

VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

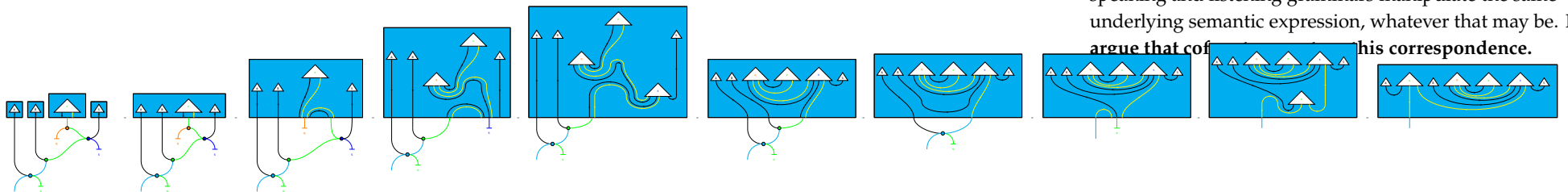
0	Where do internal wirings come from?	5	
0.1	How do we communicate using language?	5	
0.2	Discrete Monoidal Opfibrations	12	
0.2.1	What are they good for?	17	
0.3	Strictified diagrams for monoidal categories	19	
0.4	Monoidal cofunctor boxes	24	
0.4.1	From speaker to listener	25	
0.4.2	From listener to speaker	28	
0.5	Discussion and Limitations	28	(Acknowledgements will go in a margin note here.)

0

Where do internal wirings come from?

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 like Alice sees Bob quickly run to school. 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.



There is a distinction between *grammars of the speaker* – which produce sentences – and *grammars of the listener* – which deduce sentences. Viewed as mathematical processes, the two kinds of grammars go in opposite directions; speaking grammars (e.g. string-rewrite systems) start with some grammatical structure, and require informational input (e.g. which rule comes next) to produce lists of words – sentences. Conversely, listening grammars (e.g. typelogical grammars) start with a sentence, and require informational input (e.g. grammatical typing and which proof rule to try next) to deduce or parse a grammatical structure. Since we can understand each other, these two types of grammar must enjoy a systematic correspondence, and if one believes that semantics is compositional according to syntax, then the correspondence must further explain how both speaking and listening grammars manipulate the same underlying semantic expression, whatever that may be. I argue that this correspondence.

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 typological 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.

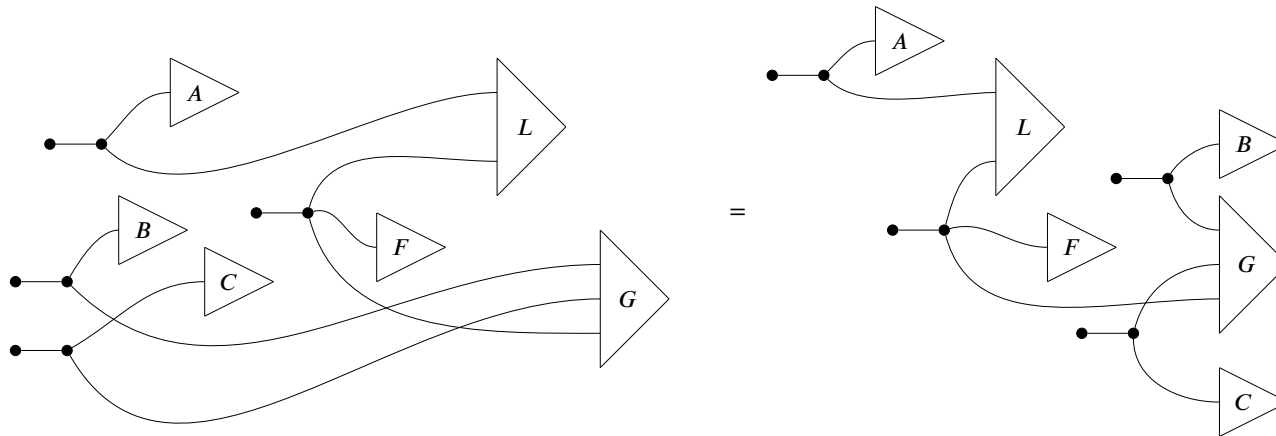


Figure 1: Charlie and Dennis agree on the conceptual organisation entities and relations up to the words for those entities and relations. Just as a running example that does not affect the point, let's say we can gloss a thought in first order logic as $\exists a \exists b \exists c \exists f : A(a) \wedge B(b) \wedge C(c) \wedge F(f) \wedge L(a, f) \wedge G(b, c, f)$. 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.

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.

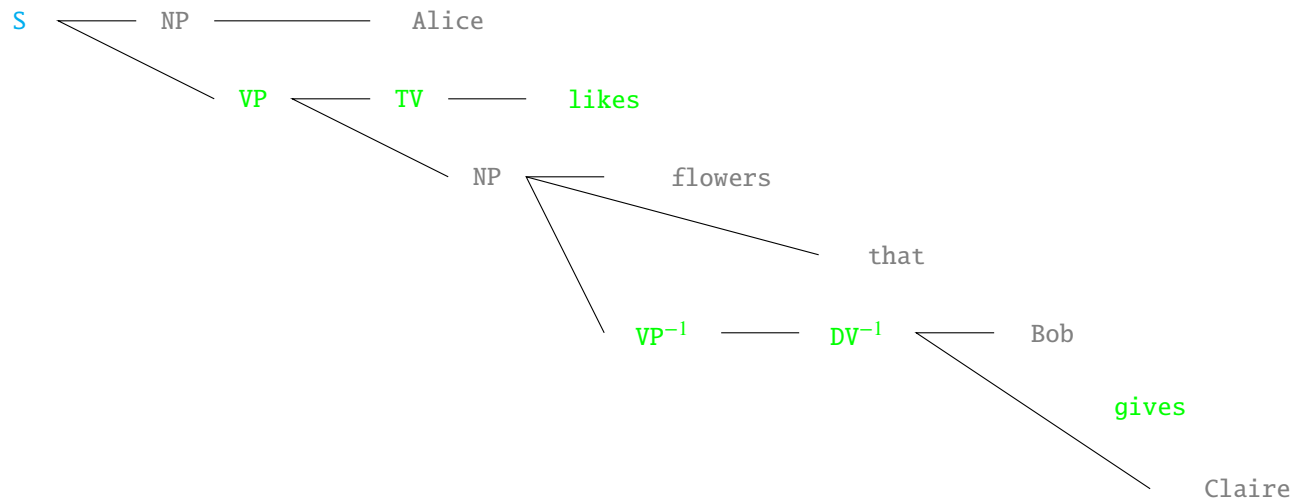


Figure 2: The rule of the game is that Charlie and Dennis can agree on a string-diagrammatic encoding strategy before having to communicate with each other. Here is one such strategy. Charlie might generate the example sentence as depicted.

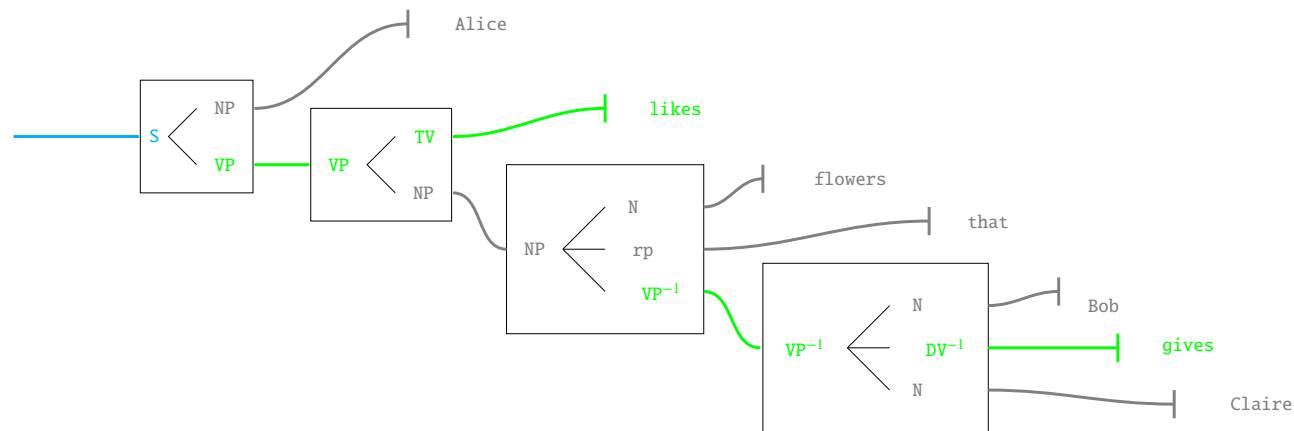


Figure 3: Mathematically, it makes no difference if we take the Poincaré dual of the tree, so that zero-dimensional nodes become one-dimensional wires, and branchings become zero-dimensional points linking wires – but we can just as well depict those points as boxes to label them more clearly.

