VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

(Acknowledgements will go in a margin note here.)

0.1 An introduction to weak n-categories for formal linguists

Geometrically, a set is a collection of labelled zero-dimensional points. A category is a collection of one-dimensional labelled arrows upon a set of zero-dimensional points (that satisfy certain additional conditions, such as having identity morphisms and associativity of composition.) Naturally, we might ask what happens when we generalise to more dimensions, asking for two-dimensional somethings going between one-dimensional arrows, and three-dimensional somethings going between two-dimensional somethings, and so on. This is the gist of an n-category, where n is a positive integer denoting dimension.

There is a fork in the road in generalisation. First, different choices of what n-dimensional somethings could be give different conceptions of n-category, because there are multiple mathematically well-founded choices for filling in the blank of `points, lines, ???`. Simplices, cubical sets, and globular sets are three common options; though the names are fancy, they correspond to triangular, cubical, and circular families of objects indexed by dimension.

Second, there is a distinction between *strict* and *weak* n-categories. Just as in regular or 1-category theory we are interested in objects up to isomorphism rather than equality – because isomorphic objects in a category are as good as one another – lifting this philosophy to n-categories gives us *weak* n-categories. In a strict k-category, all of the j-dimensional morphisms for $j > k$ are identity-equalities. In a weak k-category, all equalities for dimensions $j > k$ are replaced by isomorphisms all the way up: which means that a k-equality $\alpha = \beta$ in the strict setting is replaced by a pair of $k+1$-morphisms witnessing the isomorphism of α and β, and that pair of $k+1$-morphisms has a pair of $k+2$ morphisms witnessing that they are $k+1$-isomorphic, and so on for $k+3$ and all the way up. Unsurprisingly, strict n-categories are easy to formalise, and weaks n-categories are hard. A conjecture or guiding principle like the Church-Turing thesis holds for weak n-categories that they should all be equivalent to one another, whatever equivalence means.

Mathematicians, computer scientists, and physicists may have good reasons to work with weak n-categories CITE , but what is the value proposition for formal linguists? A philosophical draw for the formal semanticist is that insofar as semantics is synonymy – the study of when expressions are equivalent – weak n-categories are an exquisite setting to control and study sameness in terms of meta-equivalences. A practical draw for the formal syntactician is that weak n-categories provide a natural setting to model generic rewriting systems, and that is what we will focus on here. In this setting we will introduce weak n-categories in the `homotopy.io` formulation, along the way showing how they provide a common setting to formalise string-rewrite and tree-adjoining grammars, setting the stage for us to specify a circuit-adjoining grammar for text circuits later on.

0.1.1 *String-rewrite systems as 1-object-2-categories*

Say we have an alphabet $\Sigma := \{\alpha, \beta, \gamma\}$. Then the Kleene-star Σ^* consists of all strings (including the empty string ϵ) made up of Σ, and we consider formal languages on Σ to be subsets of Σ^*. Another way of viewing Σ^* is as the free monoid generated by Σ under the binary concatenation operation $(\_ \cdot \_)$ which is associative and unital with unit ϵ, the empty string. Associativity and unitality are precisely the conditions of composition of morphisms in categories, so we have yet another way to express Σ^* as a finitely presented category; we consider a category with a single object $\star$, taking ϵ to be the identity morphism $\mathbf{1}_\star$ on the single object, and we ask for the category obtained when we consider the closure under composition of three non-identity morphisms $\alpha, \beta, \gamma : \star \to \star$. In this category, every morphism $\star \to \star$ corresponds to a string in Σ^*. We illustrate this example in the margins. A string-rewrite system additionally consists of a finite number of string-transformation rules. Building on our example, we might have a named rule $R : \alpha \mapsto \beta \cdot \gamma$, which we illustrate in Figure 0.1.1.

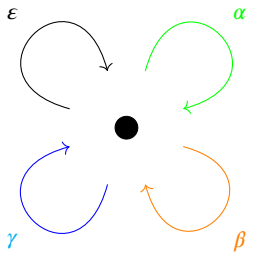


Figure 1: The category in question can be visualised as a commutative diagram.

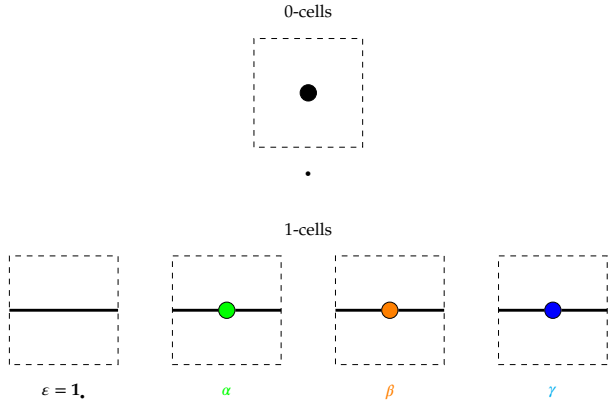
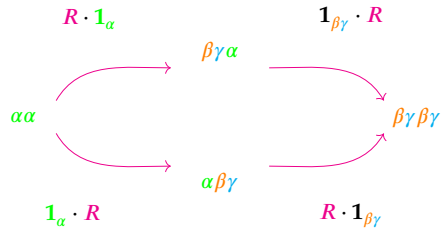
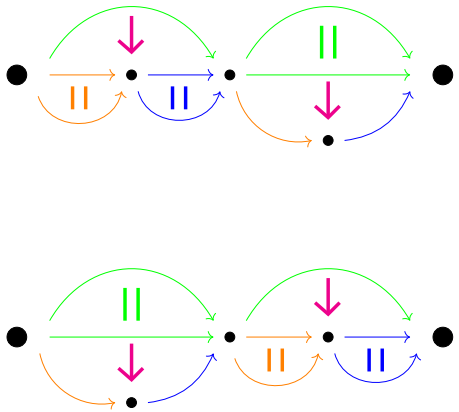


Figure 2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from tables as they come for free. Observe that composition of identities enforces the behaviour of the empty string, so that for any string x, we have $\epsilon \cdot x = x = \epsilon \cdot x$.

Figure 3: For a concrete example, we can depict the string $\alpha \cdot \gamma \cdot \gamma \cdot \beta$ as a morphism in a commuting diagram.

 In a string-rewrite system, rewrites are applied one at a time. This means that even for our simple example, there are two possible rewrites from $\alpha \cdot \alpha$ to obtain $\beta \cdot \gamma \cdot \beta \cdot \gamma$. Here are the two rewrites viewed in two equivalent ways, first on the left informally where strings are nodes in a graph and rewrites are labelled transitions and secondly on the right as two distinct commuting 2-diagrams.





What should we say about how these two different rewrites relate to each other? Let's say Alice is a formal linguist who is only interested in what strings are reachable from others by rewrites – this is *de rigeur* when we consider formal languages to be subsets of Σ^*. She might be happy to declare that these two rewrites are simply equal; categorically this is tantamount to her declaring that any two 2-cells in the 1-object-2-category that share the same source and target are in fact the same, or equivalently, that any n-cells for $n \geq 3$ are iden-

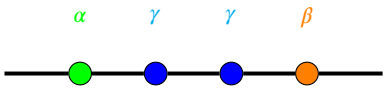


Figure 4: The string-diagrammatic view, where ⋆ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

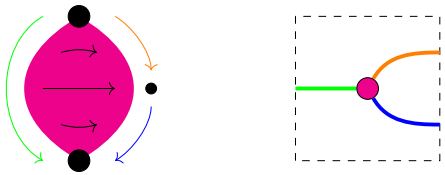


Figure 5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual of this view gives us a string diagram for the 2-cell R.

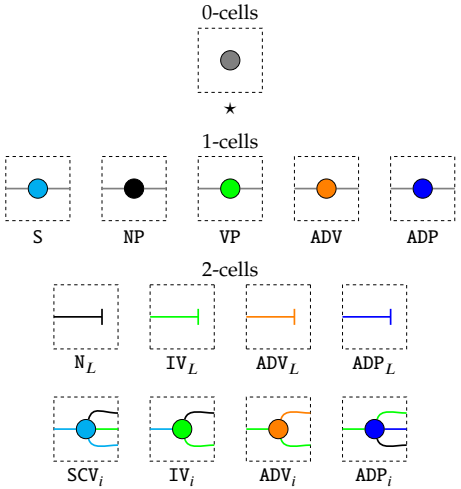
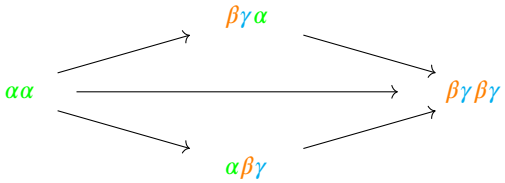
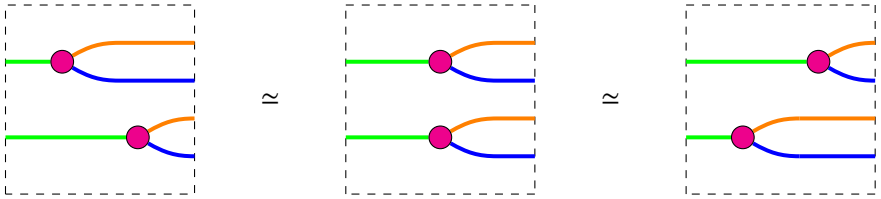


Figure 6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for `Alice sees Bob quickly run to school`.

tities. In fact, what Alice really cares to have is a category where the objects are strings from Σ^*, and the morphisms are a reachability relation by rewrites; this category is *thin*, in that there is at most one arrow between each pair of objects, which forgets what rewrites are applied.



Let's say Bob is a different sort of formal linguist who wants to model the two rewrites as nonequal but equivalent, with some way to keep track of how different equivalent rewrites relate to one another. Bob might want this for example because he wants to show that head-first rewrite strategies are the same as tail-first, so he wants to keep the observation that the two rewrites are equivalent in that they have the same source and target, while keeping the precise order of rewrites distinct. This order-independence of disjoint or non-interfering rewrites is reflected in the interchange law for monoidal categories, which in the case of our example is depicted as:



In fact, Bob gets to express a new kind of rewrite in the middle: the kind where two non-conflicting rewrites happen *concurrently*. The important aspect of Bob's view over Alice's is that equalities have been replaced by isomorphisms between syntactically inequal rewrites. This demotion of equalities to isomorphisms means that Bob is dealing with a *weak* 1-object-2-category; Bob does have 3-cells that relate different 2-cells with the same source and target, but all of Bob's n-cells for $n \geq 3$ are isomorphisms, rather than equalities.

