbf{F}$ is a functor, we must have that $\mathbf{F}f ; \mathbf{F}g = \mathbf{F}(f ; g)$. Diagrammatically this equation is represented by freely splitting and merging functor boxes vertically. **N.B.** sequential merging of two boxes requires that the two wires to-be-connected within the boxes – in this case labelled B – need to be the same; a case where merging is disallowed is when $Ff ; Fg$ typechecks in the outside/target category, but $f ; g$ does not in the inside/source category because the functor identifies nonequal wires.

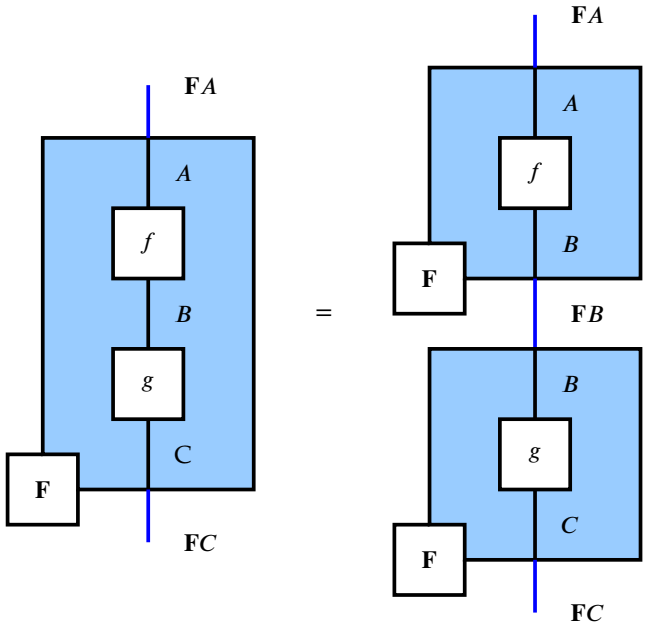


Figure 17: Assume that $\mathbf{F}$ is strict monoidal; without loss of generality by the strictification theorem CITE , this lets us gloss over the associators and unitors. For $\mathbf{F}$ to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A \otimes_D \mathbf{F}B = \mathbf{F}(A \otimes_C B)$. Diagrammatically these structural constraints amount to these equations.

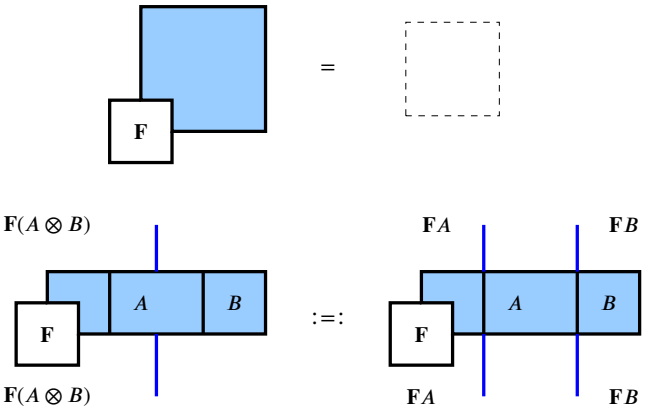


Figure 18: What remains is the monoidality of $\mathbf{F}$, which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes' way of respecting sequential composition, splitting horizontally is how they respect parallel composition.

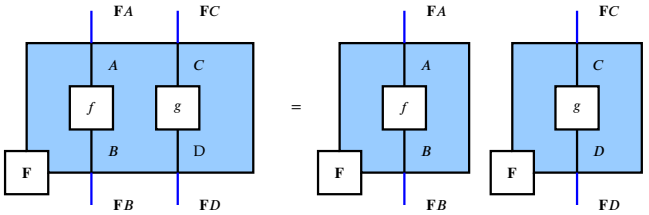


Figure 19: And for when we want $\mathbf{F}$ to be a (strict) symmetric monoidal functor, we are just asking that boxes and twists do not get stuck on one another.

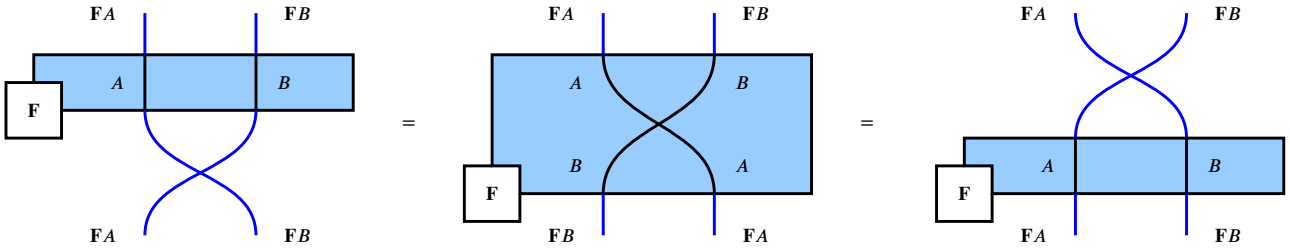


Figure 20: To motivate fibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a functor box out of the bottom. When can we do the reverse? That is, take a morphism in $\mathcal{D}$ and *slide it into* a functor box? We know that in general this is not possible, because not all morphisms in $\mathcal{D}$ may be in the image of $\mathbf{F}$. So instead we ask "under what circumstances" can we do this for a functor $\mathbf{F}$? The answer is when $\mathbf{F}$ is a discrete fibration.

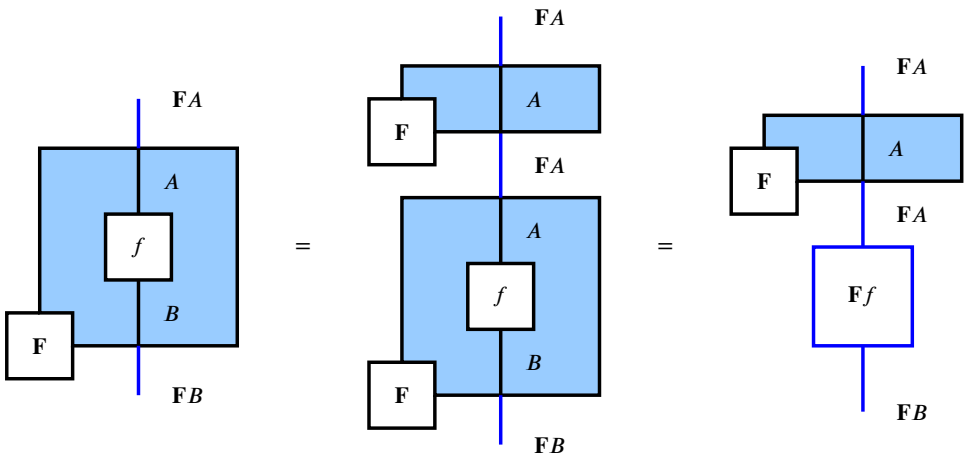


Figure 21:

Definition 0.2.1 (Discrete opfibration). $\mathbf{F} : \mathcal{C} \to \mathcal{D}$ is a *discrete fibration* when: for all morphisms $f : \mathbf{F}A \to B$ in $\mathcal{D}$ with domain in the image of $\mathbf{F}$, there exists a unique object Φ_f^A and a unique morphism $\phi_f : A \to \Phi_f^A$ in $\mathcal{C}$, such that $f = \mathbf{F}\phi_f$. Diagrammatically, we can present all of the above as an equation reminiscent of sliding a morphism *into* a functor box from below. The process inside the box is called *the lift* of the process that was slid in. The collection of all lifts over a wire or box is called *the fibre over* that wire or box.

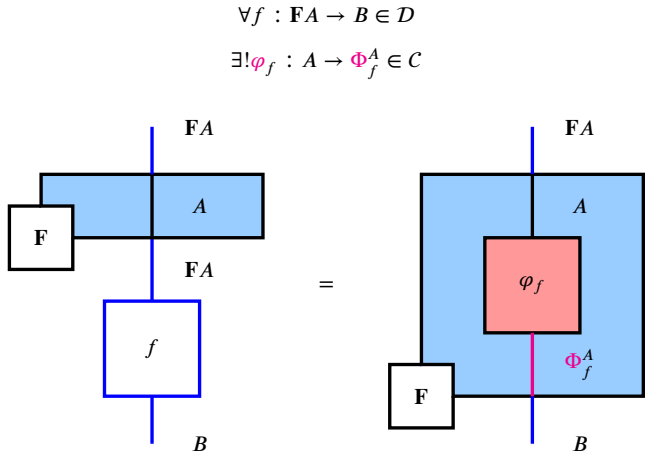
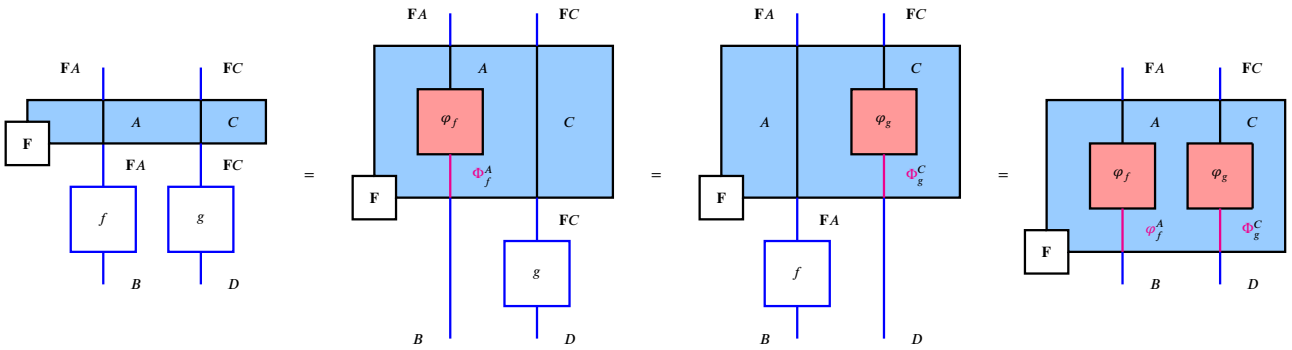


Figure 22:



Definition 0.2.2 (Monoidal discrete opfibration)**.** We consider **F** to be a *(strict, symmetric) monoidal discrete opfibration* when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the depicted equations relating lifts to interchange hold. The diagrammatic motivation for the additional coherence equations is that – if we view the lifts of opfibrations as sliding morphisms into functor boxes – we do not want the order in which sliding occurs to affect the final result. In this way, lifts behave as 'graphical primitives' in the same manner as interchange isotopies and symmetry twists.