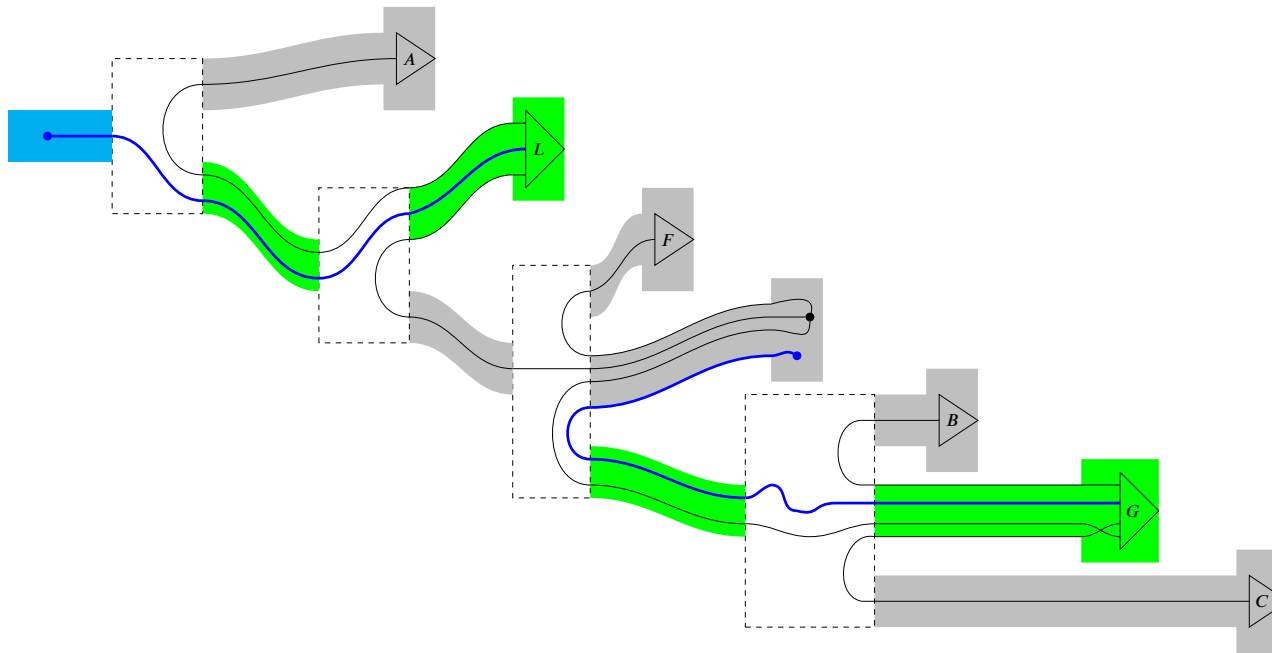


Figure 4: Now that Charlie can express their grammatical structure string-diagrammatically, they can try to deform their first-order-logic diagram – representing what they mean to communicate – subject to the constraint that every one of their branchings (the structure of the CFG) is something recoverable 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.

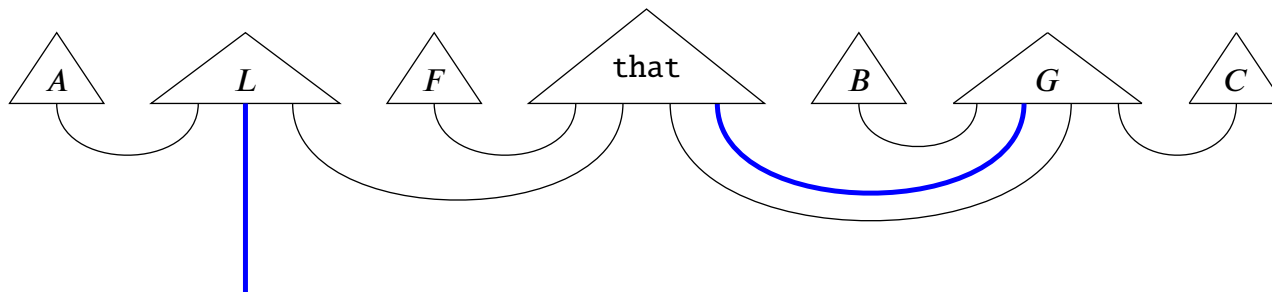


Figure 5: So, when Dennis receives the sentence, Dennis’s pregroup derivation yields a pregroup diagram that is connectively equivalent to what Charlie stuffed inside the context-free grammar structure. So now the two have strong equivalence between their grammars in the sense that every one of Charlie’s branches is resolved by one of Dennis’s reductions.

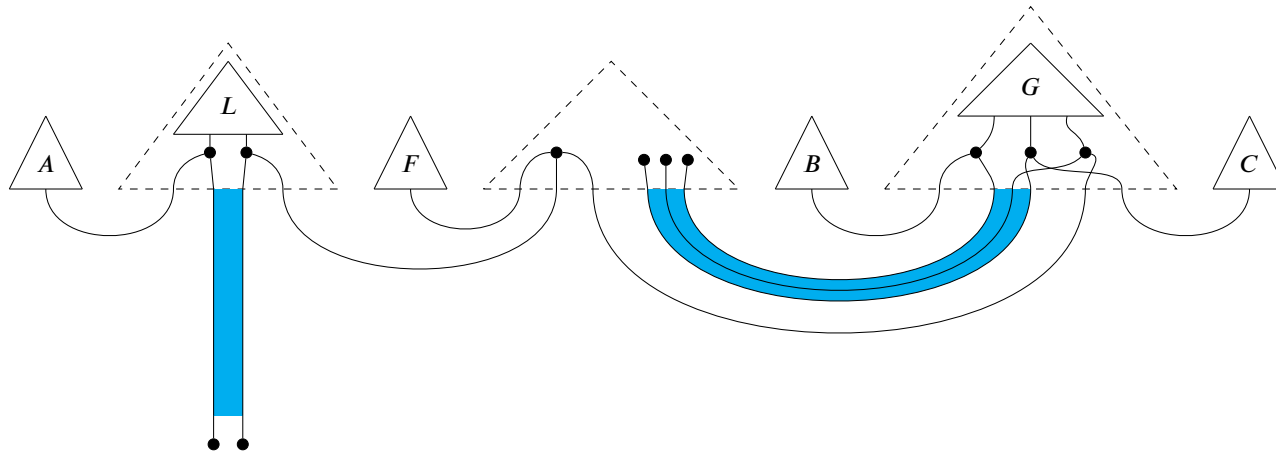


Figure 6: Now to fully recover Charlie's intended FOL-diagram, Dennis refers to the internal wirings from their shared strategy, and fills those in.

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.

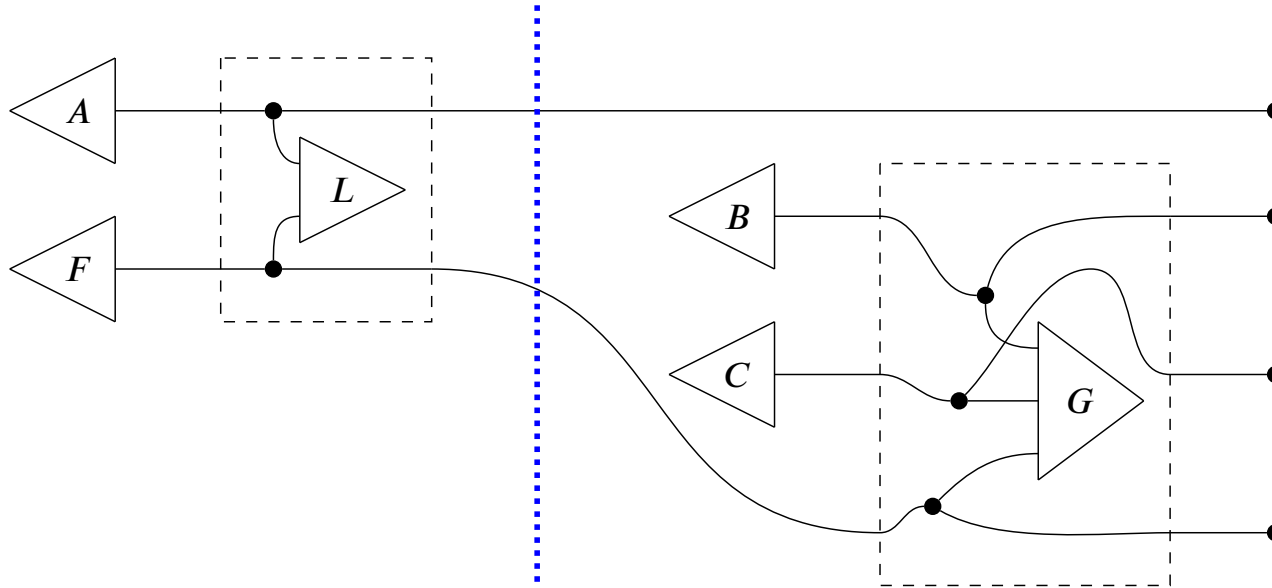


Figure 7: Charlie's diagram morphed to fit a text circuit. The dotted blue line is a formal mark to indicate a sentential boundary. Observe how new discourse elements are introduced as states, and how open wires correspond to ongoing discourse and deletions mark completed discourse. This diagram also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

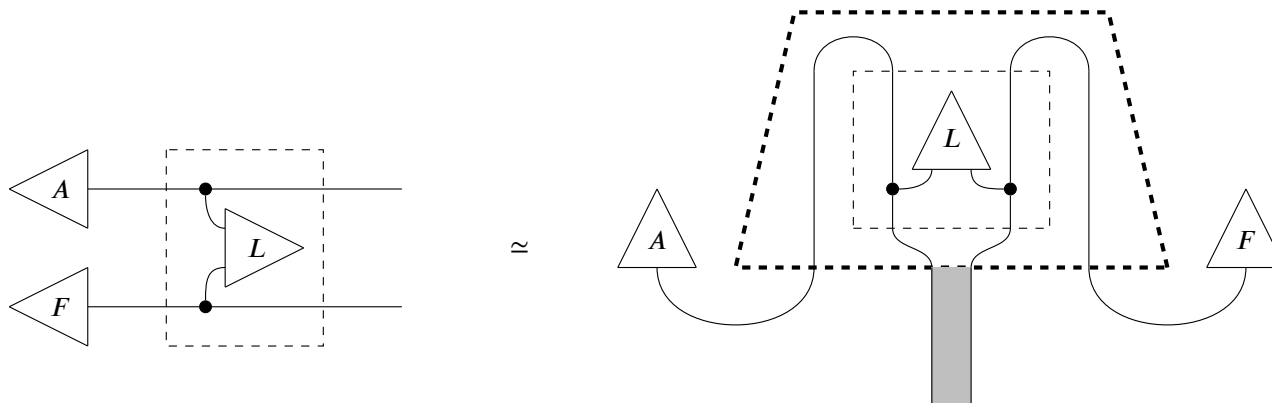


Figure 8: Dennis already knows how to parse individual sentences to extract the FOL using internal wirings. Observe there is a mathematical complication that arises in determining how many noun-wires should go into the sentence wire-bundle; we need to account for this later.