0.1.2 Tree Adjoining Grammars

Definition 0.1.1 (*Elementary* Tree Adjoining Grammar: Classic Computer Science style)**.** An elementary **TAG** is a tuple

$$(\mathcal{N}, \mathcal{N}^{\downarrow}, \mathcal{N}^{*}, \Sigma, \mathcal{I}, \mathcal{A})$$

The first four elements of the tuple are referred to as *non-terminals*. They are:

- A set of *non-terminal symbols* $\mathcal{N}$ – these stand in for grammatical types such as `NP` and `VP`.
- A bijection $\downarrow\colon \mathcal{N} \to \mathcal{N}^{\downarrow}$ which acts as $\mathtt{X} \mapsto \mathtt{X}^{\downarrow}$. Nonterminals in $\mathcal{N}$ are sent to marked counterparts in $\mathcal{N}^{\downarrow}$, and the inverse sends marked nonterminals to their unmarked counterparts. These markings are *substitution markers*, which are used to indicate when certain leaf nodes are valid targets for a substitution operation – discussed later.
- A bijection $*\colon \mathcal{N} \to \mathcal{N}^{*}$ – the same idea as above. This time to mark *foot nodes* on auxiliary trees, which is structural information used by the adjoining operation – discussed later.

Σ is a set of *terminal symbols* – these stand in for the words of the natural language being modelled. $\mathcal{I}$ and $\mathcal{A}$ are sets of *elementary trees*, which are either *initial* or *auxiliary*, respectively. *initial trees* satisfy the following constraints:

- The interior nodes of an initial tree must be labelled with nonterminals from $\mathcal{N}$
- The leaf nodes of an initial tree must be labelled from $\Sigma \cup \mathcal{N}^{\downarrow}$

Auxiliary trees satisfy the following constraints:

- The interior nodes of an auxiliary tree must be labelled with nonterminals from $\mathcal{N}$
- Exactly one leaf node of an auxiliary tree must be labelled with a foot node $\mathtt{X}^{*} \in \mathcal{N}^{*}$; moreover, this labelled foot node must be the marked counterpart of the root node label of the tree.
- All other leaf nodes of an auxiliary tree are labelled from $\Sigma \cup \mathcal{N}^{\downarrow}$

Further, there are two operations to build what are called *derived trees* from elementary and derived trees. *Substitution* replaces a substitution marked leaf node $\mathtt{X}^{\downarrow}$ in a tree α with another tree α' that has `X` as a root node. *Adjoining* takes auxiliary tree β with root and foot nodes $\mathtt{X}, \mathtt{X}^{\star}$, and a derived tree γ at an interior node `X` of γ. Removing the `X` node from γ separates it into a parent tree with an `X`-shaped hole for one of its leaves, and possibly multiple child trees with `X`-shaped holes for roots. The result of adjoining is obtained by identifying the root of β with the `X`-context of the parent, and making all the child trees children of βs foot node $\mathtt{X}^{\star}$.

The essence of a tree-*adjoining* grammar is as follows: whereas for a CFG one grows the tree by appending branches and leaves at the top of the tree (substitution), in a TAG one can also sprout subtrees from the middle of a branch (adjoining). Now we show that this gloss is more formal than it sounds, by the following steps. First we show that the 2-categorical data of a CFG can be transformed into 3-categorical data – which we call *Leaf-Ansatz* – which presents a rewrite system that obtains the same sentences as the CFG, by a bijective correspondence between composition of 2-cells in the CFG and constructed 3-cells in the leaf-ansatz. These 3-cells in the leaf ansatz correspond precisely to the permitted *substitutions* in a TAG. Then we show how to model *adjoining* as 3-cells. Throughout we work with a running example, the CFG grammar introduced earlier. The main body covers the formal but unenlightening definition of *elementary* tree adjoining grammars which we will convert to diagrams. We will deal with the extensions of links and local constraints to adjoining shortly.

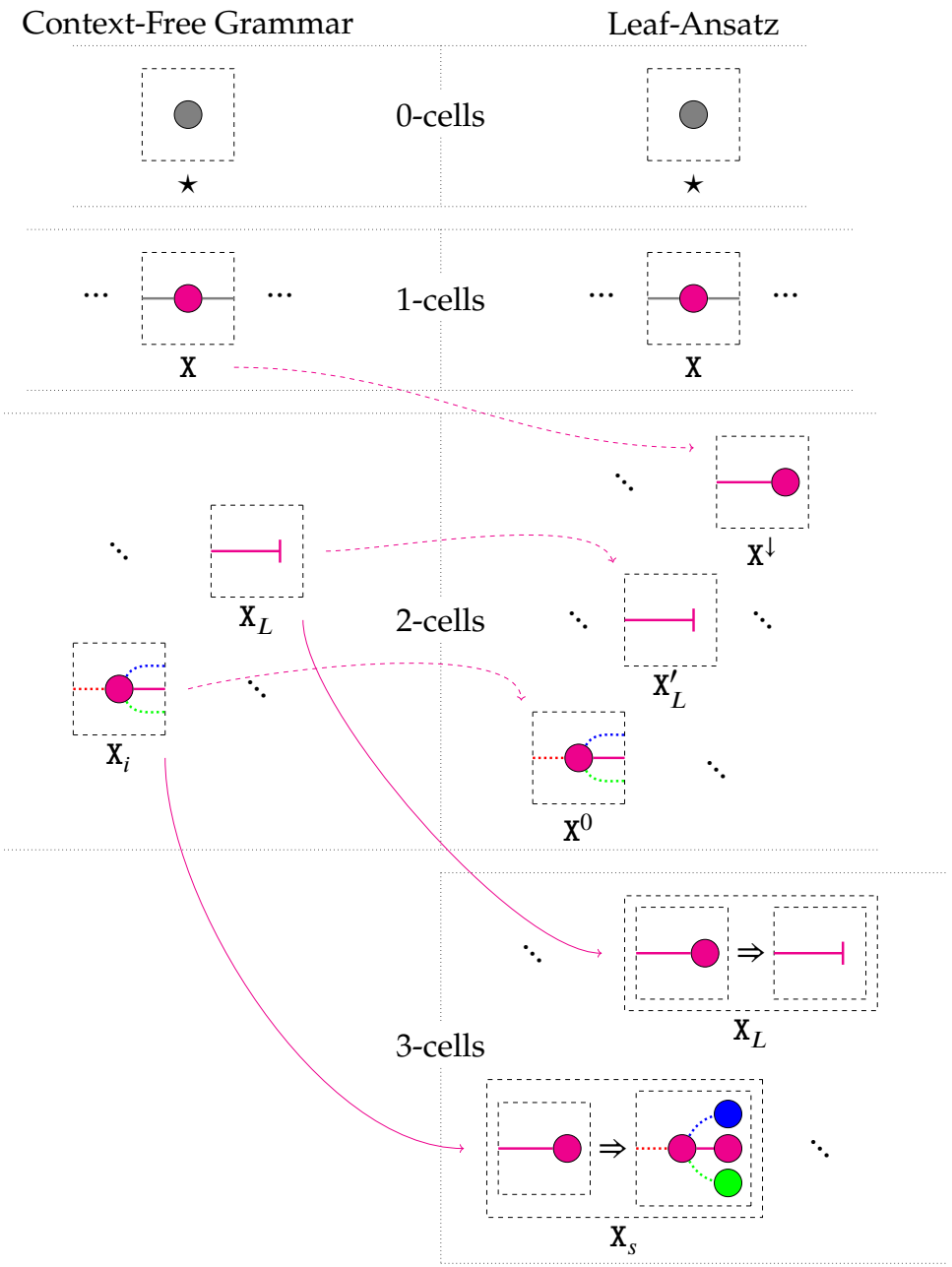


Figure 7:

Construction 0.1.2 (Leaf-Ansatz of a CFG)**.** Given a signature $\mathfrak{G}$ for a CFG, we construct a new signature $\mathfrak{G}'$ which has the same 0- and 1-cells as $\mathfrak{G}$. Now, referring to the dashed magenta arrows in the schematic below: for each 1-cell wire type $\mathtt{X}$ of $\mathfrak{G}$, we introduce a *leaf-ansatz* 2-cell $\mathtt{X}^{\downarrow}$. For each leaf 2-cell $\mathtt{X}_L$ in $\mathfrak{G}$, we introduce a renamed copy $\mathtt{X}'_L$ in $\mathfrak{G}'$. Now refer to the solid magenta: we construct a 3-cell in $\mathfrak{G}'$ for each 2-cell in $\mathfrak{G}$, which has the effect of systematically replacing open output wires in $\mathfrak{G}$ with leaf-ansatzes in $\mathfrak{G}'$.

Proposition 0.1.3. Leaf-ansatzes of CFGs are precisely TAGs with only initial trees and substitution.

Proof. By construction. Consider a CFG given by 2-categorical signature $\mathfrak{G}$, with leaf-ansatz signature $\mathfrak{G}'$. The types $\mathtt{X}$ of $\mathfrak{G}$ become substitution marked symbols $\mathtt{X}^{\downarrow}$ in $\mathfrak{G}'$. The trees $\mathtt{X}_i$ in $\mathfrak{G}$ become initial trees $\mathtt{X}^0$ in $\mathfrak{G}'$. The 3-cells $\mathtt{X}_s$ of $\mathfrak{G}'$ are precisely substitution operations corresponding to appending the 2-cells $\mathtt{X}_i$ of $\mathfrak{G}$. □

The leaf-ansatz construction just makes formal the following observation: there are multiple equivalent ways of modelling terminal symbols in a rewrite system considered string-diagrammatically. So for a sentence like `Bob drinks`, we have the following derivations that match step for step in the two ways we have considered.

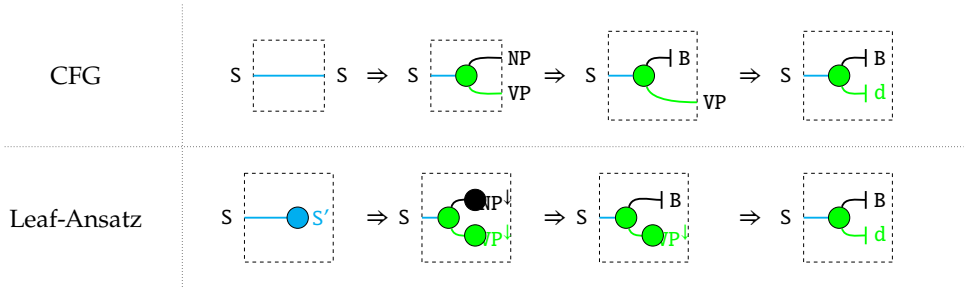


Figure 8: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction between nonterminals and terminals.