0.2.1 What are they good for?

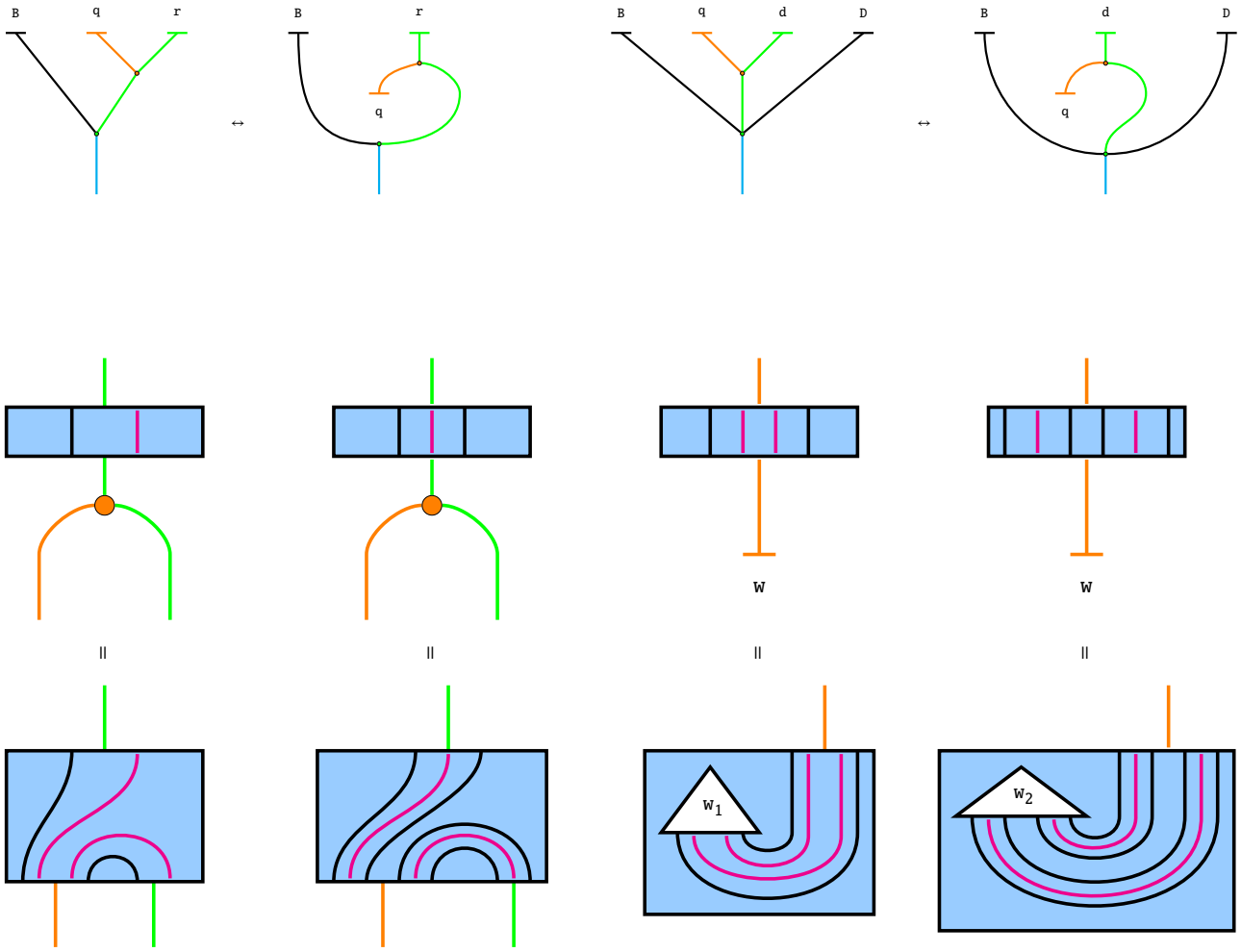


Figure 23: Now we try to use monoidal discrete opfibrations to help us solve the speaker's nonfunctoriality problem (Example 10). First we flip over the labels and introduction rules for adverbs. Call this a *dependent CFG*, or *dCFG*. Treating the label as a test rather than a state will allow the fibration-box to choose the right version based on the domain wires as it expands top-down. In this case, since CFGs are planar, flipping causes no confusion, since we can always flip the labels back over. There are several ways to do this formally, by e.g. specifying a new string-diagram signature from the old one or assuming rigid autonomous completion, and it doesn't matter which we use.

Figure 24: Recall that opfibrations can decide which lift to depict given a choice of codomain wires. We would like to encode the dependency of the upside-down adverb labels and introduction rules as lifts that depend on the lift of the verb wire, which may be either an intransitive or transitive verb.

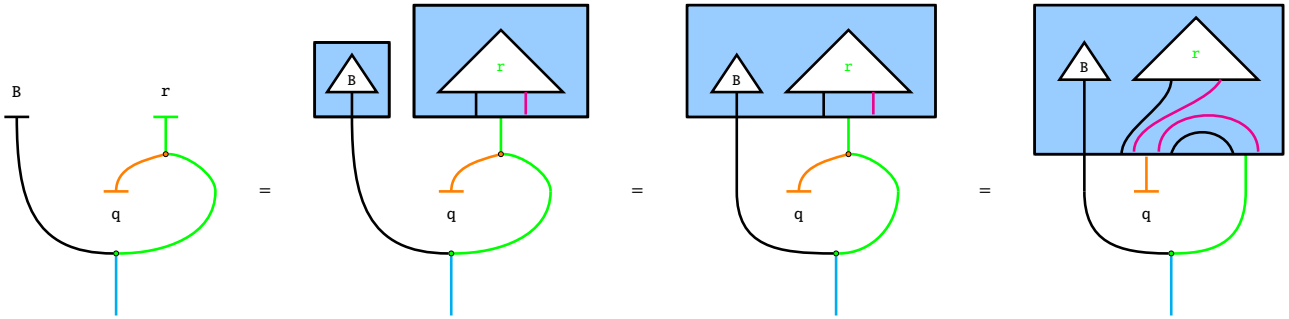


Figure 25: Instead of *us* making the choice, we can force the choice using the information of the CFG structure. Starting from a dCFG diagram, the state-labels have unique lifts: noun labels in CFGs correspond uniquely to noun-states in pregroup diagrams, and verb labels to verb-states which may be either intransitive or transitive. This obtains the first equation. The second equation is obtained by monoidality. The third "eating downwards" equation is obtained by the opfibration property; note that because the codomain wires before the lift are already decided to be those of an intransitive verb's pregroup type, the correct adverb introduction rule can be selected for the lift.

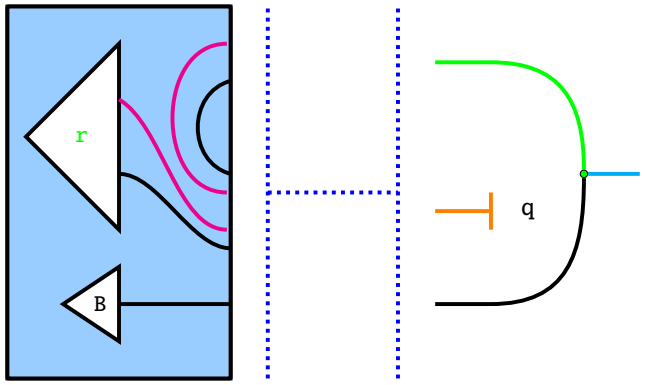


Figure 26: But there's a technical problem. We have been assigning wires from the codomain of the lift to the dCFG implicitly, by grouping wires together visually to indicate which wires inside the functor box correspond to wires outside. However, when we consider the algebraic data available, all we know is depicted in the figure: we need some way to assign the wires. Solving the wire assignment probem will be the focus of the next section.

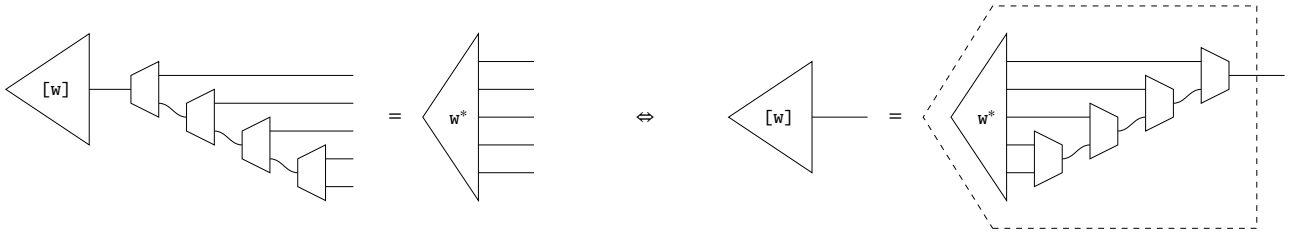
0.3 *Strictified diagrams for monoidal categories*

The crux of the issue sketched in Figure 26 is that while pregroup *proofs* – viewed as sequent trees – syntactically distinguish the roots of subtrees, interpretation as pregroup *diagrams* in a monoidal category forgets the subtree structure of the specific proof the diagram arises from. But it is precisely this forgotten structure that contains the algebraic data we require to keep track of (co)domain data diagrammatically. So a solution would be to force the diagrams in the blue domain recording pregroup data to hold onto this proof structure. For this purpose we use strictified diagrams for monoidal categories, defined in the margins.

We are seeking some way to algebraically group or bracket together pregroup types that arise from a single word, in a distinguished way from concatenation-as-tensor. In this way we can preserve the structure of pregroup-sequent proofs: grouping indicates a node in the proof-tree, while tensor indicates parallel composition of proof trees. With strictified diagrams, we can model bracketing with biased tensor structure, e.g. treating for instance the left-nested tensoring $(\cdots((A \otimes B) \otimes C) \cdots \otimes \cdots Z)$ as a bracketed expression $[A \otimes B \cdots \otimes Z]$.

Construction 0.3.3 (Pregroups with bracketing)**.** Where **PGD** is a rigid monoidal category generated by pregroup states and (directed) cups, we define pregroups-with-bracketing as a category denoted $\overline{\mathbf{PGD}}^{*}$, which is obtained as follows. Throughout, denote the rigid monoidal tensor product as $\otimes$ and the strictified tensor as $\bullet$.

For each pregroup state $\mathtt{w} : I_{\otimes} \to x_1 \otimes \cdot \otimes x_n$ that generates **PGD**, we create two corresponding generators $\mathtt{w}^* : I_{\bullet} \to x_1 \bullet \cdot \bullet x_n$ and $\mathtt{[w]} : I_{\bullet} \to (\cdots(x_1 \bullet x_2) \bullet x_3) \cdots \bullet x_n)$. `[w]` is a left-bracketed tensoring, and $\mathtt{w}^*$ is fully detensored. Note that `[w]` and $\mathtt{w}^*$ coincide for words typed with singletons. We ask for the following family of relations: either the left or the right implies the other in the presence of equations governing the structural isomorphisms.