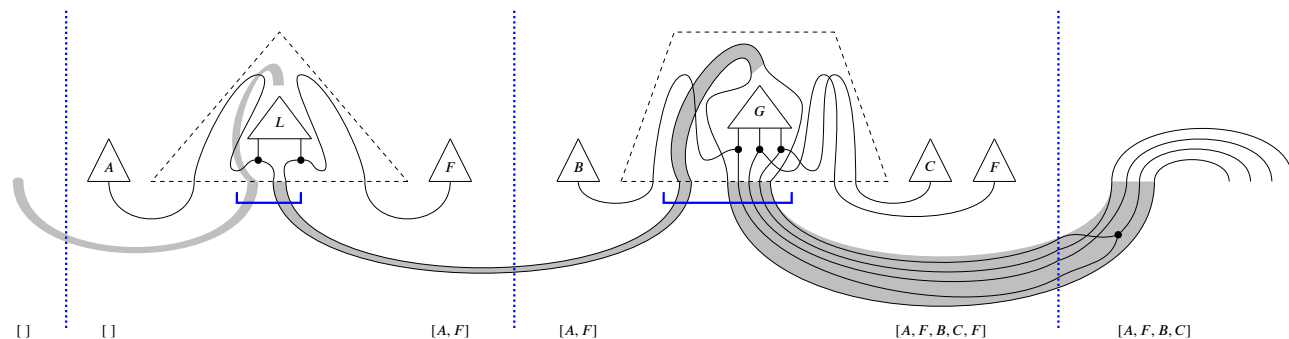


Figure 9: To deal with text, Dennis can pass a growing bundle of sentence wires along horizontally.

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 **CITE** 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.

BUT HOW DO INTERNAL WIRINGS COME ABOUT? If we take internal wirings to be somehow real, then we can wave away the problem by saying that they are obtained by trial and error in the course of the development of language. But purely mathematical mysteries remain. How do we characterise the dynamics of communication between producer and parser? What is the relationship of internal wirings to these dynamics? What is the general idea? Here is a first attempt at an answer: monoidal cofunctors relate parsing and productive grammars, and internal wirings make it so that going round-trip on either side through these cofunctors gets you back where you started up to diagrammatic equivalence. Now what the hell is a monoidal cofunctor? Bear with me.

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 fibration¹: 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 10: 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.

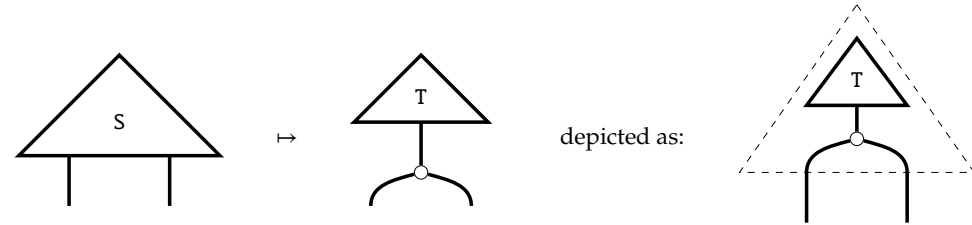


Figure 11: The other convention, following [CITE](#), is *inside-out*. For the following section, we will define the coherence conditions of discrete monoidal fibrations using this convention.

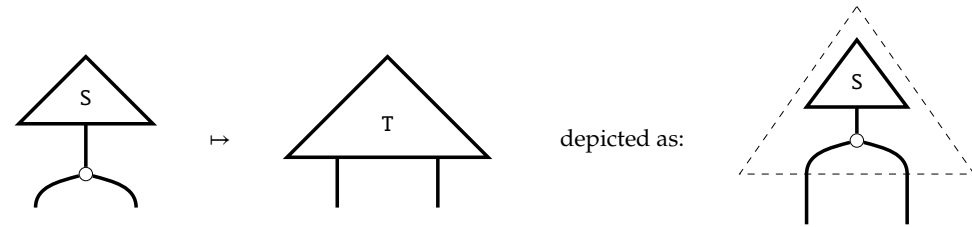


Figure 12: Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$. Then we have this diagrammatic representation of a morphism $\mathbf{F}A \xrightarrow{\mathbf{F}f} \mathbf{F}B$ in \mathcal{D} .

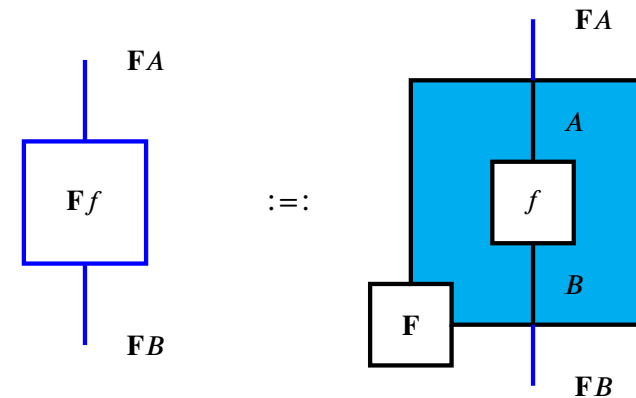


Figure 13: The use of a functor box is like a window from the target category \mathcal{D} into the source category \mathcal{C} ; when we know that a morphism in \mathcal{D} is the image under \mathbf{F} of some morphism in \mathcal{C} , the functor box notation is just a way of presenting all of that 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.

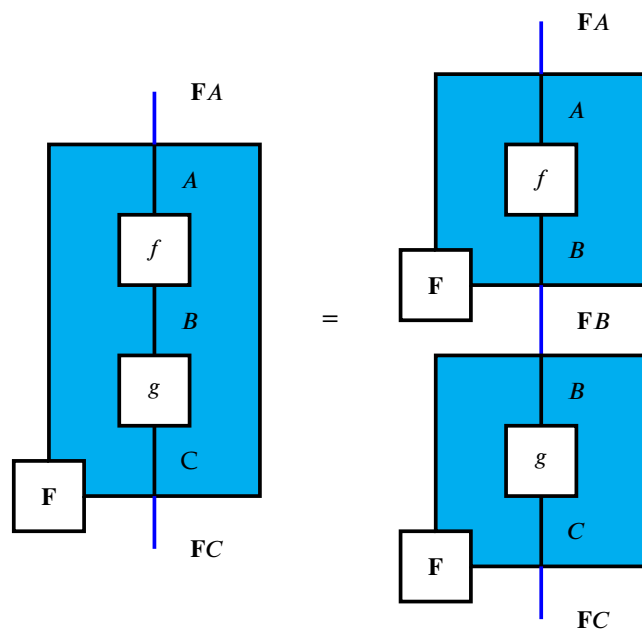


Figure 14: Assume that \mathbf{F} is strict monoidal; without loss of generality by the strictification theorem [CITE](#), this lets us gloss over the associators and unitors. For \mathbf{F} to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A \otimes_D \mathbf{F}B = \mathbf{F}(A \otimes_C B)$. Diagrammatically these structural constraints amount to these equations.

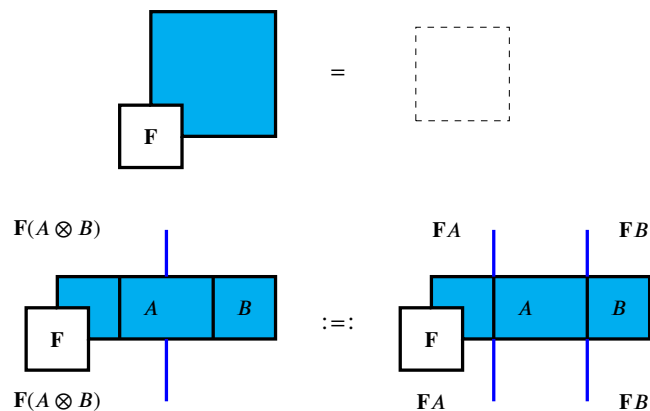


Figure 15: What remains is the monoidality of \mathbf{F} , which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes' way of respecting sequential composition, splitting horizontally is how they respect parallel composition.