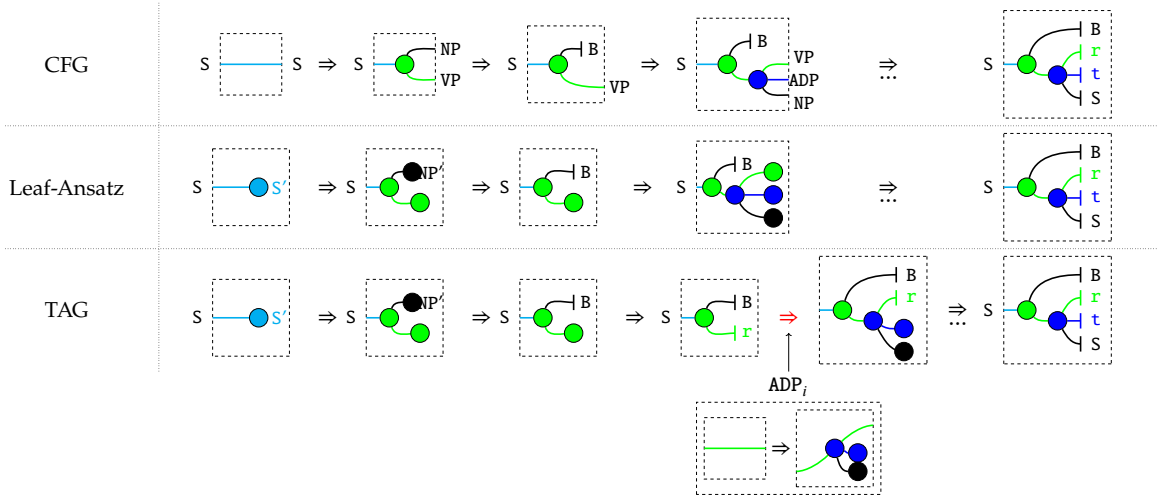
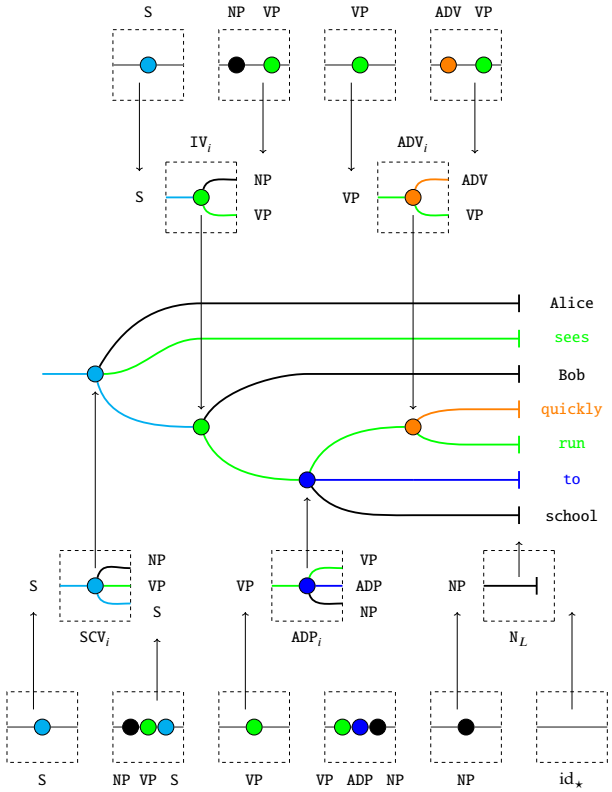


Figure 9: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted in CFGs, since terminals can no longer be modified. The adjoining operation of TAGs gets around this constraint by permitting rewrites in the middle of trees.

Figure 10: Leaf-ansatz signature of `Alice sees Bob quickly run to school` CFG. One aspect of rewrite systems we adapt for now is the distinction between terminal and nonterminal symbols; terminal symbols are those after which no further rewrites are possible. We capture this string-diagrammatically by modelling terminal rewrites as 2-cells with target equal to the 1-cell identity of the 0-cell $\star$, which amounts to graphically terminating a wire. The generators subscripted L (for *label* or *leaf*) correspond to terminals of the CFG, and represent a family of generators indexed by a lexicon for the language. The generators subscripted i (for *introducing a type*) correspond to rewrites of the CFG. Reading the central diagram in the main body from left-to-right, we additionally depict the breakdown of the derivation in terms of rewrites of lower dimension from our signature.



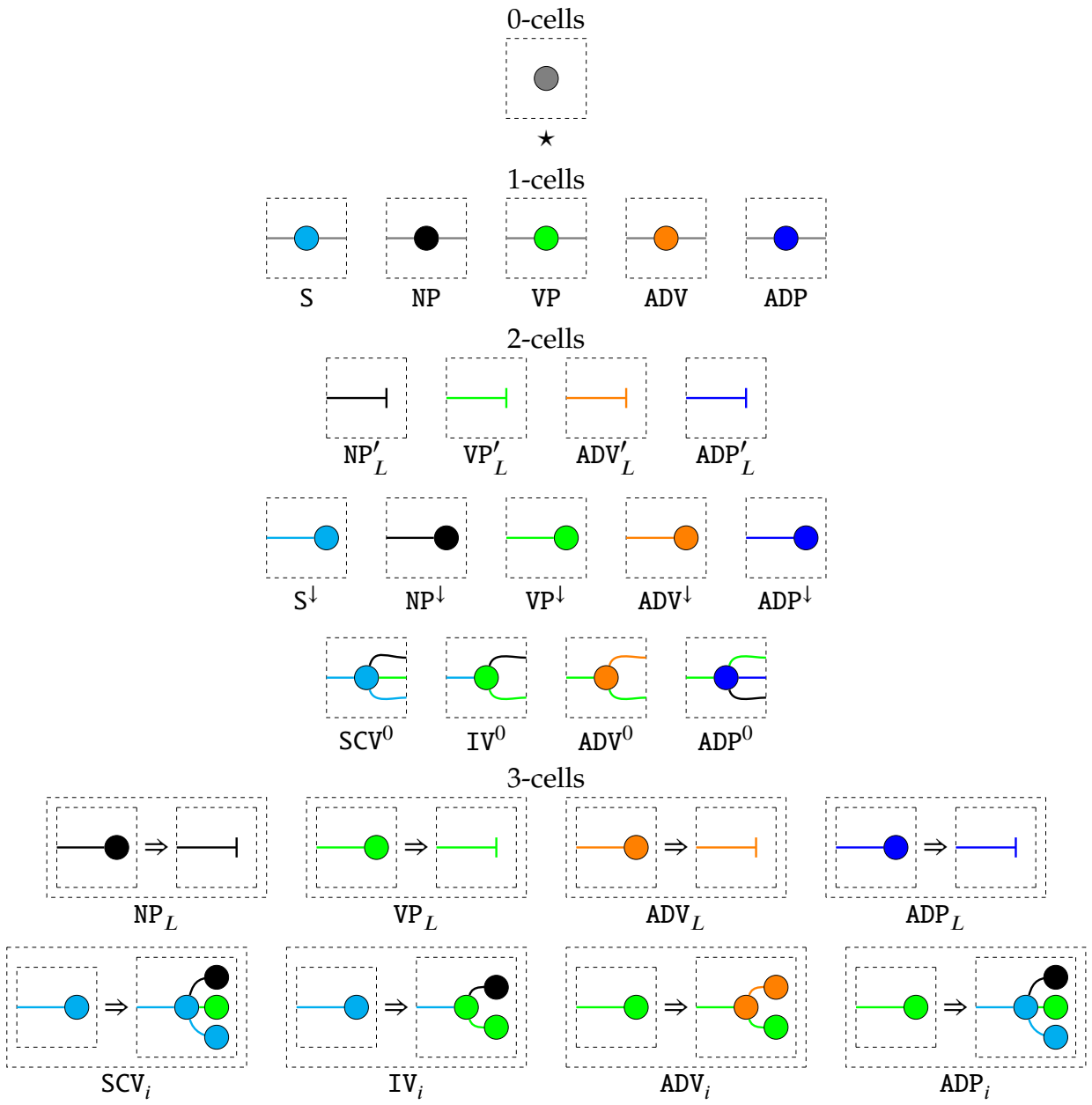
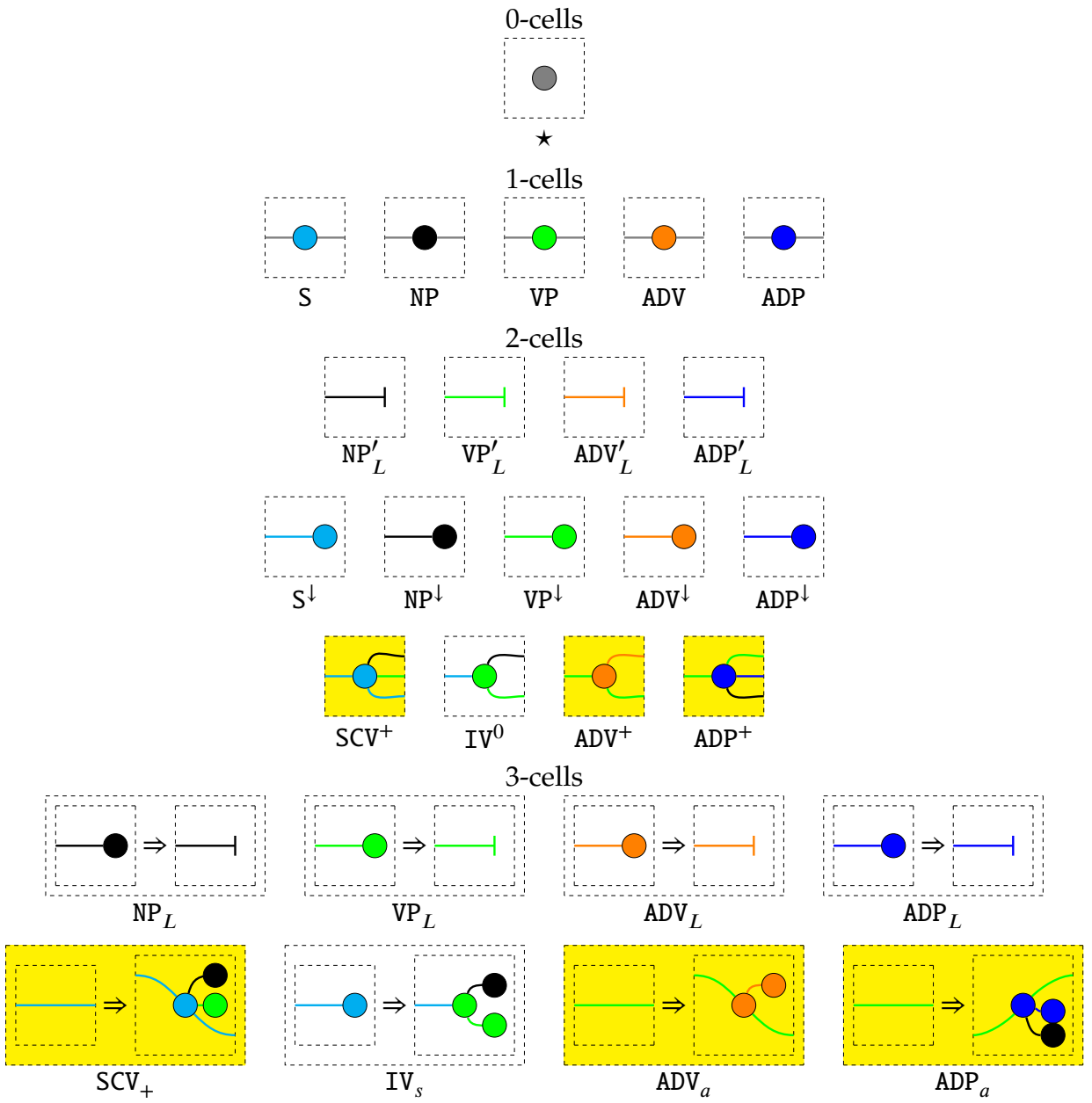