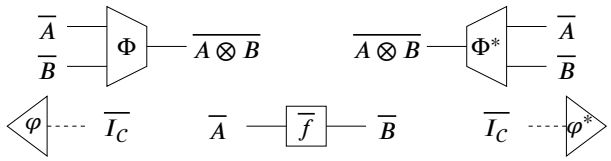
The $\mathtt{w}^*$ and `[w]` generate a freely strictified rigid autonomous category $\overline{\mathbf{PGD}}^{\dagger}$, from which we obtain the desired $\overline{\mathbf{PGD}}^{\star}$ as a subcategory generated by:

1. All `[w]`
2. Let $[A \cdot B \cdots Z]$ denote the left-nested tensoring $((A \otimes B) \cdots \otimes Z)$, and let $\mathbf{X}$ denote $(\bigotimes_i X_i)$. For each di-

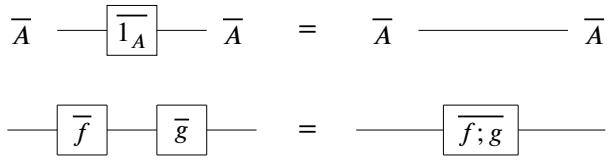
Definition 0.3.1 (Strictified string diagrams)**.** (Presentation taken from CITE) Fix an arbitrary (non-strict) monoidal category $\mathcal{C}$. The *strictification* $(\overline{\mathcal{C}}, \bullet)$ is defined as follows (where strictness of $\overline{\mathcal{C}}$ entitles use of string-diagrammatic notation):

(1) Objects $\overline{A}$ for each $A \in \mathcal{C}$

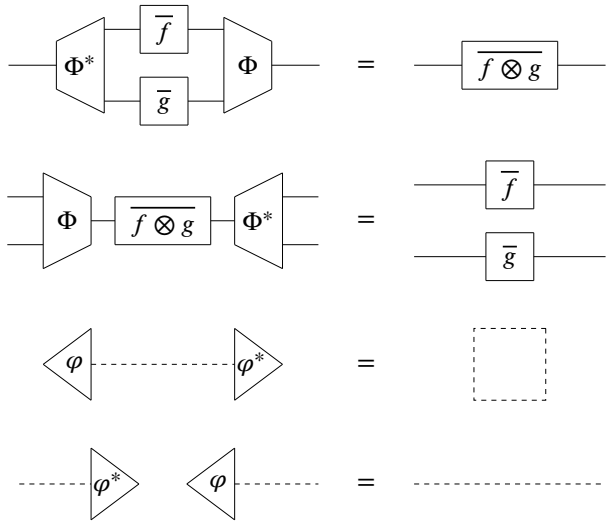
(2) The following generators, with $\overline{f} : \overline{A} \to \overline{B}$ for each $f \in \mathcal{C}(A, B)$, where we adopt the convention of notating the monoidal unit with a dashed line:



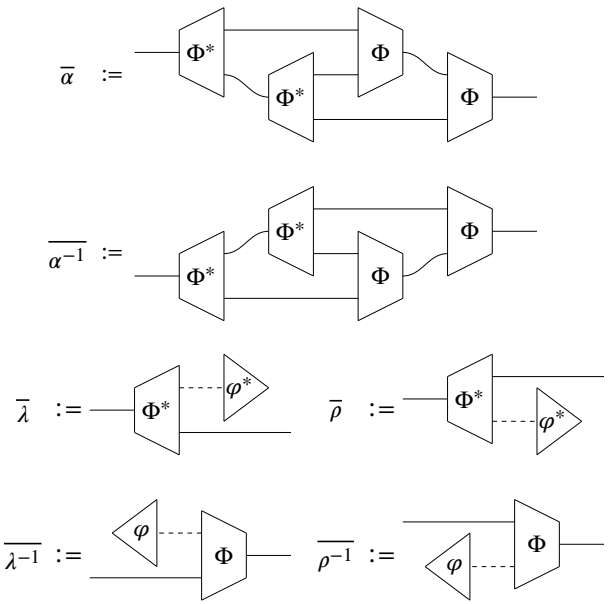
(3) The following functoriality equations:



(4) The following adapter equations:

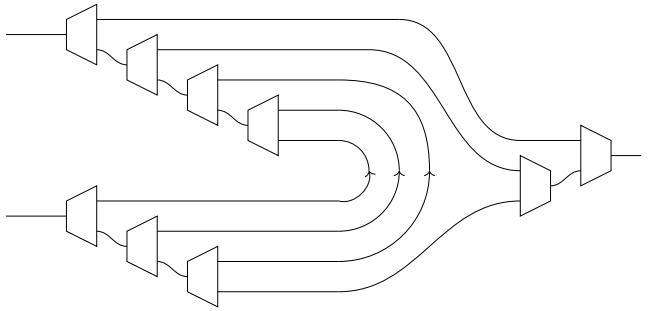


(3) The following representations of the natural isomorphisms in the definition of a monoidal category:



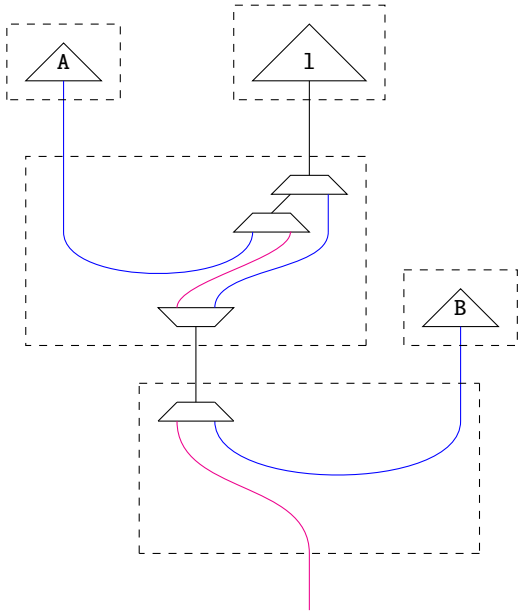
Proposition 0.3.2 ($\bar{\mathcal{M}}$ and $\mathcal{M}$ are monoidally equivalent). *Proof.*
CITE □

rected cap $\mathbf{X} \otimes \mathbf{X}^{-1} \to I$ (and symmetrically for caps of the other direction and cups), and for each pair of bracketed types $[\mathbf{A} \cdot \mathbf{X}]$ and $[\mathbf{X}^{-1} \cdot \mathbf{B}]$, we ask for a generator that detensors, applies the directed cup on $\mathbf{S}$, and then retensors while respecting the bracketing structure of $\mathbf{A}$ and $\mathbf{B}$ to obtain $[\mathbf{A} \cdot \mathbf{B}]$. Diagrammatically this amounts to asking for generators that look like the following, that mimick a single proof step.

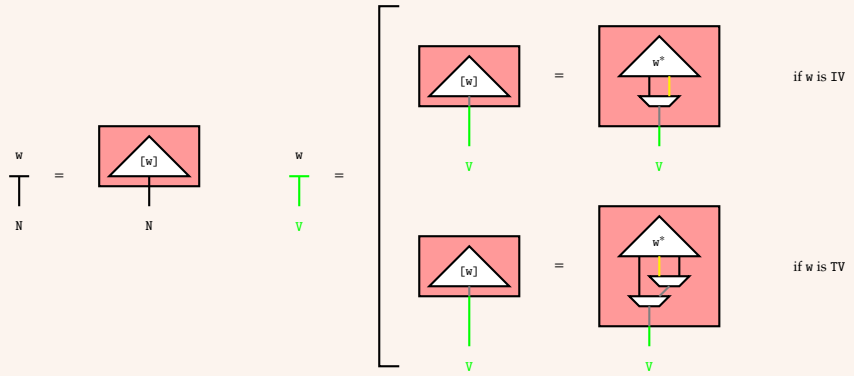


Example 0.3.4 (Pregroups with bracketing recover proof trees)**.** The essence of the construction is to maintain a correspondence with proof-tree structure: a left-bracketed collection of types corresponds to a pregroup typing that is stuck together as the outcome of a sequent rule and must thereafter travel together. Starting from bracketed word states `[w]`, point 2 of the construction maintains an invariant correspondence that bracketed collections of types are the roots of proof steps.

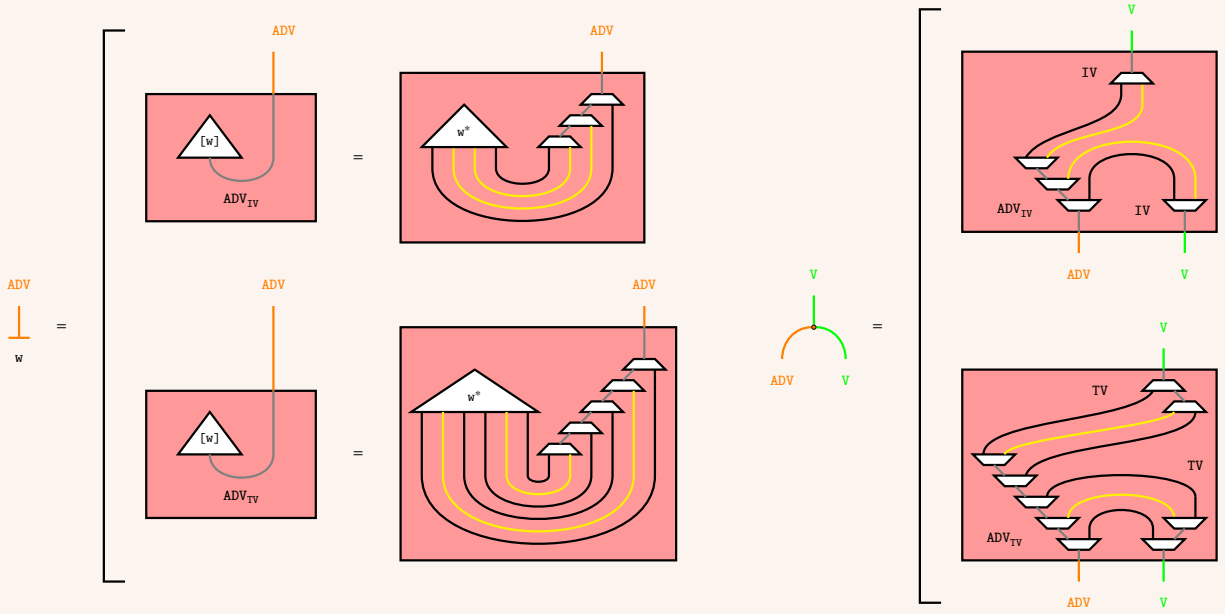
$$\dfrac{\dfrac{\texttt{Alice} : n \qquad \texttt{likes} : {}^{-}n \cdot s \cdot n^{-}}{\texttt{Alice␣likes} : s \cdot n^{-}} \qquad \texttt{Bob} : n}{\texttt{Alice␣likes␣Bob} : s}$$