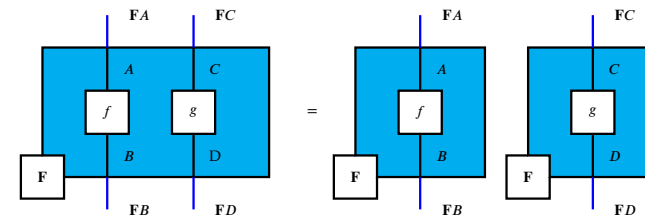


Figure 16: And for when we want \mathbf{F} to be a (strict) symmetric monoidal functor, we are just asking that boxes and twists do not get stuck on one another.

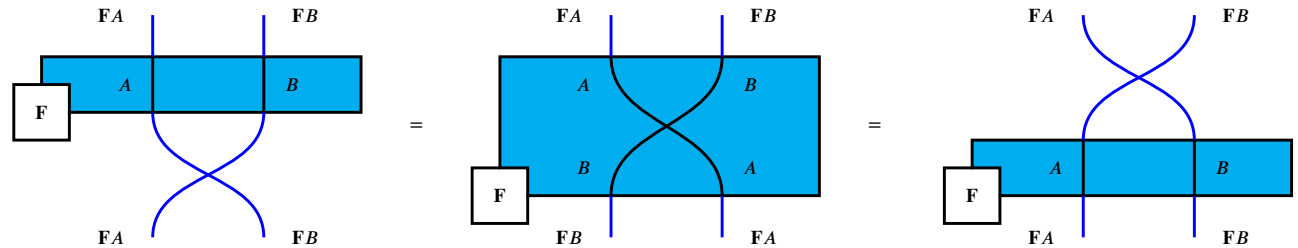


Figure 17: To motivate fibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a functor box out of the bottom. When can we do the reverse? That is, take a morphism in \mathcal{D} and *slide it into* 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.

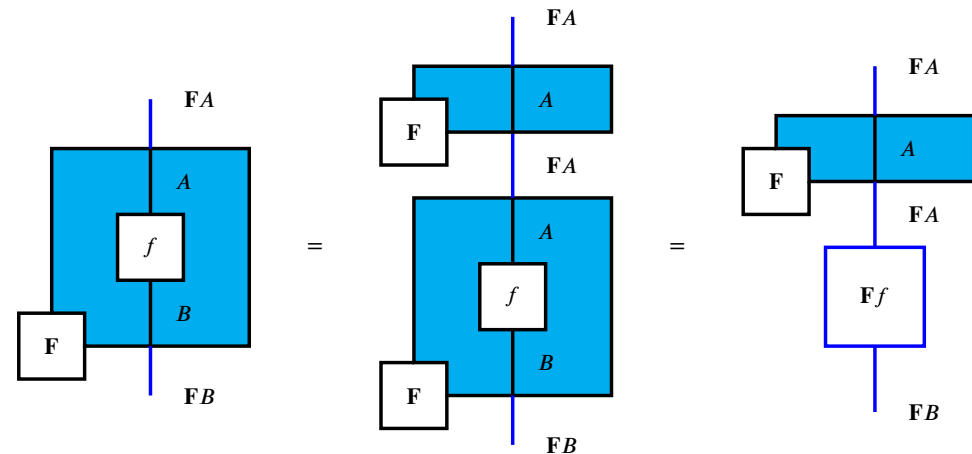


Figure 18:

Definition 0.2.1 (Discrete opfibration). $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$ is a *discrete fibration* when: for all morphisms $f : \mathbf{F}A \rightarrow B$ in \mathcal{D} with domain in the image of \mathbf{F} , there exists a unique object Φ_f^A and a unique morphism $\phi_f : A \rightarrow \Phi_f^A$ 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.

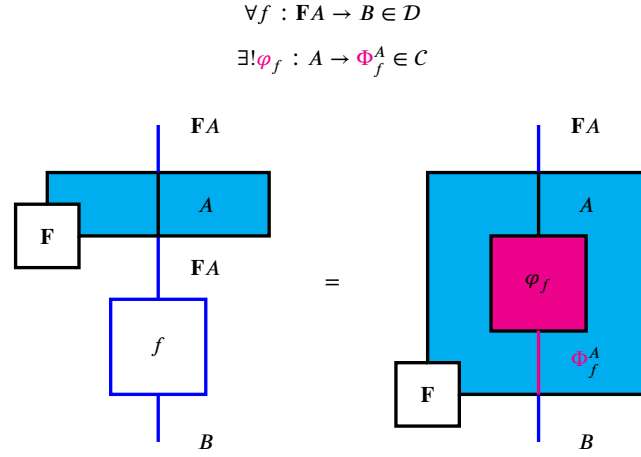
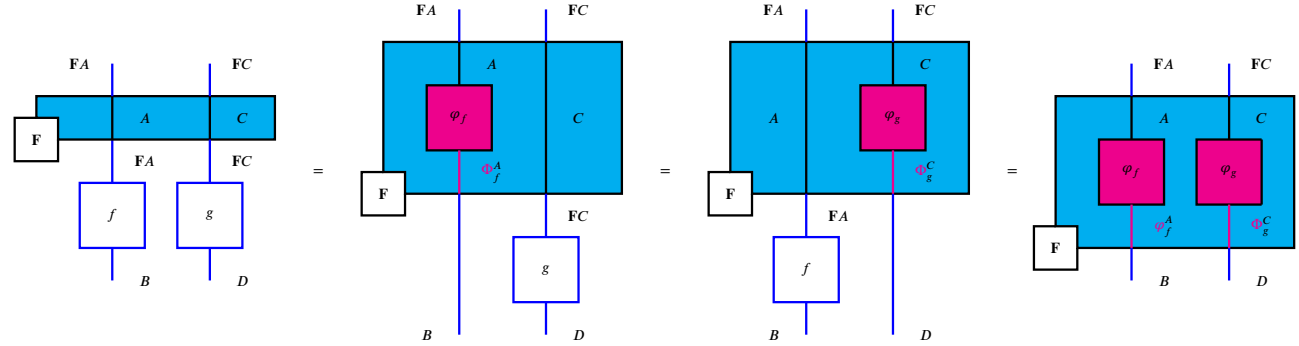


Figure 19:

Definition 0.2.2 (Monoidal discrete opfibration). We consider \mathbf{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?

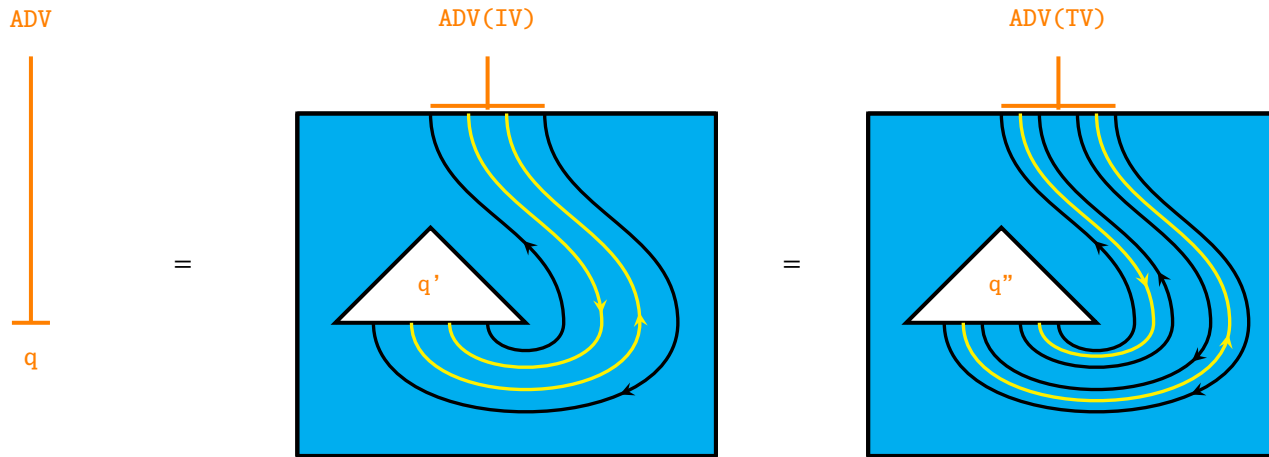


Figure 20: Monoidal discrete opfibration boxes decide which lift to depict if you decide what wires are involved, which is very useful. For example, *quickly* could find itself modifying an intransitive (single noun) or transitive (two noun) verb. Suppose that in a pre-group diagram, it is the job of some process q' to handle intransitive verbs, and similarly q'' to handle transitive ones. We can use the functor for bookkeeping, by asking it to send both q' and q'' to the dependent label \bar{q} . Treating the label as a test rather than a state allows the fibration-box to choose the right version based on the domain wires as it expands top-down.