Figure 11: TAG signature of `Alice sees Bob quickly run to school`. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction yields as a corollary an alternate proof of Theorem [Joshi 6.1.1...]...

Corollary 0.1.4. For every context-free grammar $\mathfrak{G}$ there exists a tree-adjoining grammar $\mathfrak{G}'$ such that $\mathfrak{G}$ and $\mathfrak{G}'$ are strongly equivalent – both formalisms generate the same set of strings (weak equivalence) and the same abstract syntactic structures (in this case, trees) behind the strings (strong equivalence).

Proof. Proposition 7 provides one direction of both equivalences. For the other direction, we have to show that each auxiliary tree (a 2-cell) and its adjoining operation (a 3-cell) in $\mathfrak{G}'$ corresponds to a single 2-cell tree of some CFG signature $\mathfrak{G}$, which we demonstrate by construction. The highlighted 3-cells of $\mathfrak{G}'$ are obtained systematically from the auxiliary 2-cells as follows: the root and foot nodes $\mathtt{X}, \mathtt{X}^\star$ indicate which wire-type to take as the identity in the left of the 3-cell, and the right of the 3-cell is obtained by replacing all non-$\mathtt{X}$ open wires $\mathtt{Y}$ with their leaf-ansatzes $\mathtt{Y}^\downarrow$. This establishes a correspondence between any 2-cells of $\mathfrak{G}$ considered as auxiliary trees in $\mathfrak{G}'$. □



0.1.3 *Tree adjoining grammars with local constraints*

Definition 0.1.5 (TAG with local constraints: CS-style)**.** [Joshi] $G = (I, A)$ is a TAG with local constraints if for each node n and each tree t, exactly one of the following constraints is specified:

1. Selective adjoining (SA): Only a specified subset $\bar{\beta} \subseteq A$ of all auxiliary trees are adjoinable at n.
2. Null adjoining (NA): No auxiliary tree is adjoinable at the node n.
3. Obligatory Adjoining (OA): At least one out of all the auxiliary trees adjoinable at n must be adjoined at n.

The usual conception of TAGs includes two extensions to the basic definition presented above. First, there may be *local constraints* on adjoining, which only allows certain trees to be adjoined at certain nodes. Second, TAGs may have links, which are extra edges between nodes obeying a c-command condition. Here we deal with local constraints; dealing with links requires the introduction of braiding.

The n-categorical approach easily accommodates local constraints. For (SA), whereas before we take the source of adjoining rewrites to be identities, we can instead introduce ansatz endomorphisms that are rewritable to the desired subsets. For (NA), we assert that identities have no rewrites (NA). For (OA), we can make a distinction between finished and unfinished derivations, where we require that unfinished derivations are precisely those that still contain an obligatory-rewrite endomorphism.

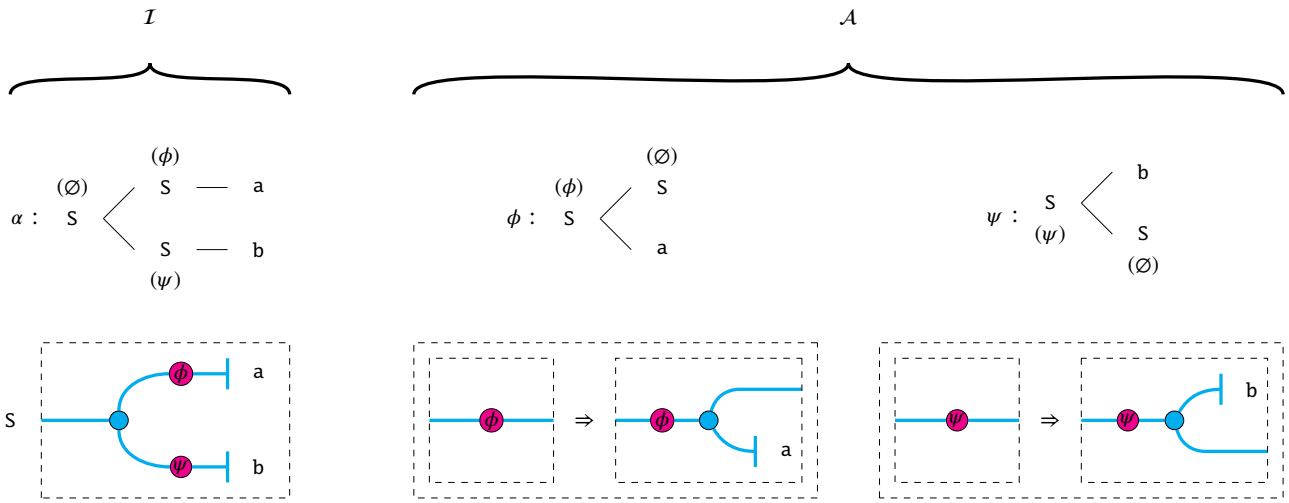


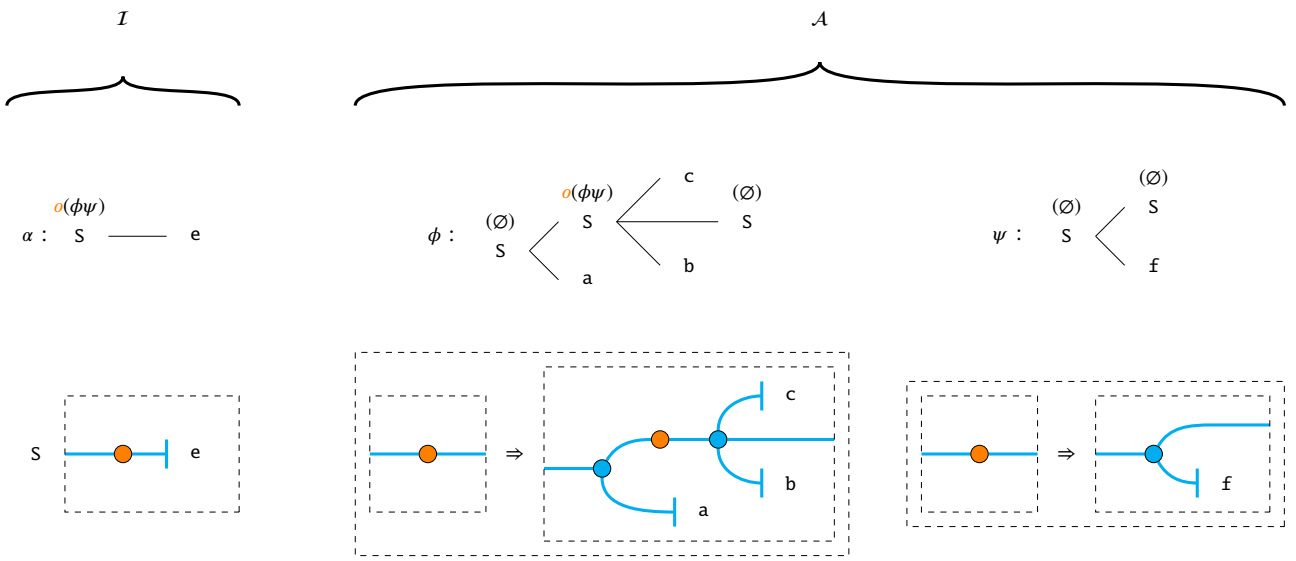
Figure 12: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree is presented as a 2-cell where the (SA) rules are rewritable nodes, that serve as sources of rewrites in the 3-cell presentations of the auxiliary trees.

0.1.4 *Braiding, symmetries, and suspension*

Before we can model TAGs with links, we must introduce the concepts of braiding and symmetries, which we have seen in the diagrammatic setting already as wires twisting past one another. In our current setting of 1-object-3-categories for TAGs, the diagrams we are dealing with are all planar – i.e. wires may not cross – but this is a restriction we must overcome when we are dealing with links.

First we observe that in a 1-object-2-categorical setting, a morphism from the identity ε on the base object $\star$ to itself would, in our analogy with string-rewrite systems, be a rewrite from the empty string to itself.

Figure 13: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree is presented as a 2-cell where the (OA) rule is given its own 2-cell, which is the source of rewrites in 3-cell presentations of auxiliary trees. We may capture the obligatory nature of the rewrite by asking that finished derivations contain no instance of the orange 2-cell.