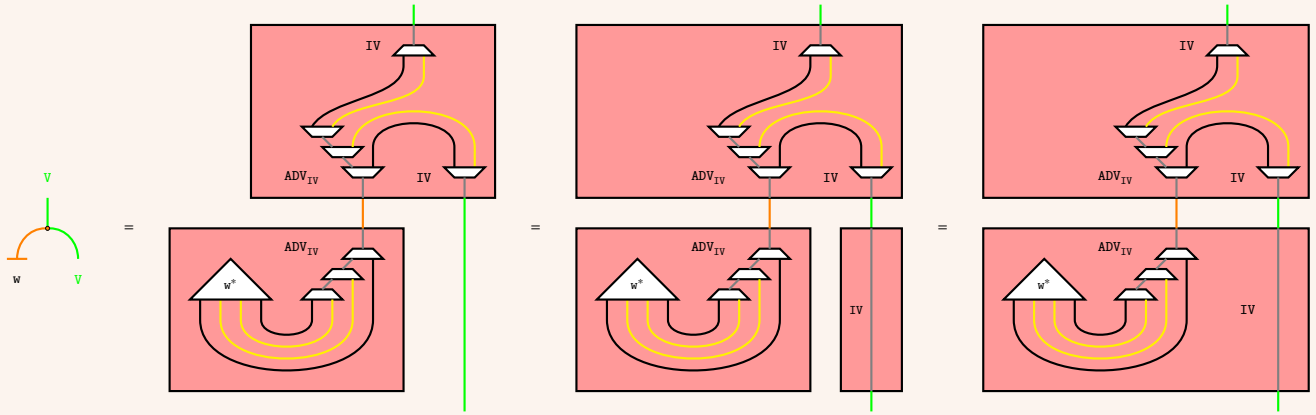
Construction 0.3.5 (Discrete monoidal opfibration from pregroups with bracketing into dependent CFGs). We aim to elucidate the pregroup with bracketing in sufficient detail to describe the functor into dCFGs; in particular, we need to know what the generators of the pregroup are. The fibre over a noun state in the dCFG is the corresponding noun-state in the bracketed pregroup. The fibre over a verb state in the dCFG is either an intransitive or transitive word-state, depending on the word `w`. Note that only the `[w]` are available as generators, through we may reason about them as if they are tensor-bracketings of `w`*.



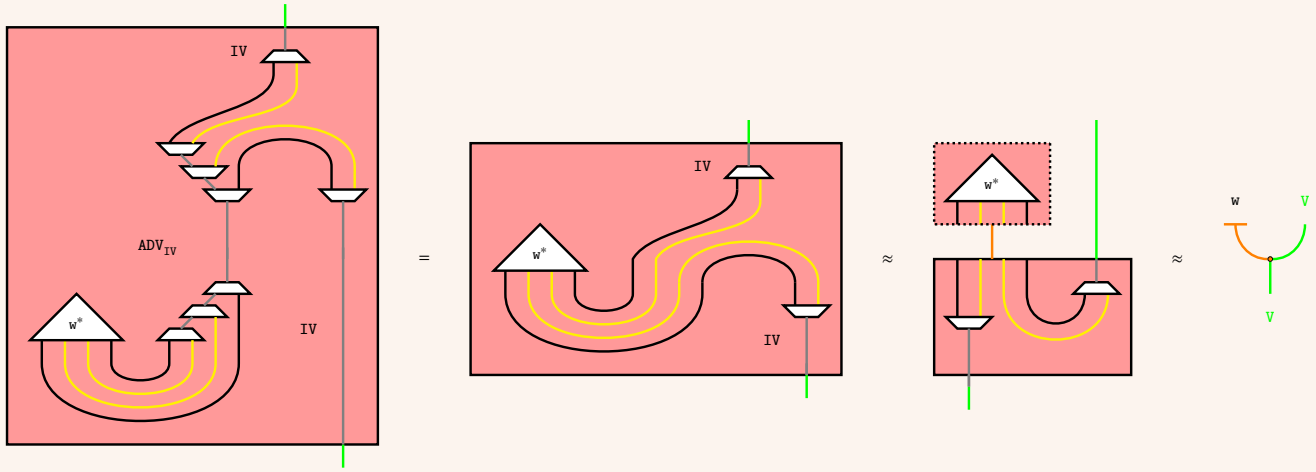
We depict the case of adverbs explicitly. Here are two lifts of the adverb label in the fibre of the opfibration corresponding to the intransitive and transitive case, and the corresponding lifts for the introduction rule for adverbs.



The correspondence of introduction rules in the dCFG to proof steps in the sequent formulation of pregroup proofs (c.f. Example 0.3.4) is obtained obliquely, because labels for dependent types are facing the wrong way; hence the cups for the lifts of labels. We can observe the correspondence by the following diagrams. The second equation is a supplied choice of lift on the V wire, so that the third monoidality equation allows the top and bottom boxes to typematch.



When (and only when) the types are matched, the boxes may be sequentially (vertically) merged. Now within the box, we may apply the equations available to us in the strictified setting to eliminate the (de)tensors. Observe that resolving snaking wires causes the second diagram to *behave as though* we defined lifts for "right-side-up" labels and introduction rules; we could not have done so directly, or else the opfibration would have no diagrammatic way to determine the correct lift.



The other lifts for other types are obtained similarly, by the solutions of a system of pregroup equations with boundary conditions that treat dCFG types as variable pregroup types in n and s. The dCFG types are:

$$\mathtt{S}, \mathtt{N}, \mathtt{V}, \mathtt{ADV}, \mathtt{ADP}$$

The determined equations of the system are the assignments of the types $\mathtt{N}$ and $\mathtt{S}$.

$$\mathtt{S} = s$$
$$\mathtt{N} = n$$

The boundary conditions are given by the particular verbs in the sentence, which may come in three kinds: intransitive, transitive, or verbs that take a sentential complement, such as sees.

$$\mathtt{V} = ({}^{-}n \cdot s) \text{ or } ({}^{-}n \cdot s \cdot n^{-}) \text{ or } ({}^{-}n \cdot s \cdot s^{-})$$

Which we rewrite using the following index system to indicate noun structure. Intransitive verbs are assigned an index 1, and transitive verbs an index 2.

$$\mathtt{V}_1 = ({}^{-}n \cdot s)$$
$$\mathtt{V}_2 = ({}^{-}n \cdot s \cdot n^{-})$$

The three dependent types are:

$$\mathtt{V}_{x \mapsto 1(x)} = ({}^{-}n \cdot s)$$
$$\mathtt{ADV}_x = \mathtt{V}_x \cdot \mathtt{V}_x^{-}$$
$$\mathtt{ADP}_x = {}^{-}\mathtt{V}_x \cdot \mathtt{V}_{(x)1} \cdot n^{-}$$

The types $\mathtt{V}$, $\mathtt{ADV}$, $\mathtt{ADP}$ are hence indexed over a string language $x := 1 \mid 2 \mid 1(x) \mid (x)1$. We have depicted the solutions for $\mathtt{ADV}_1$ and $\mathtt{ADV}_2$. The rest are obtainable inductively, where the bracketing structure is handled by the tensor and detensors of the strictified pregroup diagrams. The pregroup typing solutions for $\mathtt{V}$, $\mathtt{ADV}$, $\mathtt{ADP}$ across indices are unique, as the latter two generators of the string language correspond to verbs with sentential complement and adpositions respectively, so by the bracketing structure, indices correspond uniquely to dCFG diagrams up to labels. The family of pregroup typing solution yield the required generators, which we use to populate the fibres over the three dependent type labels and their introduction rules.

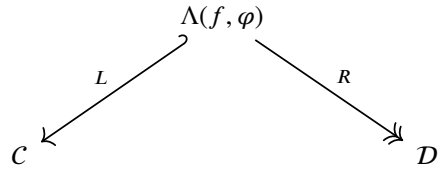
Proposition 0.3.6 (Construction 0.3.5 is a discrete monoidal fibration)**.** *Proof.* Monoidality is evident. For unique lifts, we observe that for each bracketing of wires, construction 0.3.3 guarantees a unique lift for each introduction rule in the dCFG, and Construction 0.3.5 guarantees a unique lift for each label. For the additional interchange condition of Definition 22, it suffices to observe that the introduction rules of dependent labels uniquely determine the codomain of the lift given the domain, and that by design in Construction 0.3.5, independent labels as states have predetermined lifts. □

0.4 Monoidal cofunctor boxes

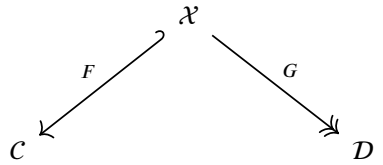
Now that we know how to solve the wire-assignment problem with the help of monoidal discrete opfibrations from strictified diagrams that do bracketing, we can at last see what monoidal cofunctor boxes are.

Definition 0.4.3 (Bijective-on-objects functor)**.** From CITE, Defn 2.8 . A functor $F : \mathcal{C} \to \mathcal{D}$ is *bijective-on-objects* if for all $d \in \mathcal{D}$, there exists a $c \in \mathcal{C}$ such that $Fc = d$.

Proposition 0.4.4 (Cofunctors as spans of functors)**.** From CITE, Prop 2.10 , all cofunctors $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ correspond to spans of functors where the left leg L is bijective on objects and the right leg R is a discrete opfibration:



Conversely, by CITE, Prop 2.10 , for every span of functors where the left leg is bijective on objects and the right leg is a discrete opfibration,



there exists a cofunctor $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ and an isomorphism $J : \Lambda(f, \varphi) \to \mathcal{X}$ such that $FJ = L$ and $GJ = R$.

Definition 0.4.5 (Monoidal cofunctor)**.** A monoidal cofunctor, following Proposition 0.4.4, is a span of functors such that the left leg is monoidal and bijective-on-objects, and the right is a monoidal discrete opfibration by Definition 22.

Construction 0.4.6 (Monoidal cofunctor box)**.** A monoidal cofunctor box first uses the inside-out convention for functor boxes for the right leg, and then the outside-in convention for the left leg.

These definitions and conventions follow CITE . Given a (small) category $\mathcal{C}$ we notate the objects $\mathcal{C}_0$ and the morphisms $\mathcal{C}_1$, hence a functor $F : \mathcal{C} \to \mathcal{D}$ consists of an object assignment $F_0 : \mathcal{C}_0 \to \mathcal{D}_0$ and a morphism assignment $F_1 : \mathcal{C}_1 \to \mathcal{D}_1$.