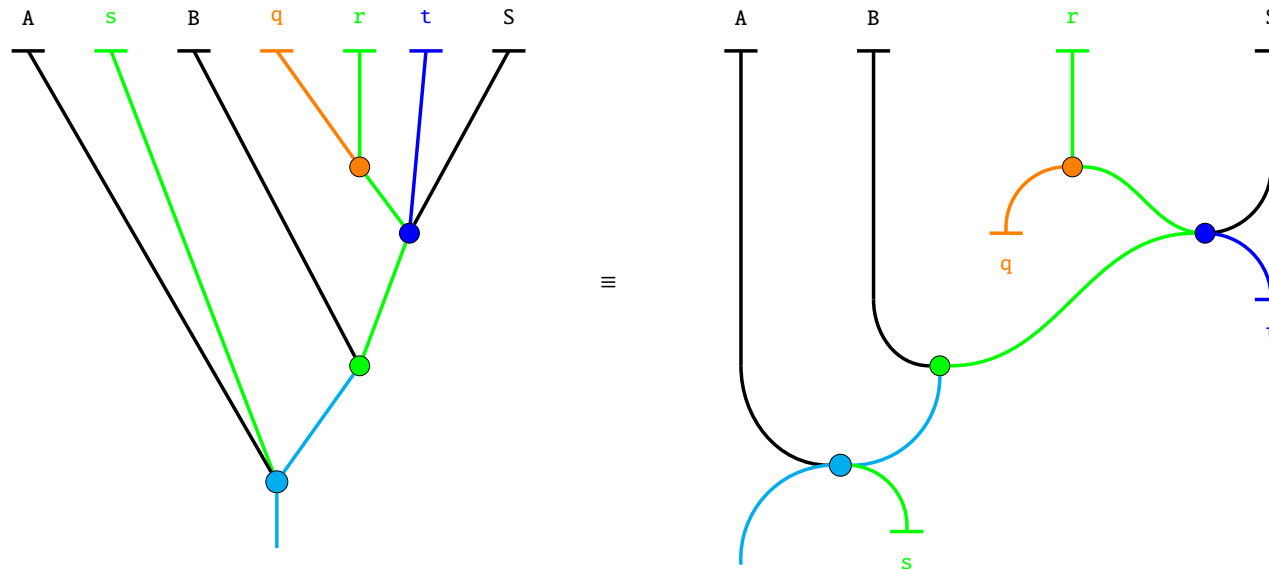


Figure 21: In our example, all grammatical categories other than nouns and basic verbs have multiple instantiations depending on what wires they have to work with. Because we want to treat these dependent types as tests so that the opfibration can work for us, we have to flip them over in the CFG to obtain a *dependent CFG*; in this case, since CFGs are planar, there's no confusion, and we can always flip the labels back over.

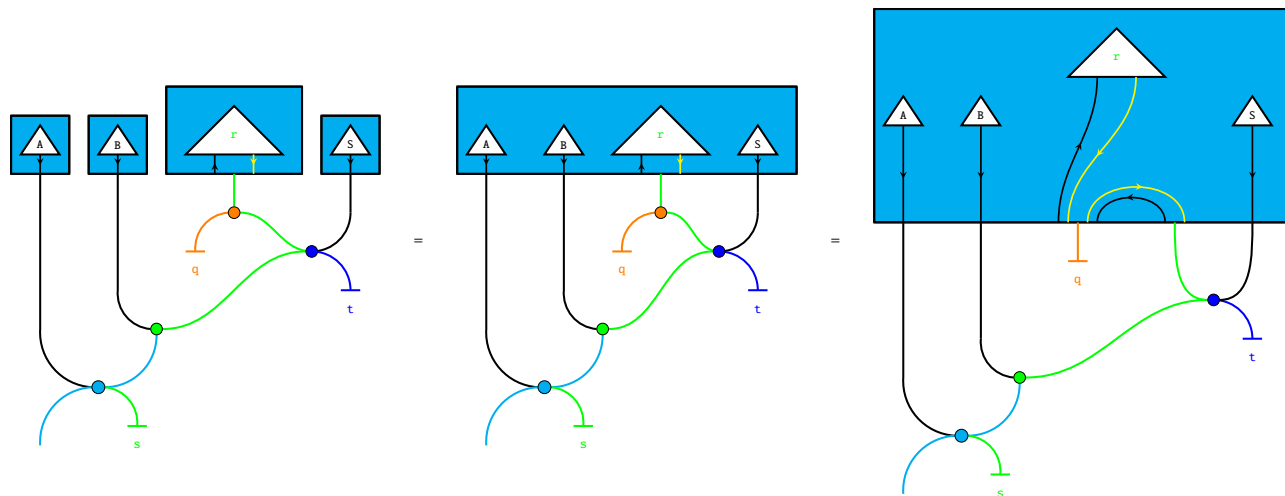


Figure 22: We aim to be able to use discrete monoidal functor-boxes like so. In the leftmost diagram, we would like to graphically introduce pregroup states at the top of a modified context-free grammar via an of-fibration box. In the first equation (isomorphism), we would like to use the monoidal condition of the functor to horizontally merge functor boxes. In the second equation, we would like to use the discrete fibration condition of the functor to expand the box downwards, converting a string diagram obtained from a context-free grammar into a pregroup diagram.

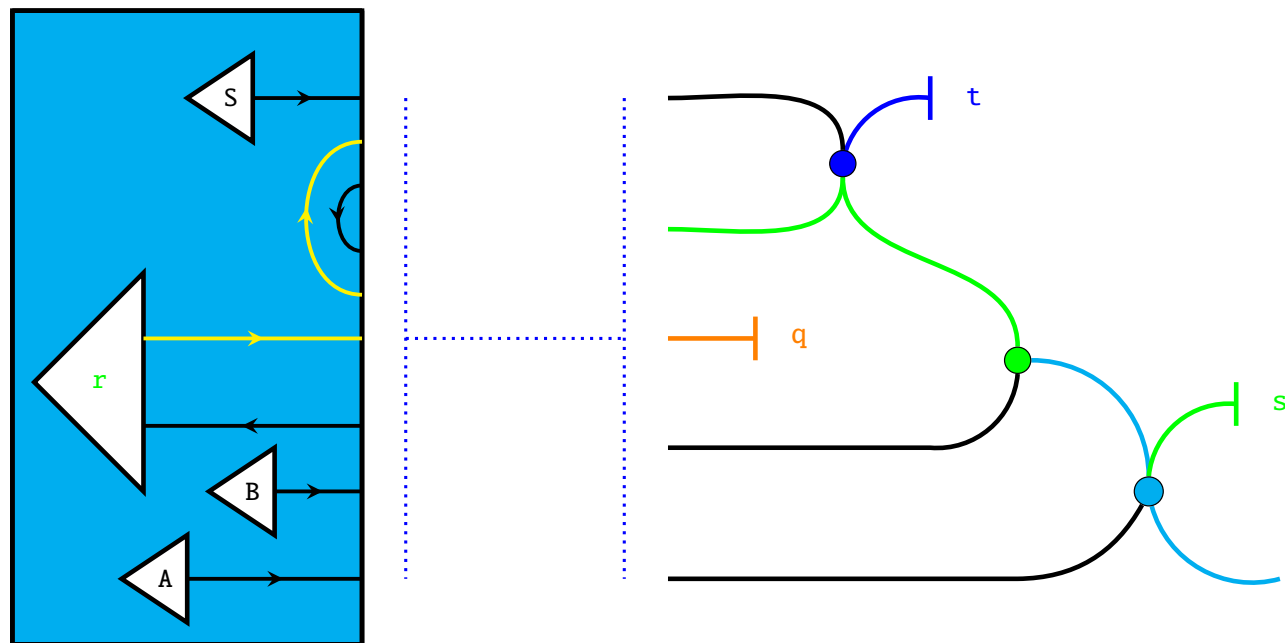


Figure 23: However, this procedure as described is at risk of being ill-defined. Observe that in the third diagram of Figure 21, the assignment of wires in the domain of the functor to wires in the codomain of the functor is only declared by diagrammatic grouping; if we consider the algebraic data available in the third diagram, really all we have is the data in the figure. How do we know which wires in the domain of the functor correspond to which wires in the codomain? Resolving this issue by formalising this diagrammatic grouping of wires algebraically is the purpose of the next section.

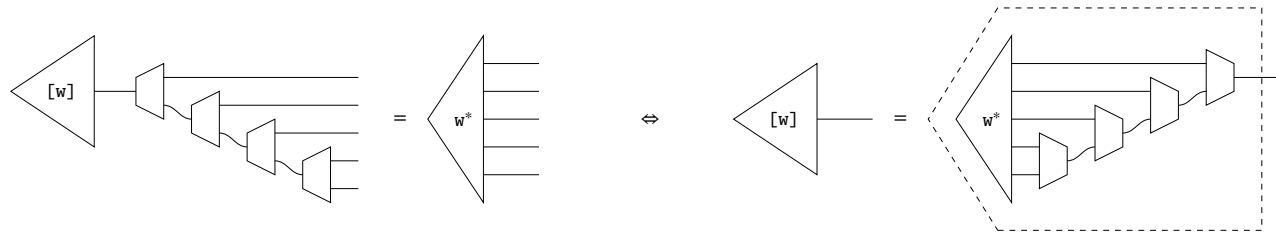
0.3 Strictified diagrams for monoidal categories

The crux of the issue sketched in Figure 23 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 $w : I_\otimes \rightarrow x_1 \otimes \cdots \otimes x_n$ that generates **PGD**, we create two corresponding generators $w^* : I_\bullet \rightarrow x_1 \bullet \cdots \bullet x_n$ and $[w] : I_\bullet \rightarrow (\cdots (x_1 \bullet x_2) \bullet x_3) \cdots \bullet x_n$. $[w]$ is a left-bracketed tensoring, and w^* is fully detensored. Note that $[w]$ and 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.