For example, a rewrite R may introduce a symbol from the empty string and then delete it. A rewrite S may create a pair of symbols from nothing and then annihilate them.

$$R := \varepsilon \mapsto x \mapsto \varepsilon \qquad S := \varepsilon \mapsto a \cdot b \mapsto \varepsilon$$

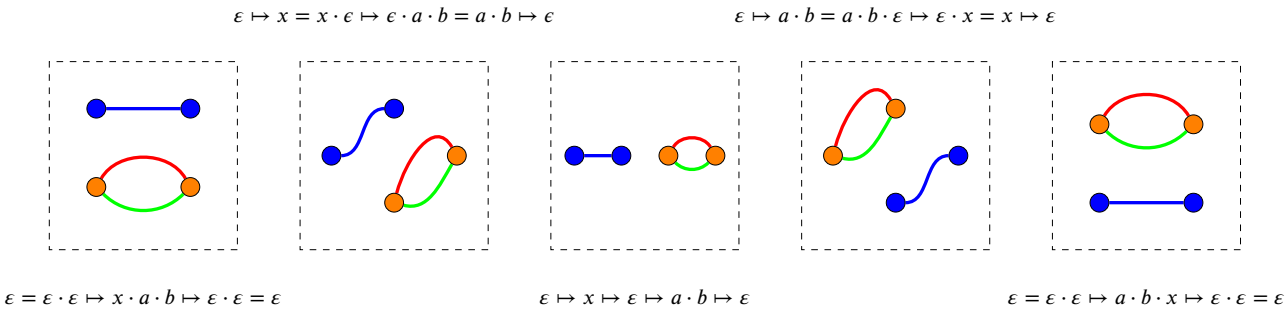


Figure 14: In our analogy with string rewrite systems, we might like that the following rewrites are equivalent, while respecting that they are not equal, representing x, a, b as blue, red, and green wires respectively. Such rewrites from the empty string to itself are more generally called *scalars* in the monoidal setting, viewed 2-categorically.

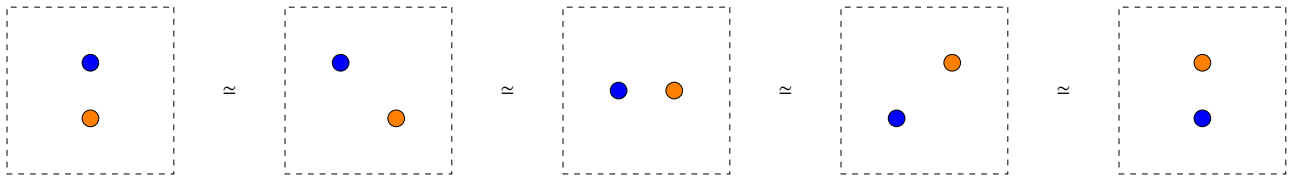


Figure 15: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument CITE is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting is that the empty string is equal to copies of itself, which creates the necessary space for manoeuvering; translating into the n-categorical setting, expressions are equivalent up to introducing and contracting identities.

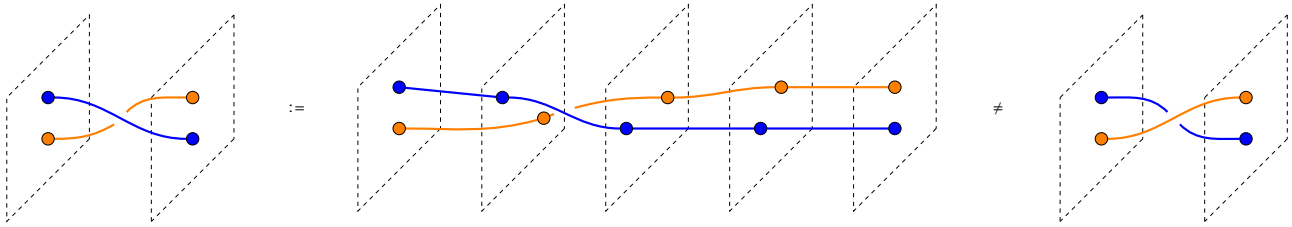


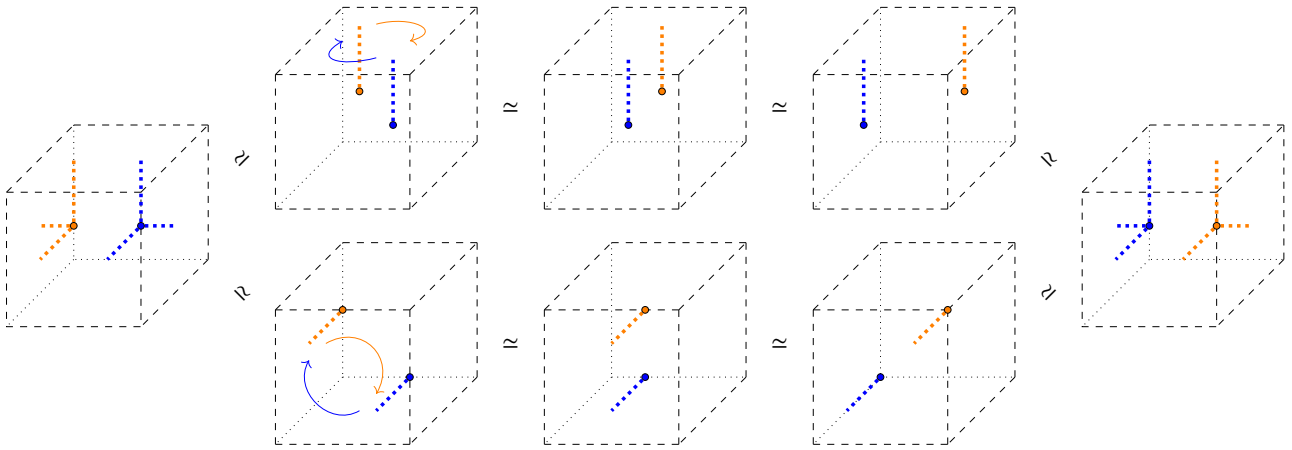
Figure 16: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies CITE , which are continuous bijective transformations that don't let wires double back on themselves, we can identify two different braidings that are not continuously deformable to one another in the 3-dimensional space of the vignette. We distinguish the braidings visually by letting wires either go over or under one another.

Now we have a setting in which we can consider wires swapping places. However, having two different ways to swap wires presents a complication. While useful for knot-theory, in symmetric monoidal categories we don't want this distinction. Perhaps surprisingly, this distinction is eliminated if we consider swapping dots in the 3D volume rather than the 2D plane. Now we have to extend our analogy to reach 1-object-4-categories. Just as symbols on a 1D string were encoded as 1-cells on the 0-cell identity initially, our braidings are the behaviour of symbols on the 2D plane, encoded as 2-cells on the 1-cell identity of the 0-cell. So, to obtain symbols in a 3D volume, we want 3-cells on the 2-cell identity of the 1-cell identity of the 0-cell. That is a mouthful, so we instead clumsily denote the stacked identity as $\mathbf{1}_{\mathbf{1}_\star}$. To obtain a dot in a 3-dimensional volume, we consider a rewrite from $\mathbf{1}_{\mathbf{1}_\star}$ to itself. These dots also enjoy a version of the Eckmann-Hilton argument in 3 dimensions (which holds for all dimensions 2 and higher).

Considering these changes of dot positions in 3D as a 4D vignette gives us braidings again. But this time, they are equivalent up to processive isotopy – in other words, any two ways of swapping the dots in the volume are continuously deformable to one another in the 4-dimensional space of the vignette. The reason is intuitive: a clockwise and counterclockwise braiding along one composition axis can be mutually deformed by making use of the extra available axis of composition. So we have eliminated the differences between the two kinds of braidings.

To recap: We follow the convention that object dimensions start at 0. In a 1-object-2-category we obtain planar string diagrams for monoidal categories, which are equivalent up to processive isotopies in an ambi-

Figure 17: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise in the new dimension. Shown below are two ways to swap left-to-right sequentially composed dots by clockwise rotations in the forward-backward and up-down directions of composition:



ent 2-dimensional space. In a 1-object-3-category, we obtain string diagrams for braided monoidal categories that are equivalent up to processive isotopy in an ambient 3-dimensional space – the page with depth. In a 1-object-4-category, we obtain string diagrams for symmetric monoidal categories, which are equivalent up to processive isotopies in an ambient 4-dimensional space – while it is difficult to visualise 4-dimensionally, diagrammatically this simplifies dramatically to just allowing wires to cross and only caring about whether input-output connectivity from left to right makes sense. Now observe that for any 1-object-2-category, we can obtain a 1-object-4-category by promoting all 2-cells and higher to sit on top of $\mathbf{1}_{1_\star}$, in essence turning dots on a line into dots in a volume; this procedure is called *suspension* CITE .

We call the above a 1-object-4-category since there is a single 0-cell object, and the highest dimension we consider is 4. Symmetric monoidal categories are equivalently seen as 1-object-4-categories CITE , which are in particular obtained by suspending 1-object-2-categories for planar string diagrams. To summarise, by appropriately suspending the signature, we power up planar diagrams to permit twisting wires, as in symmetric monoidal categories.

Remark 0.1.6 (**THE IMPORTANT TAKEAWAY!**). Now have a combinatoric way to specify string diagrams that generalises PROPs for symmetric monoidal categories, which in addition permits us to reason with directed diagram rewrites. So we have a trinity of presentations of the same mathematical concept. String diagrams are the *syntax*. Symmetric monoidal categories are the *semantics*. *n*-categories are the *finite combinatoric presentation* of theories and rewrite systems upon them. By analogy, *n*-categories provide a *vocabulary* and *rewrites* for string-diagram *expressions*, which are interpreted with respect to the *context* of a symmetric monoidal category.