Definition 0.4.1 (Cofunctors)**.** From CITE Defn 2.2 . A *cofunctor* $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ consists of a function $f : \mathcal{C}_0 \to \mathcal{D}_0$ which I'll call *lowering*, together with a *lifting operation* φ, a function that maps pairs of objects of $\mathcal{C}$ and certain morphisms in $\mathcal{D}$ to morphisms of $\mathcal{C}$:

$$(c \in \mathcal{C}_0, f(c) \xrightarrow{u} b \in \mathcal{D}_1) \mapsto a \xrightarrow{\varphi(c,u)} \mathrm{cod}(\varphi(c,u))$$

The following conditions are required:

1. Lowering the tip of a lifted arrow gets you back where you started.

$$f(\mathrm{cod}(\varphi(c,u))) = b$$

2. The lifts of identities are identities.

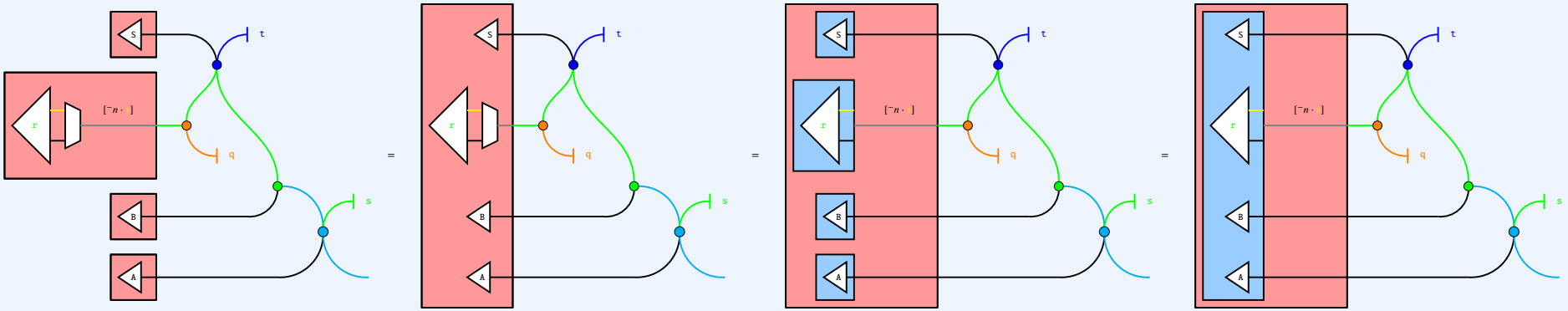
$$\varphi(c, 1_{f(c)}) = 1_c$$

3. The lift of composites is the composite of lifts-with-respect-to the tips of lifted arrows.

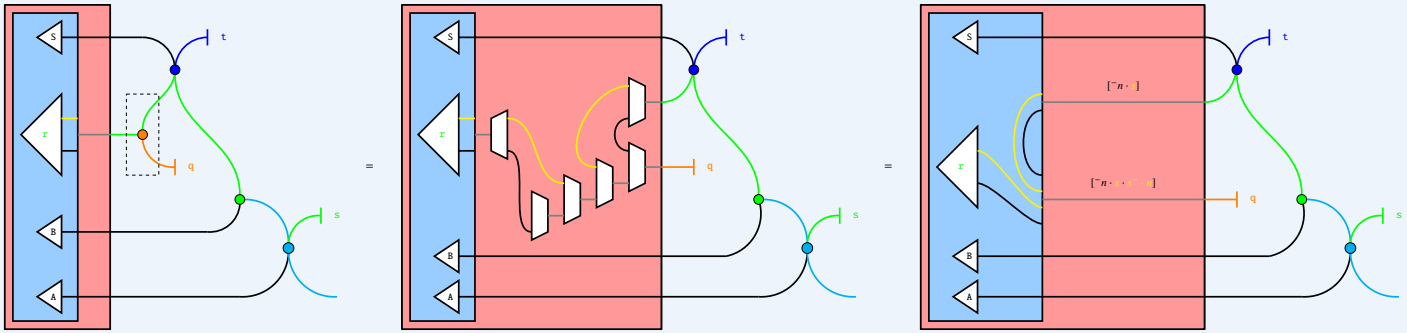
$$\varphi(c, v \circ u) = \varphi(\mathrm{cod}(\mathrm{cod}(\varphi(c,u))), v) \circ \varphi(c,u)$$

Remark 0.4.2. Conditions 2 and 3 of the definition of cofunctor are reminiscent of functors. It is instructive but tedious to calculate with the base definition. We use the slick alternative formulation by Bryce Clarke.

Example 0.4.7 (Turning dCFGs into pregroup diagrams with a monoidal cofunctor)**.** The apex is given by Construction 0.3.3. The right leg is the monoidal discrete opfibration from Construction 0.3.5 into the dCFG. In the first diagram below, we apply the opfibration to the word states, which have unique lifts. The second diagram follows by monoidality. In the third, we apply the left leg of the cofunctor using the outside-in convention, which is the evidently bijective-on-objects monoidal functor to the ambient rigid autonomous category of pregroup diagrams from the free strictification. In the fourth, we apply the monoidality of the inner functor.

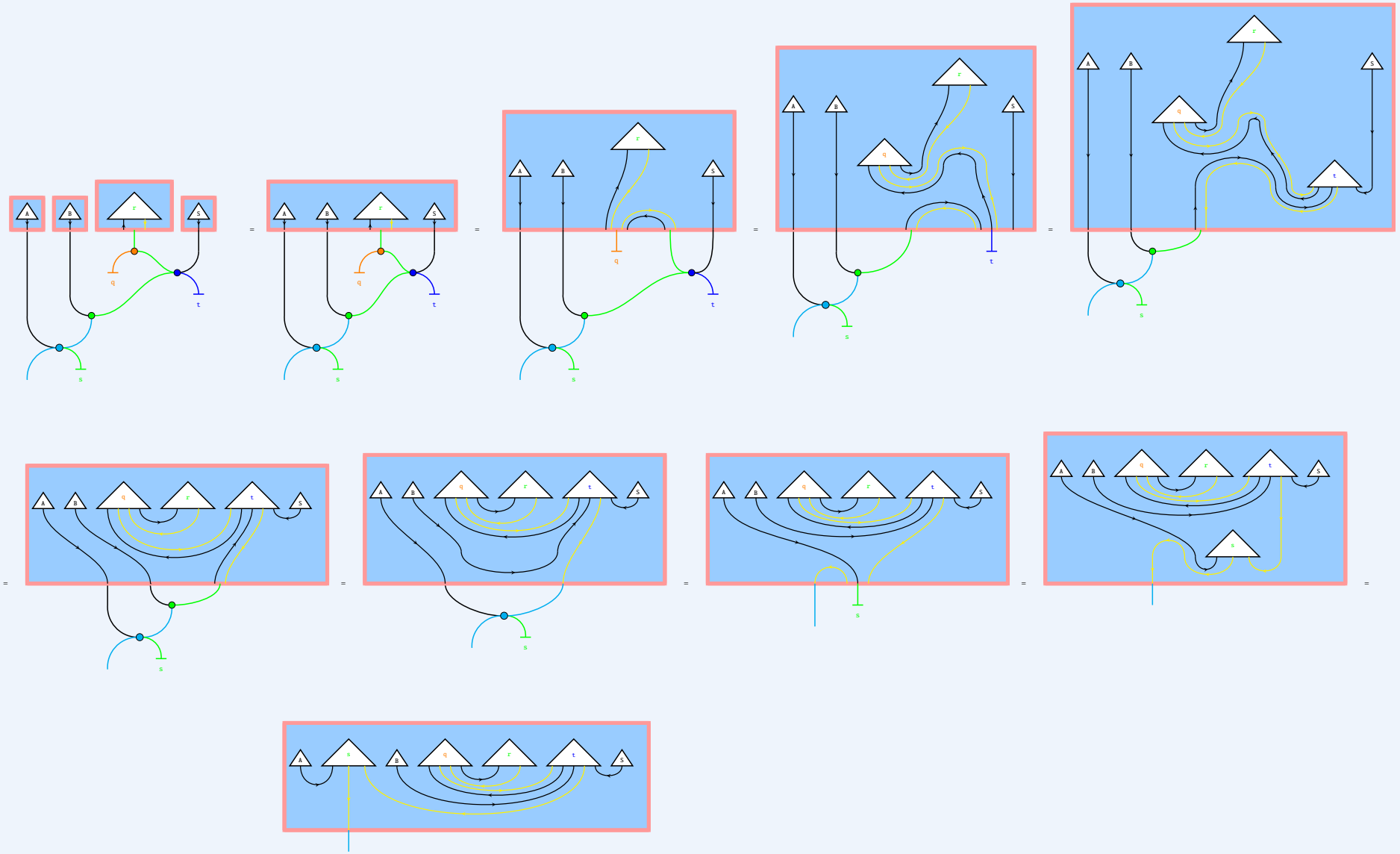


For the first equation, the magenta opfibration box now has already chosen a codomain for the lift, so it can eat the next morphism highlighted in a dashed box. For the second equation, note that eating-rules swap over for the different conventions of functor boxes: while inside-out functor boxes need extra data to eat, outside-in boxes need extra data to spit out, but can eat for free.



So both functor boxes can eat their way through the outside string diagram, and wire-assignment is resolved by the outer functor box keeping track of the current choice of codomain.

And so we can continue all the way, occasionally straightening out some wires on the inside.



0.5 *Monoidal kinda-confunctor boxes*

Now we'd like to use the cofunctor technology we have developed to tackle the other problem, the nonfunctoriality of internal wirings for the listener (Example 12.) We run into a problem again: the right leg cannot be a discrete opfibration into pregroup diagrams.

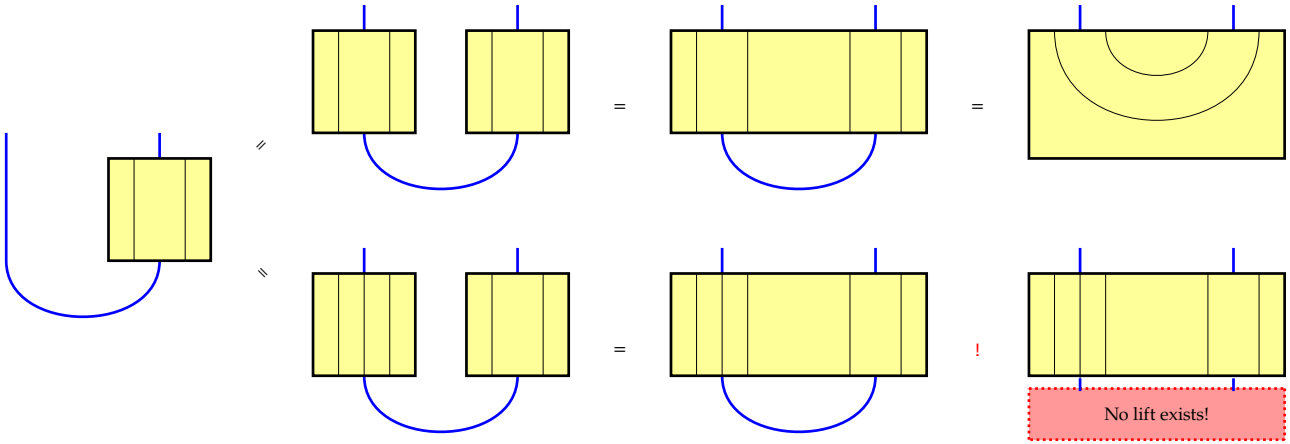


Figure 27:

Example 0.5.1 (Lifts for cups are not always defined)**.**

Starting from the leftmost diagram, in order to let the functor box eat the whole diagram, we need to first choose a lift for the left-sentence wire for the cup. Recalling Figures 6 and 12, there are at least two lifts for the sentence-wire in pregroup diagrams, for the case of two or three noun-wires. Everything works smoothly when the lifts on the two sentence wires of a cup match. When if we make the wrong choice and they don't, there is no lift, because there is no such thing as a cup that has two wires on one end and three on the other. Recall from Definition 0.5.2 that a unique lift is required for *every possible* codomain inside the functor box; so we do not have a discrete opfibration, and so we cannot have a cofunctor.