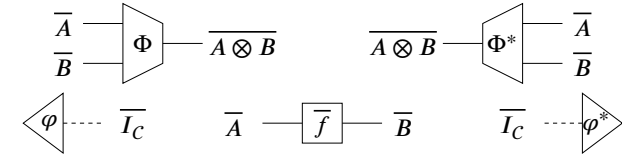


The w^* and $[w]$ generate a freely strictified rigid autonomous category $\overline{\mathbf{PGD}}^+$, from which we obtain the desired $\overline{\mathbf{PGD}}^*$ as a subcategory generated by:

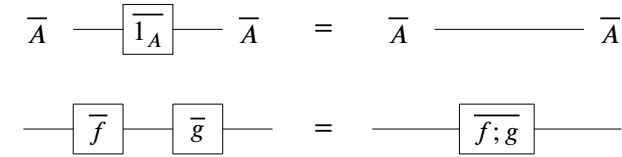
1. All $[w]$
2. Let $[A \cdot B \cdots Z]$ denote the left-nested tensoring $((A \otimes B) \cdots \otimes Z)$, and let \mathbf{X} denote $(\bigotimes_i X_i)$. For each di-

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

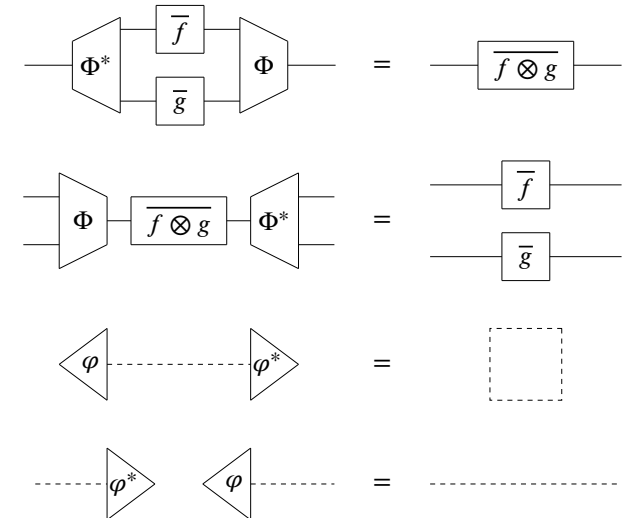
- (1) Objects \bar{A} for each $A \in \mathcal{C}$
- (2) The following generators, with $\bar{f} : \bar{A} \rightarrow \bar{B}$ for each $f \in \mathcal{C}(A, B)$, where we adopt the convention of notating the monoidal unit with a dashed line:



- (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:

$$\bar{\alpha} := \text{Diagram with } \Phi^* \text{ and } \Phi \text{ boxes and wires}$$

$$\overline{\alpha^{-1}} := \text{Diagram with } \Phi^* \text{ and } \Phi \text{ boxes and wires}$$

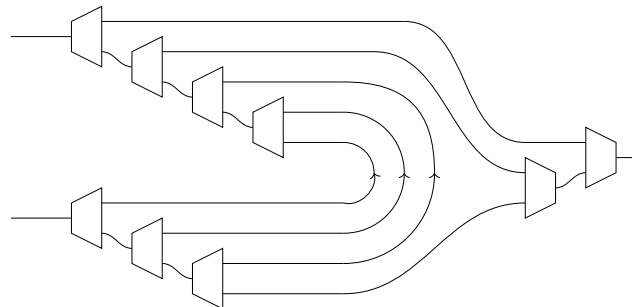
$$\bar{\lambda} := \text{Diagram with } \Phi^* \text{ and } \varphi^* \text{ boxes and wires} \quad \bar{\rho} := \text{Diagram with } \Phi^* \text{ and } \varphi^* \text{ boxes and wires}$$

$$\overline{\lambda^{-1}} := \text{Diagram with } \varphi \text{ and } \Phi \text{ boxes and wires} \quad \overline{\rho^{-1}} := \text{Diagram with } \varphi \text{ and } \Phi \text{ boxes and wires}$$

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

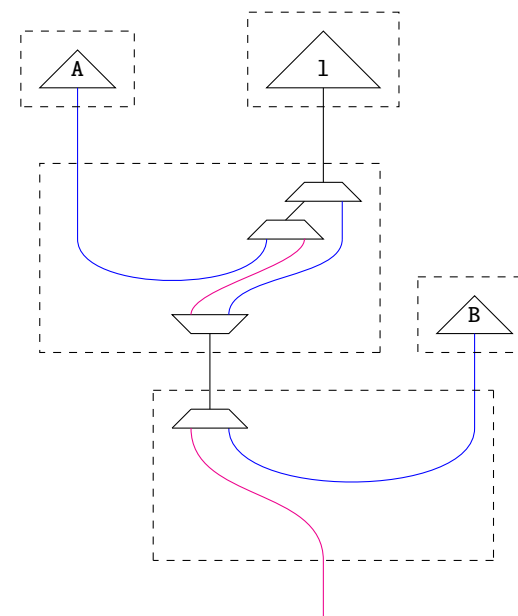
CITE □

rected cap $\mathbf{X} \otimes \mathbf{X}^{-1} \rightarrow I$ (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.

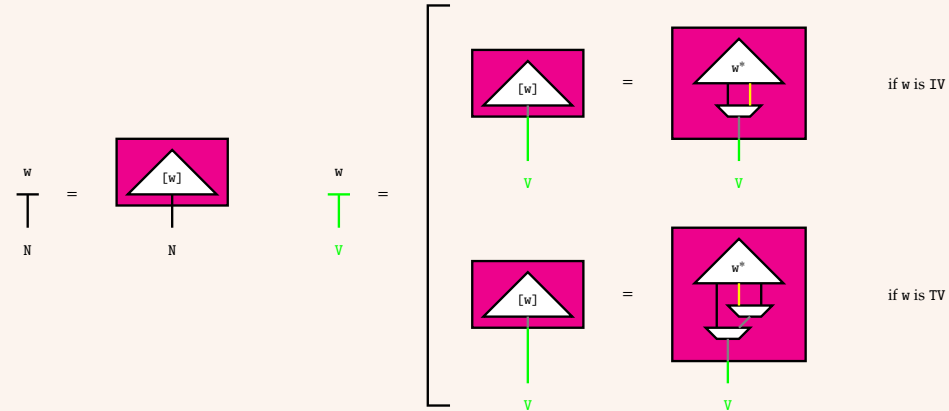


Example 0.3.4 (Pregroups with bracketing recover proof trees). The essence of the construction is to maintain a correspondence with proof-tree structure: a left-bracketed collection of types corresponds to a pregroup typing that is stuck together as the outcome of a sequent rule and must thereafter travel 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.

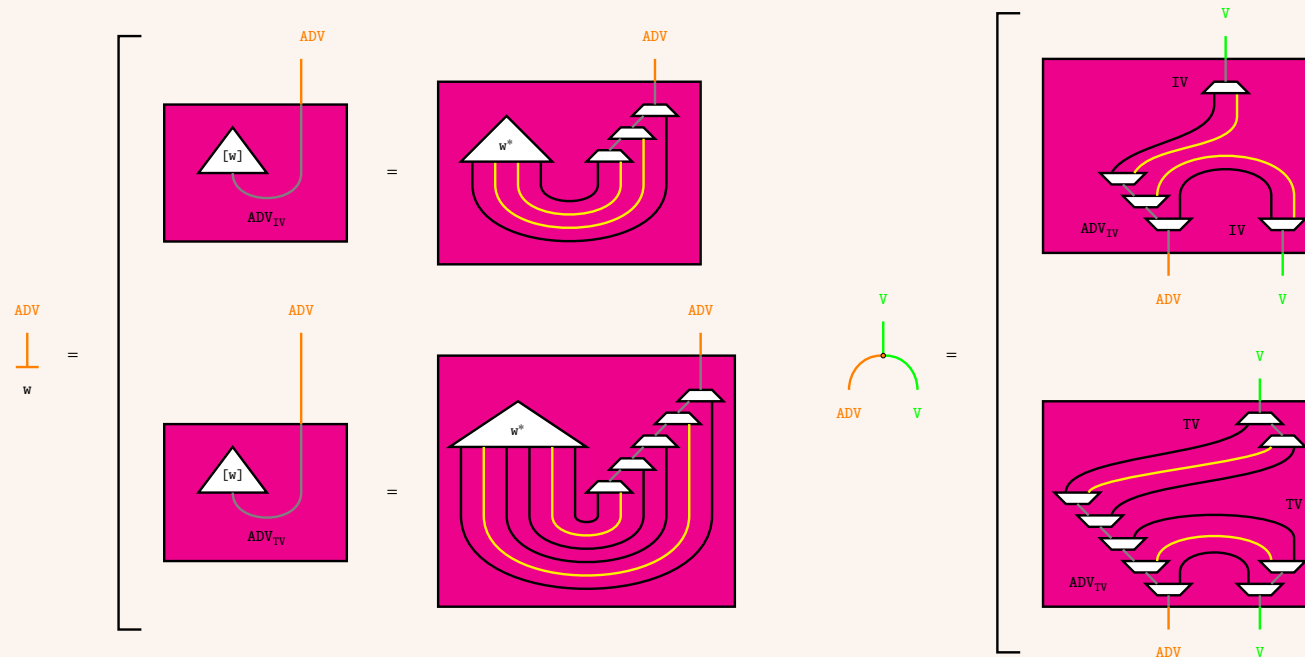
$$\frac{\frac{\text{Alice} : n \quad \text{likes} : ^-n \cdot s \cdot n^-}{\text{Alice_likes} : s \cdot n^-} \quad \text{Bob} : n}{\text{Alice_likes_Bob} : s}$$