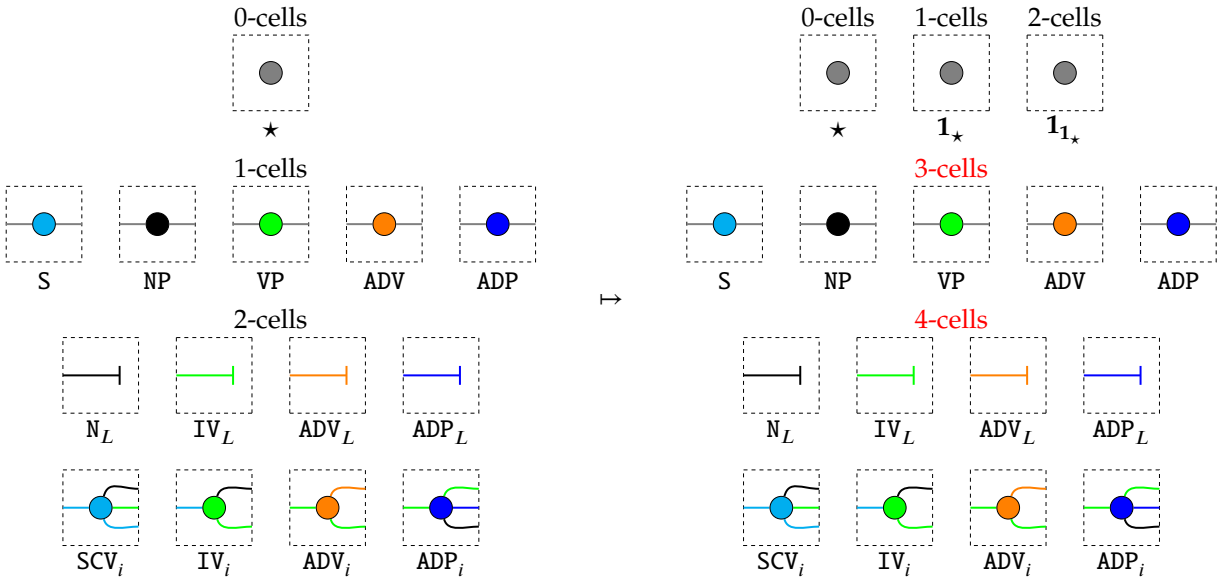


Figure 18: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with free swapping of symbol order.

0.1.5 TAGs with links

Definition 0.1.7 (TAGs with links: CS-style)**.** The following is a reproduction of the discussion on nodes with links from [Joshi]. Linking can be defined for any two nodes in an elementary tree. However, in the linguistic context we will require the following conditions to hold for a link in an elementary tree. If a node n_1 is linked to a node n_2 then:

1. n_2 *c-commands* n_1, (i.e., n_2 is not an ancestor of n_1, and there exists a node m which is the immediate parent node of n_2, and an ancestor of n_1).
2. n_1 and n_2 have the same label.
3. n_1 is the parent only of a null string, or terminal symbols.

A *TAG with links* is a TAG in which some of the elementary trees may have links as defined above.

Now we have enough to spell out full TAGs with local constraints and links as an n-categorical signature. To recap briefly, we have seen that we can model the passage from CFGs to TAGs in the n-categorical setting, and then how selective and null adjoining rules by specifying endomorphisms on wires as the source of rewrites, and finally that we can model links in the symmetric monoidal setting. Maintaining planarity (which is important for a left-to-right order of terminals for spelling out a language) and modelling obligatory rewrites can be done by requiring that *finished* derivations are precisely those whose only twists are link-wires, and have no sources for obligatory adjoins.

Example 0.1.8. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n-categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out.

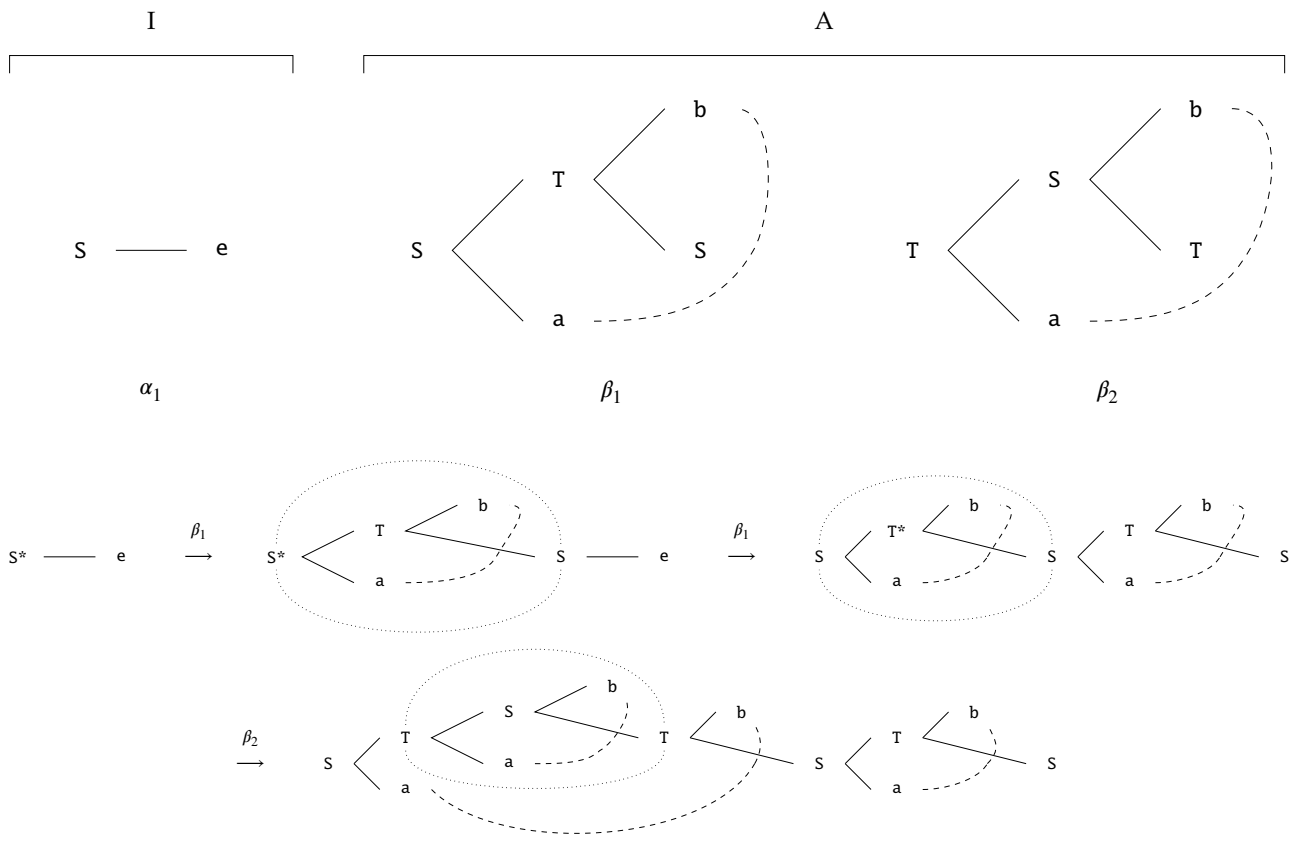


Figure 19: The TAG signature and example derivation are as above. Joshi stresses that adjoining *preserves* links, and that elementary trees may become *stretched* in the process of derivation, which are fundamentally topological constraints, akin to the "only (processive) connectivity" matters criterion identifying string diagrams up to isomorphism. Moreover, TAGs evidently have links of two natures: tree edges intended to be planar, and dashed dependency edges intended to freely cross over tree edges. It is easy, but a hack, to ask for planar processive isomorphisms for tree edges and extraplanar behaviour for dependency edges: these are evidently two different kinds of structure glued together, rather than facets of some whole. Weak n-categories offer a unified mathematical framework that natively accommodates the desired topological constraints while also granting expressive control over wire-types of differing behaviours. One method to recover TAGs true to the original conception is to stay in a planar 1-object-2-category setting while explicitly including wire-crossing cells for dependency links. The alternative method we opt for in Section ref is to work in a pure "only connectivity matters" setting, recovering the linear ordering of words by generating cells along a chosen wire. I do not know of any conceptual justification for why planarity is so often an implicit constraint in approaches to formal syntax. My best guesses are either that the first port of call for rewrites between 1-dimensional strings of words is a 2-dimensional setting, or it is a limitation of 2-dimensional paper as a medium of thought along with some confusion of map for territory.

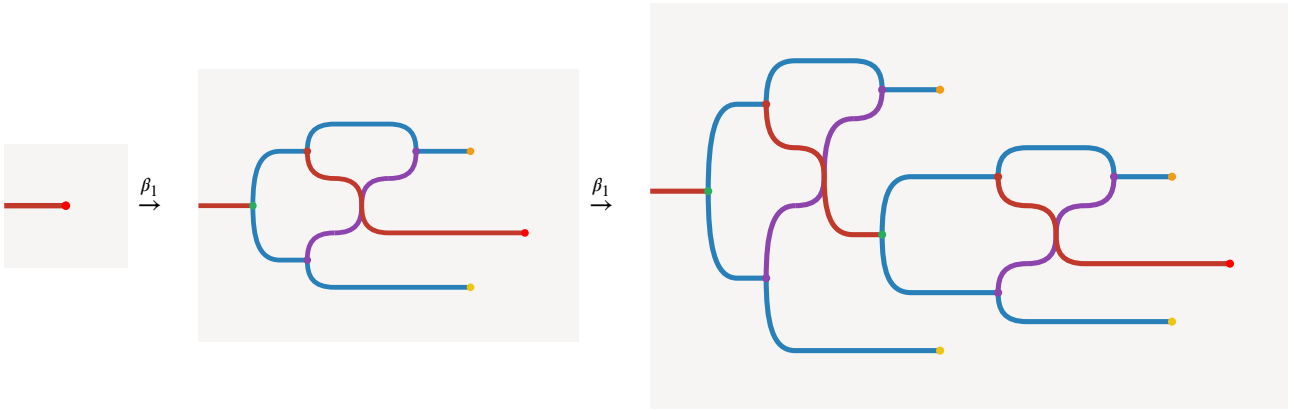


Figure 20: With our interpretation of TAGS as weak n-categorical signatures, We can recover each step of the example derivation automagically in `homotopy.io`; just clicking on where we want rewrites allows the proof assistant to execute a typematching tree adjunction. In the process of interpretation, we introduce a link wire-type (in purple), and include directed link generation and elimination morphisms for the T wire-type (in blue). A necessary step in the process of interpretation (which for us involves taking a Poincaré dual to interpret nodes as wires) is a typing assignment of the tree-branches connected to terminal nodes, which we have opted to read as sharing a T-type for minimality, though we could just as well have introduced a separate label-type wire.