But hold on, if we know that cups need pairs of matching identity-lifts, in the above example it would have been obvious *from the connectivity of the diagram* what the correct choice of lift ought to have been. In order to reason diagrammatically with hungry functor boxes in the way we'd like, we can weaken the requirement that the right leg of the cofunctor be a discrete opfibration[2]. We don't need lifts to always *exist uniquely* for *all* codomains; that they always exist for some codomains is enough for our functor boxes to eat and merge the way they do. However, now instead of just looking up the lift, we'll get to make decisions in order to help the functor box grow, but as we've seen from our example, we'll be guided by the connectivity of the diagrams.

Definition 0.5.2 (Kinda-opfibration)**.** $\mathbf{F} : C \to \mathcal{D}$ is a *kinda-opfibration*[3] when for all morphisms $f : \mathbf{F}A \to B$ in $\mathcal{D}$ with domain in the image of $\mathbf{F}$, there exists some object Φ_f^A and some $\phi_f : A \to \Phi_f^A$ in C, such that $f = \mathbf{F}\phi_f$.

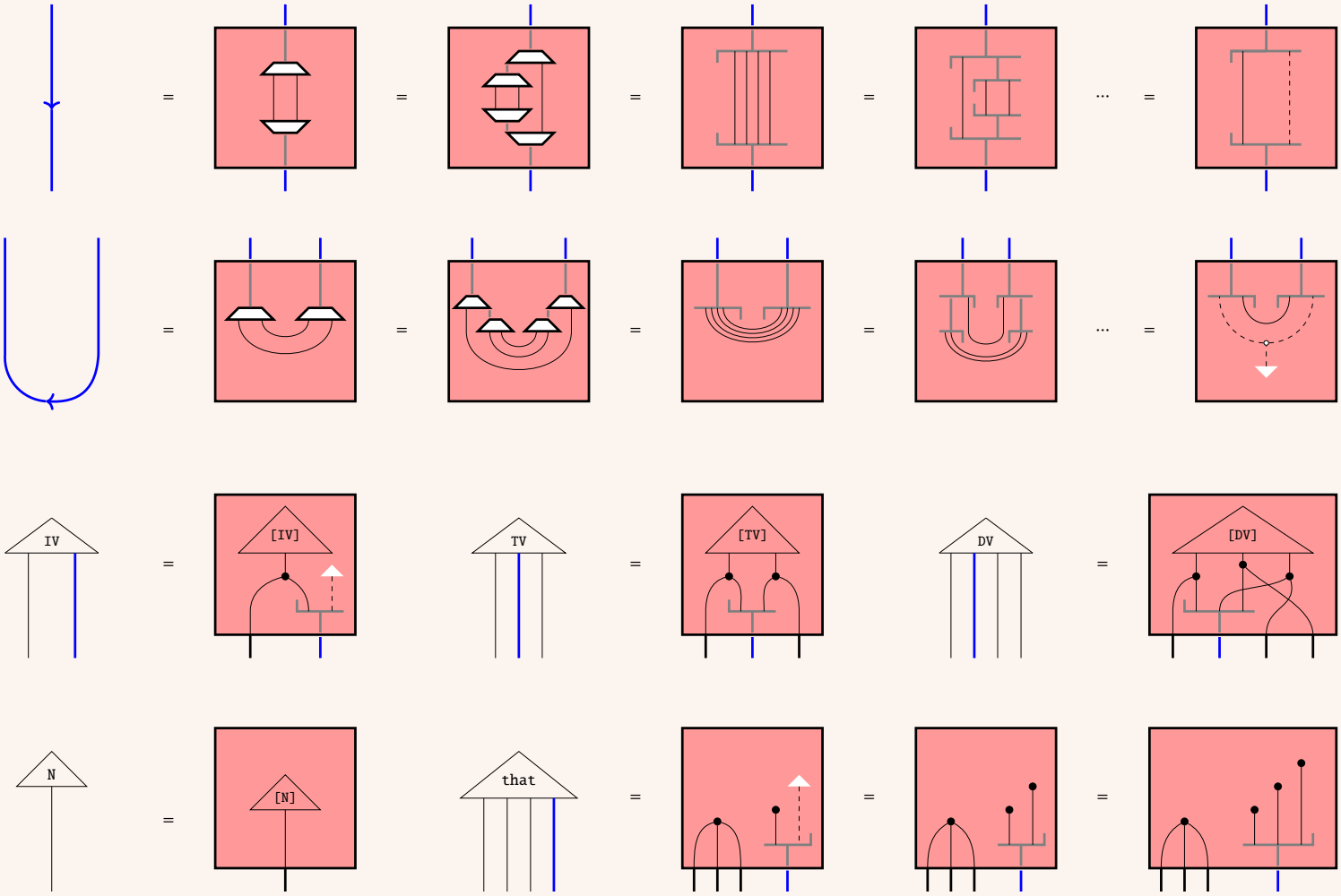
Definition 0.5.3 (Monoidal kinda-opfibration and kinda-cofunctor)**.** Mentally search and replace discrete for kinda- in Definition 22 and Proposition 0.4.4.

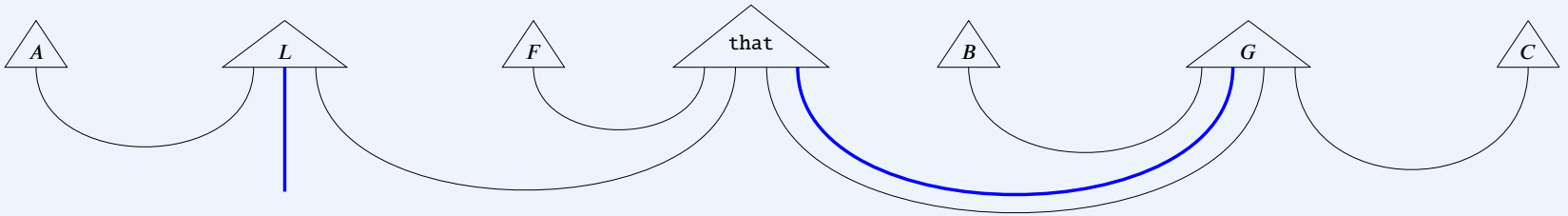
[2] The job we want the right leg to do is just to bookkeep choices of lift in its fibres. Discrete opfibrations do too much by enacting safety standards and curtailing our choice. Nevermind universal properties, we want to make our own decisions, good or bad.

[3] This probably exists somewhere in the literature, though maybe not, since we're getting rid of the universal properties.

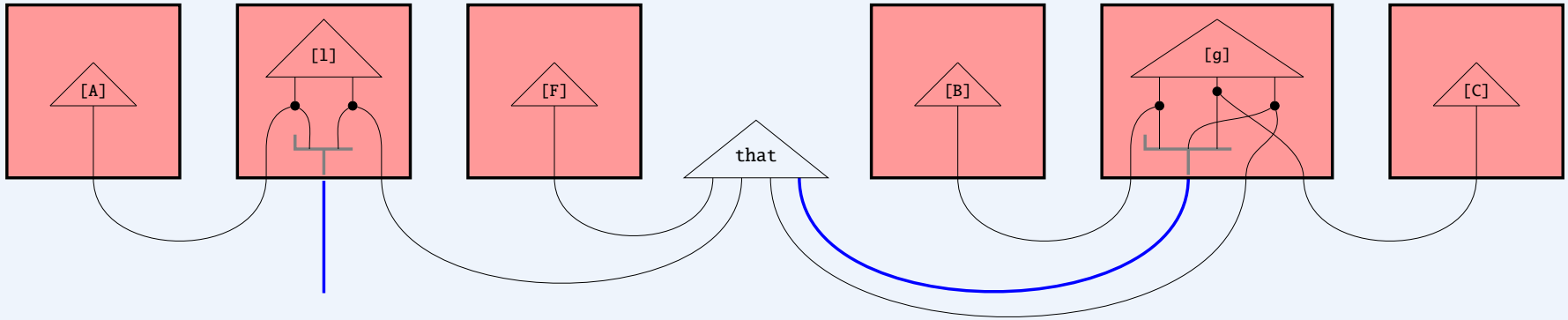
Example 0.5.4 (A monoidal kinda-opfibration into pregroup diagrams from a subcategory of bracketed pregroups with spiders)**.**

Source category is the free strictification of a single wire equipped with a spider, with generator-states implied by the following equations. Target is pregroup diagrams in n, s. Sentence type is bracketings of nouns: alternate bracketing notation is introduced for brevity. Strictified unit is used to distinguish the bracketed single noun wire from the plain noun wire, which is its own lift. Bracketing monoidal units gives the Dyck language, which may be in principle used to distinguish turning numbers in the rigid autonomous setting, omitted for brevity. For more internal wirings for other grammatical categories, see CITE . This data suffices to model how the listener recovers the data conveyed by the speaker, c.f. Example 12. Solution to Example 12 on next page.