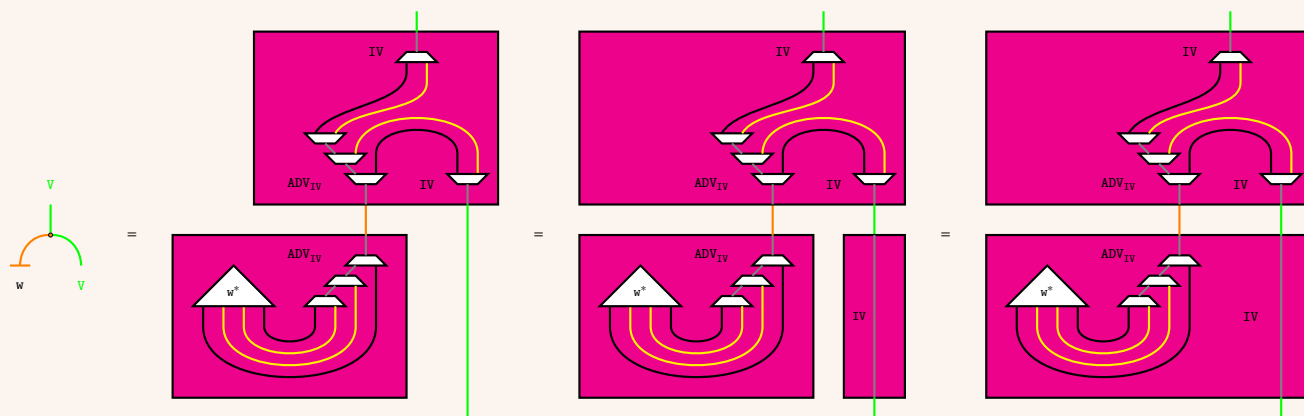
Construction 0.3.5 (Discrete monoidal opfibration from pregroups with bracketing into dependent CFGs). We aim to elucidate the pregroup with bracketing in sufficient detail to describe the functor into dCFGs; in particular, we need to know what the generators of the pregroup are. The fibre over a noun 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 .



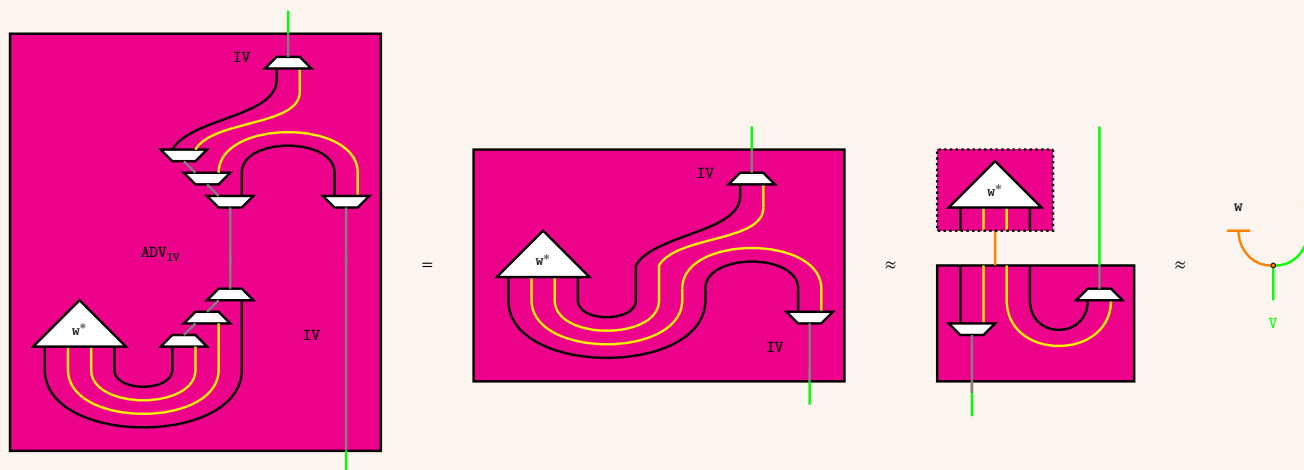
We depict the case of adverbs explicitly. Here are two lifts of the adverb label in the fibre of the opfibration corresponding to the intransitive and transitive case, and the corresponding lifts for the introduction rule for adverbs.



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. Consider the IV lift of the introduction of ADV label. We obtain the following series of 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 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 the snakes 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:

$$S, N, V, ADV, ADP$$

The determined equations of the system are the assignments of the types N and S .

$$S = s$$

$$N = n$$

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*.

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

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.

$$V_1 = (\bar{n} \cdot s)$$

$$V_2 = (\bar{n} \cdot s \cdot n^-)$$

The three dependent types are:

$$V_{x \mapsto 1(x)} = (\bar{n} \cdot s)$$

$$ADV_x = V_x \cdot V_x^-$$

$$ADP_x = \bar{V}_x \cdot V_{(x)1} \cdot n^-$$

The types V , ADV , ADP are hence indexed over a string language $x := 1 \mid 2 \mid 1(x) \mid (x)1$. We have depicted the solutions for ADV_1 and 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 V , ADV , 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.

These definitions and conventions follow [CITE](#). Given a (small) category \mathcal{C} we notate the objects C_0 and the morphisms C_1 , hence a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ consists of an object assignment $F_0 : C_0 \rightarrow D_0$ and a morphism assignment $F_1 : C_1 \rightarrow D_1$.

Definition 0.4.1 (Cofunctors). From [CITE Defn 2.2](#). A cofunctor $(f, \varphi) : \mathcal{C} \rightharpoonup \mathcal{D}$ consists of a function $f : C_0 \rightarrow D_0$ 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} :

$$(c \in C_0, f(c) \xrightarrow{u} b \in D_1) \mapsto a \xrightarrow{\varphi(c,u)} \text{cod}(\varphi(c,u))$$

The following conditions are required:

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

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

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.

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

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.

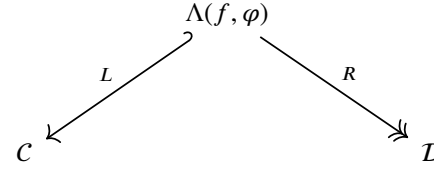
Proposition 0.3.6 (Construction 0.3.5 is a discrete monoidal fibration). *Proof.* Monoidality is evident. For unique lifts, we observe that for each bracketing of wires, construction 0.3.3 guarantees a unique lift for each introduction rule in the dCFG, and Construction 0.3.5 guarantees a unique lift for each label. For the additional interchange condition of Definition 19, 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. \square

0.4 Monoidal cofunctor boxes

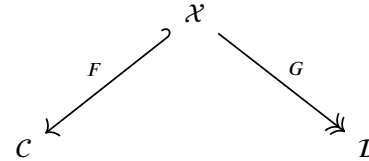
Now that we know how to solve the wire-assignment problem with the help of monoidal discrete opfibrations from strictified diagrams that do bracketing, we can at last see what monoidal cofunctor boxes are.

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

Proposition 0.4.4 (Cofunctors as spans of functors). From [CITE, Prop 2.10](#), all cofunctors $(f, \varphi) : \mathcal{C} \rightharpoonup \mathcal{D}$ 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,



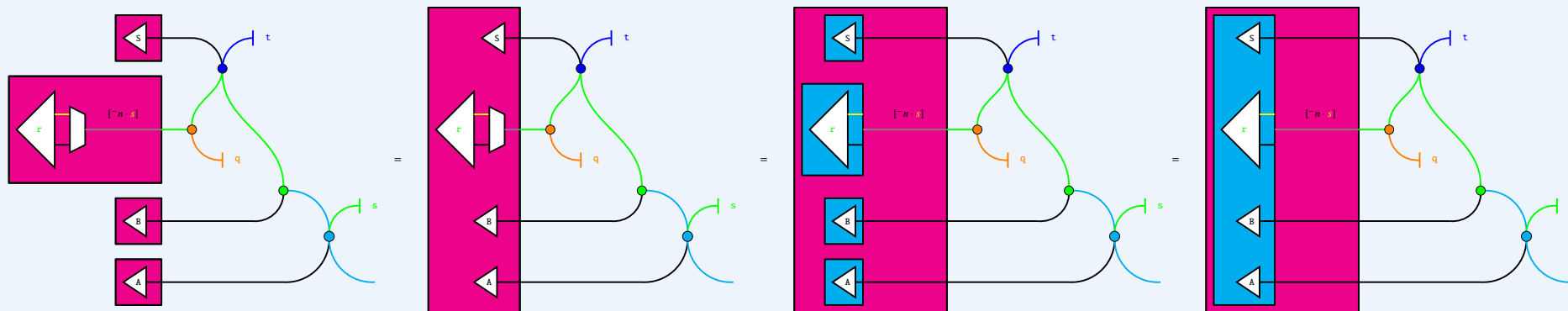
there exists a cofunctor $(f, \varphi) : \mathcal{C} \rightharpoonup \mathcal{D}$ and an isomorphism $J : \Lambda(f, \varphi) \rightarrow \mathcal{X}$ such that $FJ = L$ and $GJ = R$.

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 19.