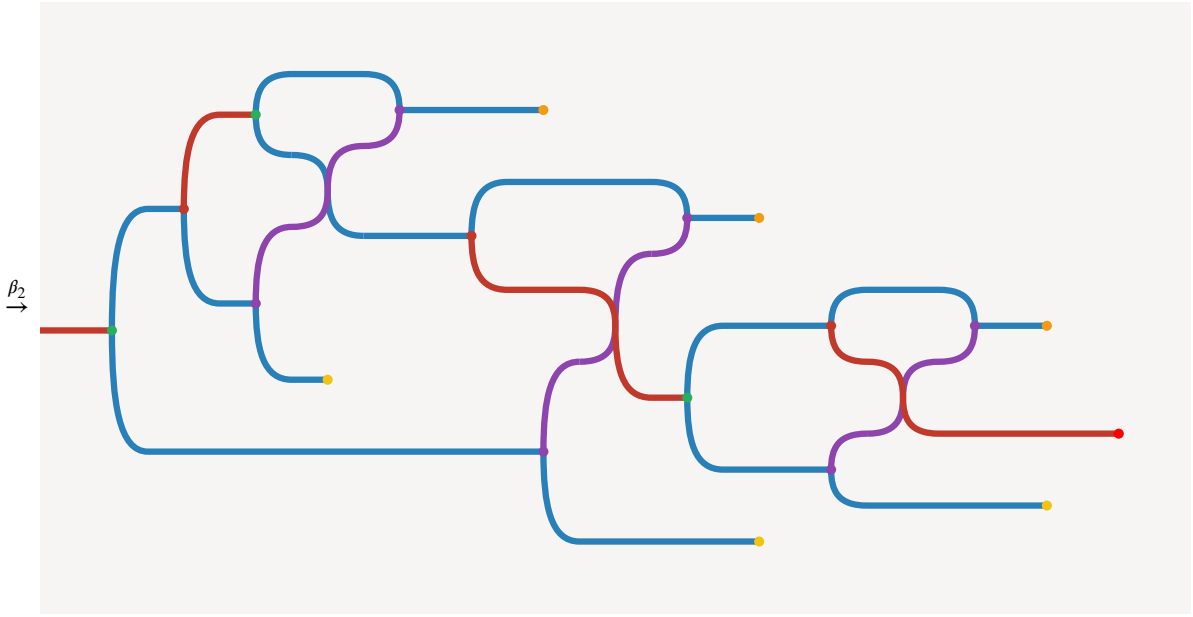


Figure 21: The intended takeaway is that even if you don't buy the necessity or formality of weak n-categories, there is always the fallback epistemic underpinning of a formal proof assistant for higher dimensional rewriting theories, which is rather simple to use if I have succeeded in communicating higher-dimensional intuitions in this section. **N.B.** In practice when using `homotopy.io` for the symmetric monoidal setting, it is simpler to suspend symmetric monoidal signatures to begin at 4-cells rather than 3-cells. The reason for this is that under- and over-braids still exist in the symmetric monoidal setting, and while sequentially composed braids are homotopically equivalent to the pair of identities, they are not uniquely so, thus these homotopies must be input manually. By beginning at 4-cells (or higher, due to the stabilisation hypothesis CITE), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

Definition 0.1.9 (Linguistic Tree Adjoining Grammars)**.** A *Linguistic Tree Adjoining Grammar* is given by the following data:

$$(\mathcal{N}, \mathcal{N}^{\downarrow}, \mathcal{N}^{*}, \Sigma, \mathcal{I}, \mathcal{A}, \mathfrak{S} \subseteq \mathcal{P}(\mathcal{A}), \square, \Diamond, \mathfrak{L} \subseteq \mathcal{N})$$

The initial elements are the same as an elementary tree-adjoining grammar and obey the same constraints. The modifications are:

- $\mathcal{I}$ is a nonempty set of *initial* constrained-linked-trees.
- $\mathcal{A}$ is a nonempty set of *auxiliary* constrained-linked-trees.
- $\mathfrak{S}$ is a set of sets of *select* auxiliary trees.
- $\square, \Diamond$ are fresh symbols. $\square$ marks *obligatory adjoins*, and $\Diamond$ marks *optional* adjoins.
- $\mathfrak{L}$ is a set permissible *link types* among nonterminals or $\top$.

A *constrained-linked-tree*, or CL-tree, is a pair consisting of:

- A tree where each internal node is an element of $\mathcal{N} \times \mathfrak{S} \times \{\square, \Diamond\} \times \{*, \bar{*}\}$, and each leaf is an element of $\mathcal{N} \times \mathfrak{S} \times \{\square, \Diamond\} \times \{*, \bar{*}\} \cup \Sigma$. In prose, each label is either a terminal symbol (as a leaf), or otherwise a nonterminal, along with a subset of auxiliary trees that indicates select adjoins (or null-adjoins when the subset is $\varnothing$), a marker indicating whether those adjoins are obligatory or optional, and a marker indicating whether the node is a foot node or not. Observe that there is no need to indicate when a node is a valid target for substitution since that function is subsumed by null adjoining rules.
- A set of ordered pairs of nodes (n_1, n_2) of the tree such that:
  1. n_2 *c-commands* n_1, (i.e., n_2 is not an ancestor of n_1, and there exists a node m which is the immediate parent node of n_2, and an ancestor of n_1).
  2. n_1 and n_2 share the same type $\mathbf{T} \in \mathcal{N}$ and $\mathbf{T} \in \mathfrak{L}$, or both n_1, n_2 are terminals.
  3. n_1 is the parent of terminal symbols, or childless.

0.1.6 Full TAGs in weak n-categories

I apologise in advance for the sheer ugliness of the following construction. Partly this is inevitable for combinatorial data, but the diagrammatic signatures also present combinatorial data and are easier to read, so what gives? This is what happens when one defines mathematical objects outside of their natural habitat. The native habitat of TAGs is diagrammatic, to accommodate a natural and spatially-intuitive generalisation of context-free grammars by allowing trees to be adjoined on nodes other than the leaves. I consider the offensiveness of the translation to follow a direct measure of the wrongness of linear-symbolic foundations, and an example of the harm that results from the prejudice that pictures cannot be formal.

Construction 0.1.10 (TAGs in `homotopy.io`). We spell out how the data of a TAG becomes an n-categorical signature by enumerating cell dimensions:

0. A single object $\star$
1. None.
2. None.
3. - For each $\mathbf{T} \in \mathcal{N}$, a cell $\mathbf{T} : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$.
   - $\top : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$ A wire for terminal symbols.
   - For each $\mathbf{L} \in \mathfrak{L}$, a cell $\mathbf{L} : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$.
4. We distinguish between *atomic* and *composite* generators at this level. The trees themselves are composites of *atomic* generators. The *atomic* generators are:
   - For each node n that occurs in either $\mathcal{I}$ or $\mathcal{A}$, we populate cells by a case analysis:
     - If n is a terminal $\sigma \in \Sigma$, we create a cell $\sigma : \top \to \mathbf{1}_{\mathbf{1}_\star}$.
     - Where $\phi \in \mathfrak{S}$ denotes a selective adjoining rule where required, if $n = (\mathbf{T}, \mathbf{S}, \dagger, \bar{*})$, we create $\phi\mathbf{S}^{\square}_{\mathbf{T}} : \mathbf{T} \to \mathbf{T}$ if the node is marked obligatory ($\dagger = \square$), and a cell $\phi\mathbf{S}^{\Diamond}_{\mathbf{T}} : \mathbf{T} \to \mathbf{T}$ otherwise.
     - If $n = (\mathbf{T}, \mathbf{S}, \dagger, *)$, it is a foot node, for which we create a cell $\mathbf{S}^{\square}_{\mathbf{T}} : \mathbf{T} \to \mathbf{1}_{\mathbf{1}_\star}$ if the node is marked obligatory ($\dagger = \square$), and a cell $\mathbf{S}^{\Diamond}_{\mathbf{T}} : \mathbf{T} \to \mathbf{1}_{\mathbf{1}_\star}$ otherwise.
   - For each $\mathbf{L} \in \mathfrak{L}$ (which is also a type $\mathbf{T} \in \mathcal{N}$) a pair of cells $\mathbf{T}^{\mathbf{L}} := \mathbf{T} \to \mathbf{T} \otimes \mathbf{L}$ and $\mathbf{T}_{\mathbf{L}} := \mathbf{L} \otimes \mathbf{T} \to \mathbf{T}$.
   - For each node p of type $\mathbf{T}_p$ in either $\mathcal{I}$ or $\mathcal{A}$ with a nonempty left-to-right list of children $C_p :=< c_1, c_2, \cdots c_i, c \ldots c_n >$ with types $\mathbf{T}_i$, a branch cell $C_p : \mathbf{T}_p \to \bigotimes_{i=1}^{n} \mathbf{T}_i$.

   We represent trees by composite generators, defined recursively. For a given tree $\mathcal{T}$ in either $\mathcal{I}$ or $\mathcal{A}$, we define a composite generator beginning at the root. Where the root node is $p = (\mathbf{T}, \mathbf{S}, \dagger)$, we begin with the cell $\mathbf{S}^{\dagger}_{\mathbf{T}}$. For branches, we compose the branch cell C_p to this cell sequentially. If p has a child c that has a link, we do a case analysis. If that child c-commands the other end of the link we generate the first half of the linking wire by composing $\mathbf{T}^{\mathbf{L}}$ for the appropriate type $\mathbf{T}$ of the child node. Otherwise the child is c-commanded by a previously generated link, which we braid over and connect using $\mathbf{T}_{\mathbf{L}}$, again with the appropriate typing for the child. Now we may recurse the procedure for subtrees. If a node has no children, it is a leaf $l = (\mathbf{T}, \mathbf{S}, \dagger)$ or a terminal symbol. We append a terminal cell σ if l is a terminal symbol (thus killing the wire), and otherwise we leave an open $\mathbf{T}$ wire after appending $\mathbf{S}^{\dagger}_{\mathbf{T}}$. Altogether this obtains a 3-cell which we denote $\mathcal{T}$, overloading notation; different execution-orders of the above procedure evidently obtain 3-cells equivalent up to homotopy.
5. For each $\phi\mathbf{S}^{\square}_{\mathbf{T}}$, $\phi\mathbf{S}^{\Diamond}_{\mathbf{T}}$, and for each typematching 4-cell tree in the subset of select adjoins of ϕ, we create a 5-cell rewrite that performs tree adjoining, taking the former to be the source and the latter to be the target.

A derivation is finished when there are no obligatory adjoin nodes, all leaves are terminal symbols, and homotopies are applied such that only dependency link-wires participate in braidings, which implies that the tree-part is planar.

0.2 A generative grammar for text circuits

0.2.1 A circuit-growing grammar

There are many different ways to write a weak n-categorical signature that generates circuits. Mostly as an illustration of expressivity, I will provide a signature where the terms "surface" and "deep" structure are taken literally as metaphors; the generative grammar will grow a line of words in syntactic order, and like mushrooms on soil, the circuits will behave as the mycelium underneath the words. It won't be the most efficient way to do it in terms of the number of rules to consider, but it will look nice and we'll be able to reason about it easily.