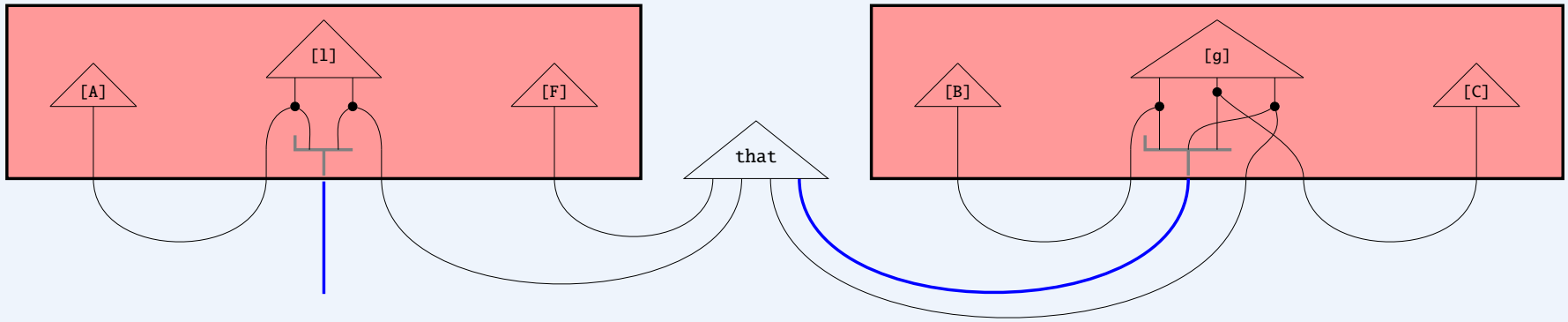




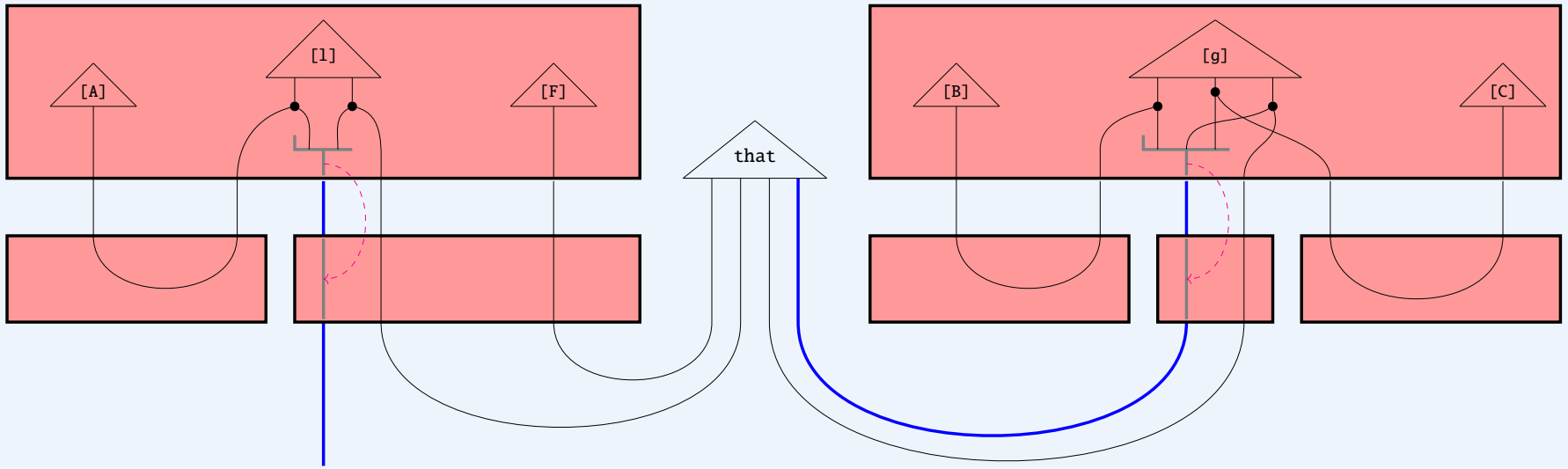
Some lifts are determined.



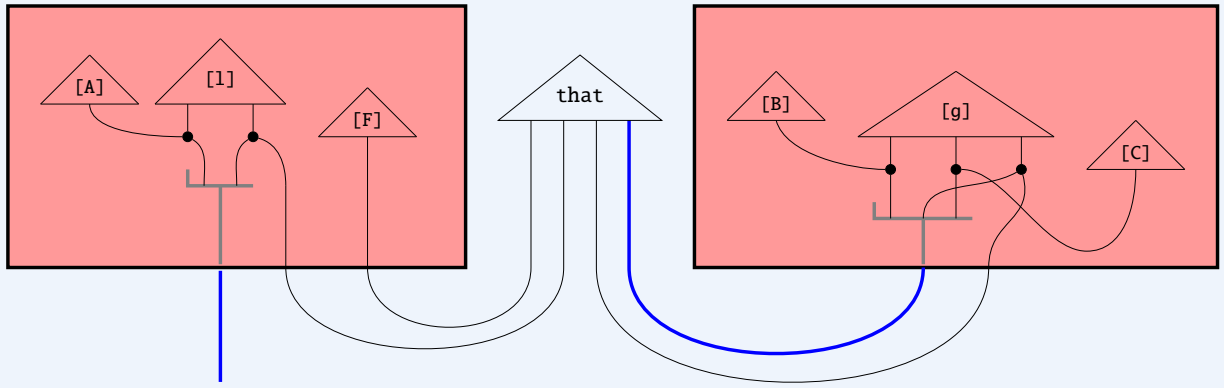
Merge.



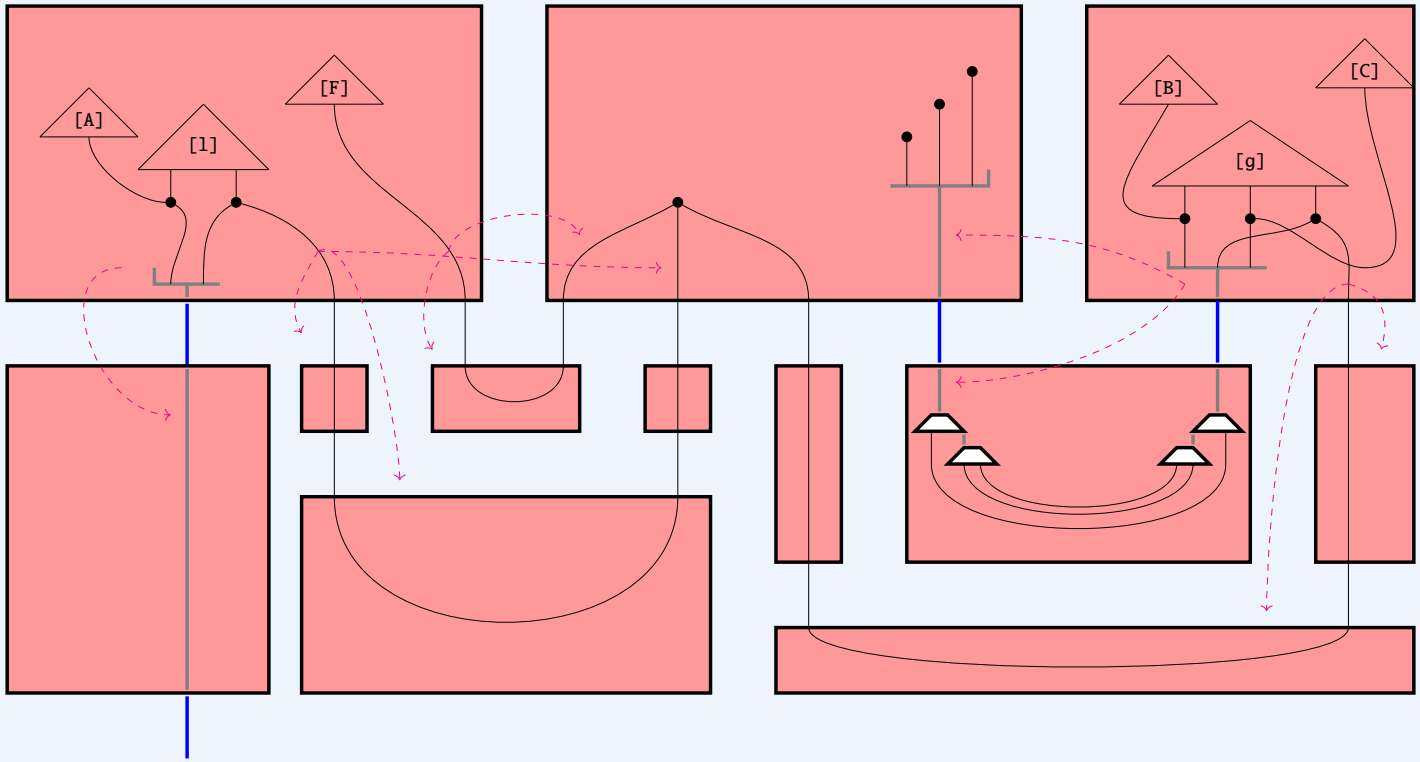
Noun-cups have determined lifts. Typematching lifts inferred from connectivity.



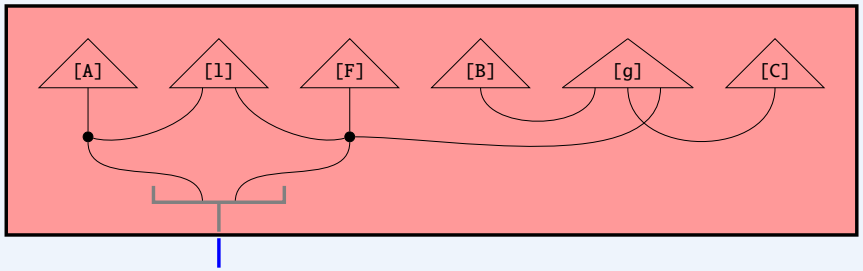
Merge.



Typematching lifts inferred from connectivity, and type-restriction of cups.



Merge. Cancel brackets. Simplify spiders.



0.6 Discussion and Limitations

What are the assumptions and limitations? In my view, the preceding analysis is fair if one entertains the following three commitments.

1. At some level, semantics is compositional, and syntax directs this composition.
2. Speakers produce sentences, and listeners parse sentences.
3. Speakers and listeners understand each other, insofar as the compositional structure of their semantic representations are isomorphic.

Insofar as compositionality entails that infinite ends can be achieved by finite combinatorial means, discrete monoidal fibrations are bookkeeping for an idealised structural correspondence between the components of productive and parsing grammars, and internal wirings arise as balancing terms in the bookkeeping.

The first assumption establishes an idealised view of communication and compositionality where there are no extraneous rules in the language, i.e. that a particular phrase of five or sixty-seven words is to be parsed exceptionally. This is not the case in natural languages, where everyday idioms may be considered semantically atomic despite being compositionally decomposable. For example, in Mandarin, 马上 is the concatenation of "horse" and "up", and would be "on horseback" if interpreted literally, but is treated as an adverbial "as soon as possible" in imperative contexts. I use the hedging phrase "at some level" in the first assumption to describe the compositionality of semantics just to indicate an assumption that we are not dealing with exceptional rules all the way up.

The second assumption commits to an idealisation that speakers and listeners communicate for the purposes of exchanging propositional information as well-formed and disambiguated sentences, which is clearly not all that language is for. I can promise nothing regarding questions, imperatives, speech acts, and so on.