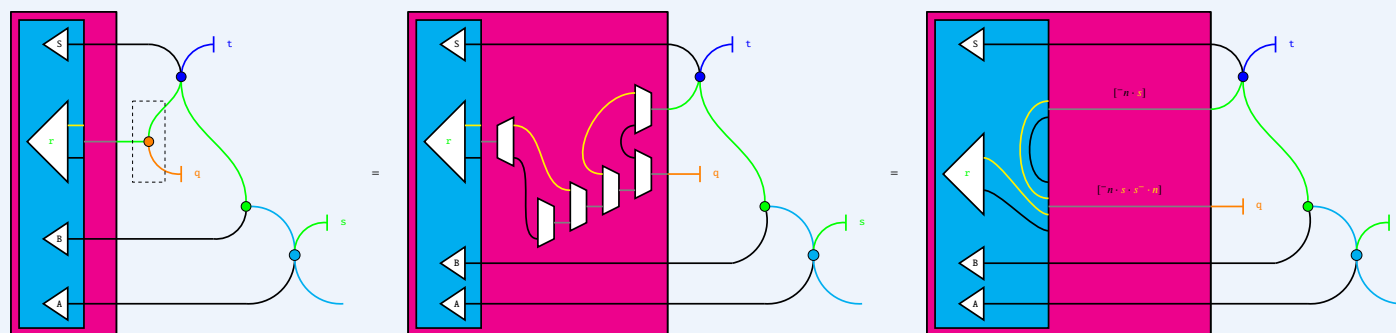
Construction 0.4.6 (Monoidal cofunctor box). A monoidal cofunctor box first uses the inside-out convention for functor boxes for the right leg, and then the outside-in convention for the left leg.

0.4.1 *From speaker to listener*

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.

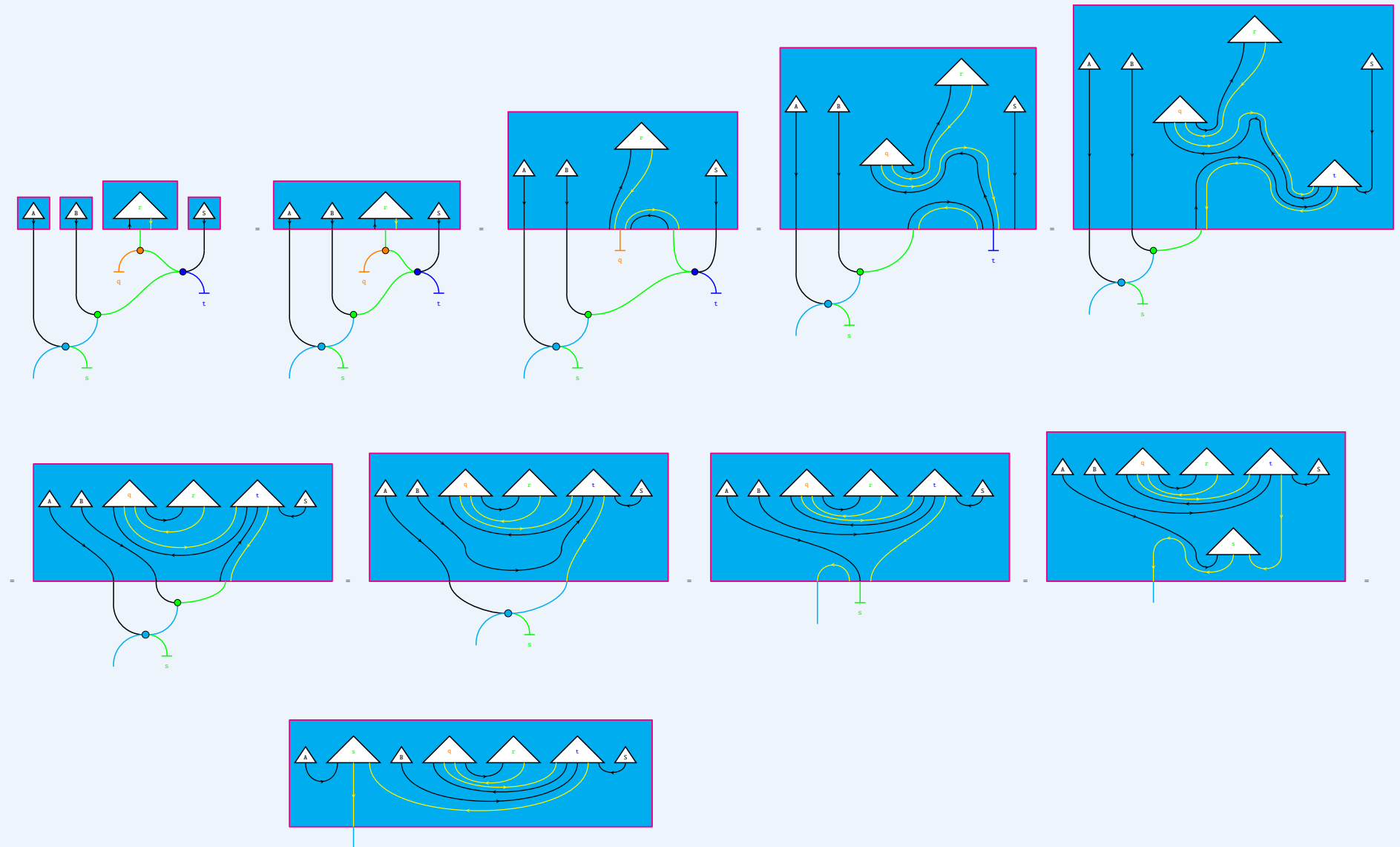


For the first equation, the magenta opfibration box now has already chosen a codomain for the lift, so it can eat the next morphism highlighted in a dashed box. For the second equation, note that eating-rules swap over for the different conventions of functor boxes: while inside-out functor boxes need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



0.4.2 *From listener to speaker*

0.5 *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 0.5.1 (Exclamations: how meaning affects grammar). The following examples from [Lakoffecture] 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) $\dots \neg \exists x_{\text{person}} \dots$ of God knows is lost, and what is left is a literal reading $\dots \neg \text{knows}(\text{God}, \dots) \dots$. 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 0.5.2 (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:

$$\begin{array}{c}
 \frac{\text{the} : n \cdot n^{-1} \quad \text{old} : n \cdot n^{-1}}{\text{the_old} : n \cdot n^{-1}} \quad \text{man} : n \quad \frac{\text{the} : n \cdot n^{-1} \quad \text{boat} : n}{\text{the_boat} : n} \\
 \hline
 \text{the_old_man} : n \quad \text{the_boat} : n \\
 \hline
 \text{Not a sentence!}
 \end{array}$$

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:

$$\begin{array}{c}
 \frac{\text{the} : n \cdot n^{-1} \quad \text{old} : n}{\text{the_old} : n} \quad \text{man} :^{-1} n \cdot s \cdot n^{-1} \quad \frac{\text{the} : n \cdot n^{-1} \quad \text{boat} : n}{\text{the_boat} : n} \\
 \hline
 \text{the_old_man} : s \cdot n^{-1} \quad \text{the_old} : n \\
 \hline
 \text{the_old_man_the_boat} : s
 \end{array}$$

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 0.5.3 (Ambiguous scoping). Consider the following sentence:

Everyone loves someone

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

$$\begin{array}{c}
 \frac{\text{everyone} : (n \multimap s) \multimap s \quad \text{loves} : n \multimap s \multimap n}{\text{everyone_loves} : s \multimap n} \quad \text{someone} : (s \multimap n) \multimap s \\
 \hline
 \text{everyone_loves_someone} : s \\
 \\
 \frac{\text{everyone} : (n \multimap s) \multimap s \quad \frac{\text{loves} : n \multimap s \multimap n \quad \text{someone} : (s \multimap n) \multimap s}{\text{loves_someone} : n \multimap s}}{\text{everyone_loves_someone} : s}
 \end{array}$$

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:

$$\llbracket \text{everyone} \rrbracket = \lambda(\lambda x. V(x)). \forall x : V(x) \quad (1)$$

$$\llbracket \text{loves} \rrbracket = \lambda x \lambda y. \text{loves}(x, y) \quad (2)$$

$$\llbracket \text{someone} \rrbracket = \lambda(\lambda y. V(y)). \exists y : V(y) \quad (3)$$

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.

$$\begin{array}{c}
 \frac{\lambda(\lambda x. V(x)). \ulcorner \forall x : V(x) \urcorner : (n \multimap s) \multimap s \quad \lambda x \lambda y. \ulcorner \text{loves}(x, y) \urcorner : n \multimap s \multimap n}{\lambda y. \ulcorner \forall x : \text{loves}(x, y) \urcorner : s \multimap n} \quad \lambda(\lambda y. V(y)). \ulcorner \exists y : V(y) \urcorner : (s \multimap n) \multimap s \\
 \hline
 \ulcorner \exists y \forall x : \text{loves}(x, y) \urcorner : s \\
 \\
 \frac{\lambda(\lambda x. V(x)). \ulcorner \forall x : V(x) \urcorner : (n \multimap s) \multimap s \quad \frac{\lambda x \lambda y. \ulcorner \text{loves}(x, y) \urcorner : n \multimap s \multimap n \quad \lambda(\lambda y. V(y)). \ulcorner \exists y : V(y) \urcorner : (s \multimap n) \multimap s}{\lambda x. \ulcorner \exists y : \text{loves}(x, y) \urcorner : n \multimap s}}{\ulcorner \forall x \exists y : \text{loves}(x, y) \urcorner : s}
 \end{array}$$

Example 0.5.4. (grammar pieces – as simple as it gets) Bob runs. Bob quickly runs. Bob drinks beer. Bob quickly drinks beer.