The third assumption asks that one entertain string diagrams as representative of what the content of language is, and even so, it still requires some elaboration on what is meant by "understanding", as it is obviously untrue that everyone understands one another. I do not mean understanding in the strong sense as a form of telepathy of mental states, c.f. Wittgenstein's beetle-in-a-box thought experiment. I mean that insofar as the speaker and listener both have their own ideas about cats, sitting, space, and mats, their respective mental models of `the cat sat on the mat` are indistinguishable as far as meaningfully equivalent syntactic re-presentations and probings go; for instance, both speakers ought to agree that `the mat is beneath the cat`, and both speakers ought to agree despite the concrete images in their minds that there is insufficient information to know the colour of the cat from the sentence alone, and so on. This is a shallow form of understanding; consider the case where one communicator is a human with mental models encoded in meat and another is an LLM with tokens encoding who-knows-what – they may be in perfect agreement about

rephrasings of texts for an arbitrary finite amount of communication, even if the representations of the latter are not compositional. It would be nice to ask that "mutual understanding" requires structurally equivalent (as opposed to extensionally indistinguishable) meaning-representation mechanisms between language agents c.f. Chomsky's universal grammar, but our means of achieving mutual understanding in practice seems to align with the shallow view: we pose comprehension challenges and ask clarifying questions all at the level of language, without taking a scalpel to the other's head. Depending on one's view of what understanding language entails, it may be that humans and LLMs both understand language in their own way, but mutual understanding between the two kinds is an illusion.

Despite these limitations, I believe that this formal approach to grounding relationships between productive and parsing grammars in mathematical considerations surrounding communication has some merits.

THEORIES OF GRAMMAR BY THEMSELVES ARE INSUFFICIENT TO ACCOUNT FOR COMMUNICATION. At minimum, for every grammar that produces sentences, one must also provide a corresponding parsing grammar. A theory of grammar that only produces correct sentences or correct parses is a 'theory' of language outperformed in every respect by an LLM. So we must distinguish between grammars of the speaker and listener, and then investigate how they cohere.

COHERENCE OF THEORIES OF GRAMMAR IS INEXTRICABLE FROM SEMANTICS. We are interested in the ideal of communication, the end result of a single turn of which is that both speaker and listener have the same semantic information, whether that be a logical expression or something else. A consequence of this criterion is that in order to obtain an adequate account of communication, we must seek a relation between grammars and semantics beyond weak and strong equivalence of pairs of theories of grammar.

Firstly, "weak equivalence" between grammar formalisms in terms of possible sets of generated sentences is insufficient. Weak equivalence proofs are mathematical busywork that have nothing to do with a unified account of syntax and semantics. For example, merely demonstrating that, e.g. pregroup grammars and context-free grammars can generate the same sentences [] only admits the possibility that a speaker using a context-free grammar and a listener using a pregroup grammar *could* understand each other, without providing any explanation *how*. But we already know that users of language *do* understand one another more-or-less, so the exercise is more-or-less pointless.

Secondly, "strong equivalence" that seeks equivalence at a structural level between theories of syntax often helps, but is not always necessary. I will explain by analogy. Theories of syntax are like file formats, e.g. `.png` or `.jpeg` for images. A model for a particular language is a particular file or photograph. The task here is to show that two photographs in different file formats that both purport to model the same language are really photographs of the same thing from different perspectives. It is overkill to demonstrate that all `.pngs` and `.jpegs` are structurally bijectable, just as it is overkill to show that, say, context-free grammars are strongly equivalent to pregroup grammars, because there are context-free and pregroup grammars that generate sets

of strings that have nothing to do with natural language. It could just as well be that there is a pair of productive and parsing grammar-formats that are not strongly equivalent, but happen to coincide for a particular natural language – in this sense, asking for a discrete monoidal fibration is a way to check a weaker condition than strong equivalence that achieves the more specific aim of determining whether a pair of productive and parsing grammars for a language are plausible models.

A systematic analysis of communication requires intimacy with specific grammars and a specific semantics. Specific grammars – and not formats of grammar, such as "all CFGs" – that model natural languages, even poorly, are the only relevant objects of study for any form of language intended to communicate information. Once you have a specific grammar that produces sentences in natural language, then to explain communication, you must supply a specific partnered parsing grammar such that on the produced sentences, both grammars yield the same semantic objects by a Montagovian approach, broadly construed as a homomorphism from syntax to semantics. On this account, syntax does not hold a dictatorship over semantics, but we can find duarchies, and in these duumvirates the two syntaxes and the semantics mutually constrain one another.

It is worth noting that in practice, neither grammar nor meaning strictly determines the other. Clearly there are cases where grammar supercedes: when Dennis hears `man bites dog`, despite his prior prejudices and associations about which animal is more likely to be biting, he knows that the `man` is doing the biting and the `dog` is getting bitten. Going the other way, there are many cases in which the meaning of a subphrase affects grammatical acceptability and structure.

Example 0.6.1 (Exclamations: how meaning affects grammar)**.** The following examples from [Lakofflecture] illustrate how whether a phrase is an *exclamation* affects what kinds of grammatical constructions are acceptable. By this argument, to know whether something is an exclamation in context is an aspect of meaning, so we have cases where meaning determines grammar. Observe first that the following three phrases are all grammatically acceptable and mean the same thing.

`nobody knows how many beers Bob drinks`

`who knows how many beers Bob drinks`

`God knows how many beers Bob drinks`

The latter two are distinguished when `God knows` and `who knows` are exclamations. First, the modularity of grammar and meaning may not match when an exclamation is involved. For example, negating the blue text, we obtain:

`somebody knows how many beers Bob drinks`

`who doesn't know how many beers Bob drinks`

God doesn't know how many beers Bob drinks

The first two are acceptable, but mean different things; the latter means to say that everyone knows how many beers Bob drinks, which is stronger than the former. The last sentence is awkward: unlike in the first two cases, the quantified variable in the (gloss) $\cdots \neg \exists x_{Person} \cdots$ of God knows is lost, and what is left is a literal reading $\cdots \neg \texttt{knows}(\texttt{God}, \cdots) \cdots$. Second, whether a sentence is grammatically acceptable may depend on whether an exclamation is involved. God knows and who knows can be shuffled into the sentence to behave as an intensifier as in:

Bob drinks God knows how many beers

Bob drinks who knows how many beers

But it is awkward to have:

Bob drank nobody knows how many beers

And it is not acceptable to have:

Bob drank Alice knows how many beers

Example 0.6.2 (Garden path sentences)**.** So-called "garden path" sentences illustrate that listeners have to make choices to resolve lexical ambiguities. One such garden-path sentence is The old man the boat, where typically readers take The old man as a noun-phrase and the boat as another noun-phrase. We can sketch how the readers might think with a (failed) pregroup grammar derivation:

$$\dfrac{\dfrac{\dfrac{\texttt{the} : n \cdot n^{-1} \quad \texttt{old} : n \cdot n^{-1}}{\texttt{the␣old} : n \cdot n^{-1}} \quad \texttt{man} : n}{\texttt{the␣old␣man} : n} \quad \dfrac{\texttt{the} : n \cdot n^{-1} \quad \texttt{boat} : n}{\texttt{the␣boat} : n}}{\text{Not a sentence!}}$$

So the reader has to backtrack, taking The old as a noun-phrase and man as the transitive verb. This yields a sentence as follows:

$$\dfrac{\dfrac{\dfrac{\texttt{the} : n \cdot n^{-1} \quad \texttt{old} : n}{\texttt{the␣old} : n} \quad \texttt{man} :^{-1} n \cdot s \cdot n^{-1}}{\texttt{the␣old␣man} : s \cdot n^{-1}} \quad \dfrac{\texttt{the} : n \cdot n^{-1} \quad \texttt{boat} : n}{\texttt{the␣old} : n}}{\texttt{the␣old␣man␣the␣boat␣} : s}$$

Garden-path sentences illustrate that listeners must make choices about what grammatical roles to assign words. We make these kinds of contextual decisions all the time with lexically ambiguous words or highly homophonic languages like Mandarin; garden-path sentences are special in that they trick the default strategy badly enough that the mental effort for correction is noticeable.

Example 0.6.3 (Ambiguous scoping)**.** Consider the following sentence:

$$\texttt{Everyone loves someone}$$

The sentence is secretly (at least) two, corresponding to two possible parses. The usual reading is (glossed) $\forall x \exists y : \texttt{loves}(x, y)$. The odd reading is $\exists y \forall x : \texttt{loves}(x, y)$: a situation where there is a single person loved by everyone. We can sketch this difference formally using a simple combinatory categorial grammar.

$$\dfrac{\dfrac{\texttt{everyone} : (n \multimap s) \multimap s \qquad \texttt{loves} : n \multimap s \circ\!\!- n}{\texttt{everyone␣loves} : s \circ\!\!- n} \qquad \texttt{someone} : (s \circ\!\!- n) \multimap s}{\texttt{everyone␣loves␣someone} : s}$$

$$\dfrac{\texttt{everyone} : (n \multimap s) \multimap s \qquad \dfrac{\texttt{loves} : n \multimap s \circ\!\!- n \qquad \texttt{someone} : (s \circ\!\!- n) \multimap s}{\texttt{loves␣someone} : n \multimap s}}{\texttt{everyone␣loves␣someone} : s}$$

CCGs have functorial semantics in any cartesian closed category, such as one where morphisms are terms in the lambda calculus and composition is substitution. So we might specify a semantics as follows:

$$[\![\texttt{everyone}]\!] = \lambda(\lambda x.V(x)).\forall x : V(x) \tag{1}$$

$$[\![\texttt{loves}]\!] = \lambda x \lambda y.\texttt{loves}(x, y) \tag{2}$$

$$[\![\texttt{someone}]\!] = \lambda(\lambda y.V(y)).\exists y : V(y) \tag{3}$$

Now we can plug-in these interpretations to obtain the two different meanings. We decorate with corners just to visually distinguish which bits are partial first-order logic.

$$\dfrac{\dfrac{\lambda(\lambda x.V(x)).\ulcorner\forall x : V(x)\urcorner : (n \multimap s) \multimap s \qquad \lambda x \lambda y.\ulcorner\texttt{loves}(x, y)\urcorner : n \multimap s \circ\!\!- n}{\lambda y.\ulcorner\forall x : \texttt{loves}(x, y)\urcorner : s \circ\!\!- n} \qquad \lambda(\lambda y.V(y)).\ulcorner\exists y : V(y)\urcorner : (s \circ\!\!- n) \multimap s}{\ulcorner\exists y \forall x : \texttt{loves}(x, y)\urcorner : s}$$

$$\dfrac{\lambda(\lambda x.V(x)).\ulcorner\forall x : V(x)\urcorner : (n \multimap s) \multimap s \qquad \dfrac{\lambda x \lambda y.\ulcorner\texttt{loves}(x, y)\urcorner : n \multimap s \circ\!\!- n \qquad \lambda(\lambda y.V(y)).\ulcorner\exists y : V(y)\urcorner : (s \circ\!\!- n) \multimap s}{\lambda x.\ulcorner\exists y : \texttt{loves}(x, y)\urcorner : n \multimap s}}{\ulcorner\forall x \exists y : \texttt{loves}(x, y)\urcorner : s}$$

Example 0.6.4. (grammar pieces – as simple as it gets) `Bob runs. Bob quickly runs. Bob drinks beer. Bob quickly drinks beer.`