Definition 0.2.1 (Lexicon)**.** We define a limited lexicon $\mathcal{L}$ to be a tuple of disjoint finite sets $(\mathbf{N}, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_S, \mathbf{A}_N, \mathbf{A}_V, \mathbf{C})$

Where:

- $\mathbf{N}$ is a set of *proper nouns*
- $\mathbf{V}_1$ is a set of *intransitive verbs*
- $\mathbf{V}_2$ is a set of *transitive verbs*
- $\mathbf{V}_S$ is a set of *sentential-complement verbs*
- $\mathbf{A}_N$ is a set of *adjectives*
- $\mathbf{A}_V$ is a set of *adverbs*
- $\mathbf{C}$ is a set of *conjunctions*

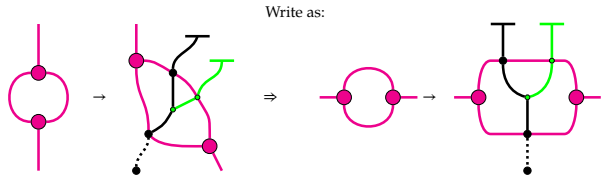


Figure 22: **How to read the diagrams in this section:** we will be making heavy use of pink and purple bubbles as frames to construct circuits. We will depict the bubbles horizontally, as we are permitted to by compact closure, or by reading diagrams with slightly skewed axes.

Simplifications and limitations: For now we only consider word types as in Definition 0.2.1, though we will see how to engineer extensions later. We only deal with propositional statements, without determiners, in only one tense, with no morphological agreement between nouns and their verbs and referring pronouns, and we assume that adverbs, adjectives stack indefinitely and without further order requirements: e.g. `Alice happily secretly finds red big toy shiny car that he gives to Bob` is a sentence we consider grammatical enough. For now, we consider only the case where adjectives and adverbs appear before their respective noun or verb. Note that all of these limitations apart from the limited lexicon can principle be overcome by the techniques we developed in Section **??** for restricted tree-adjoining and links. As a historical remark, generative-transformational grammars fell out of favour linguistically due to the problem of overgeneration: the generation of nonsense or unacceptable sentences in actual language use. We're undergenerating and overgenerating at the same time, but we're also not concerned with empirical capture: we only require a concrete mathematical basis to build interesting things on top of. On a related note, there's zero chance that this particular circuit-growing grammar even comes close to how language is actually produced by humans, and I have no idea whether a generalised graph-rewriting approach is cognitively realistic.

Mathematical assumptions: We work in a dimension where wires behave symmetric monoidally by homotopy, and further assume strong compact closure rewrite rules for all wire-types. Our strategy will be to generate "bubbles" for sentences, within which we can grow circuit structure piecemeal. We will only express the rewrite rules; the generators of lower dimension are implicit. We aim to recover the linear ordering of words in text (essential to any syntax) by traversing the top surface of a chain of bubbles representing sentence structure in text – this order will be invariant up to compact closed isomorphisms. The diagrammatic consequence of these assumptions is that we will be working with a conservative generalisation of graph-rewriting defined by local rewriting rules. The major distinction is that locality can be redefined up to homotopy, which allows locally-defined rules to operate in what would be a nonlocal fashion in terms of graph neighbourhoods, as in Figure 23. The minor distinction is that rewrite rules are sensitive to twists in

wires and the radial order in which wires emanate from nodes, though it is easy to see how these distinctions can be circumvented by additional by imposing the equivalent of commutativity relations as bidirectional rewrites. It is worth remarking that one can devise weak n-categorical signatures to simulate turing machines, where output strings are e.g. 0-cells on a selected 1-cell, so rewrite systems of the kind we propose here are almost certainly expressively sufficient for anything; the real benefit is the interpretable geometric intuitions of the diagrams.

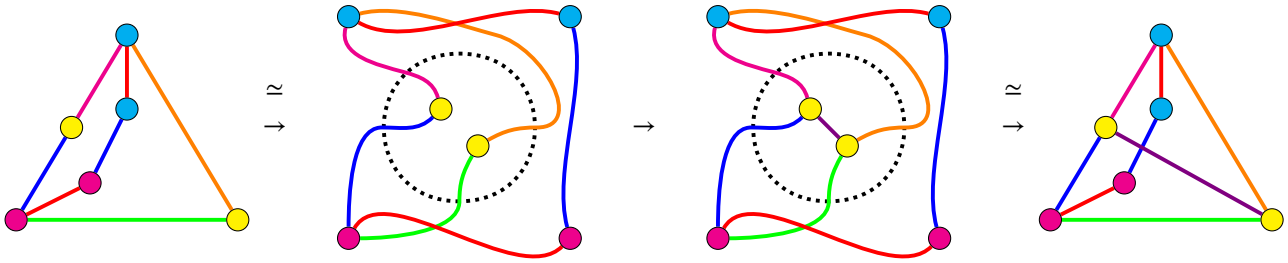
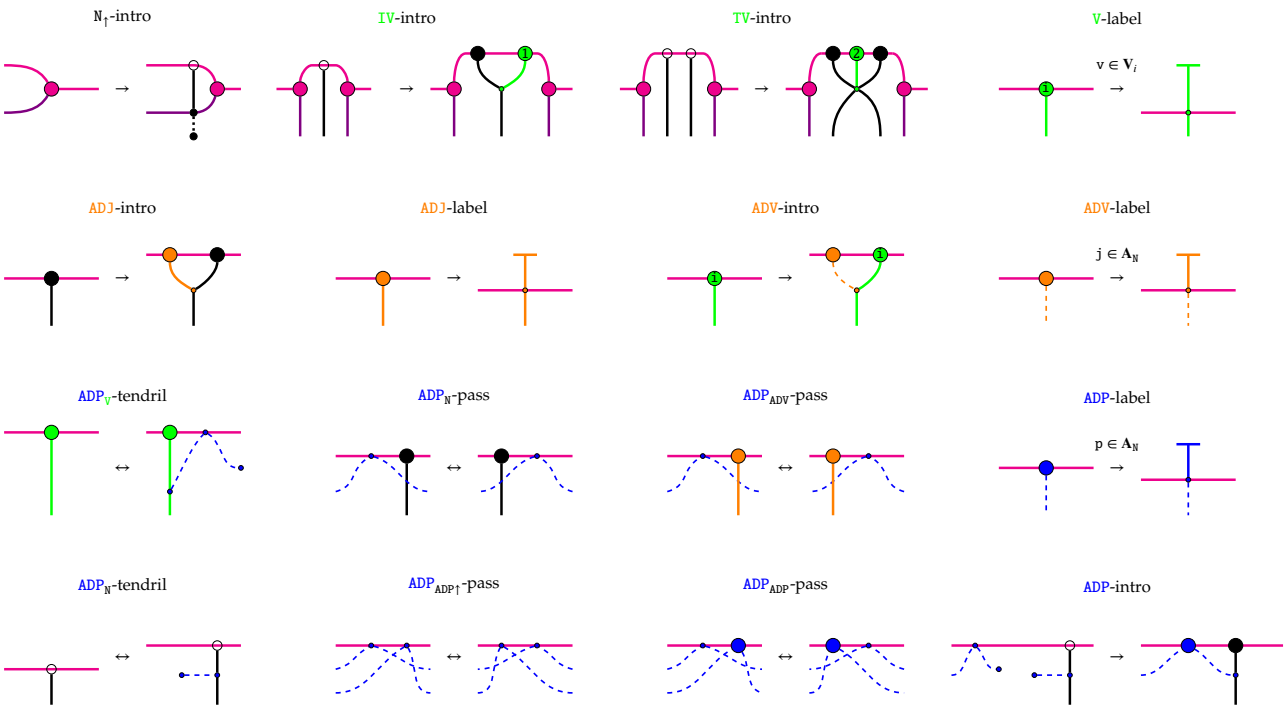


Figure 23: In this toy example, obtaining the same rewrite that connects the two yellow nodes with a purple wire using only graph-theoretically-local rewrites could potentially require an infinite family of rules for all possible configurations of pink and cyan nodes that separate the yellow, or would otherwise require disturbing other nodes in the rewrite process. In our setting, strong compact closure homotopies handle navigation between different spatial presentations so that a single rewrite rule suffices: the source and target notated by dotted-black circles. Despite the expressive economy and power of finitely presented signatures, we cannot "computationally cheat" graph isomorphism: formally we must supply the compact-closure homotopies as part of the rewrite, absorbed and hidden here by the $\simeq$ notation.

The plan: We start with simple sentences that only contain a single intransitive or transitive verb. Then we consider more general sentences. For these two steps, we characterise the expressive capacity of our rules in terms of a context-sensitive grammar that corresponds to the surface structure of the derivations. Then we introduce text structure as lists of sentences with coreferential structure on nouns, along with a mathematical characterisation of coreferential structure and a completeness result of our rules with respect to them. Then we (re)state and prove the text circuit theorem: that the fragment of language we have built with the syntax surjects onto text circuits. Finally we examine how we may model extensions to the expressive capacity of text circuits by introduction of new rewrite rules.

0.2.2 *Simple sentences*

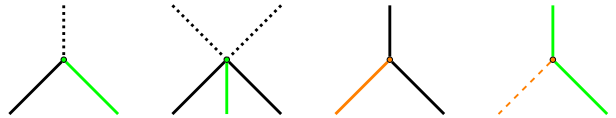
Simple sentences are sentences that only contain a single intransitive or transitive verb. Simple sentences will contain at least one noun, and may optionally contain adjectives, adverbs, and adpositions. The rules for generating simple sentences are as follows:



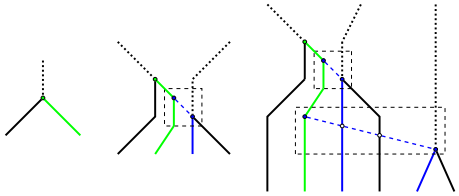
The $\mathtt{N}_\uparrow$-intro rule introduces new unsaturated nouns from the end of a simple sentence. The IV-intro rule applies when there is precisely one unsaturated noun in the sentence, and the TV-intro rule applies when there are precisely two. Both verb-introduction rules saturate their respective nouns, which we depict with a black bulb. Adjectives may be introduced immediately preceding saturated nouns, and adverbs may be introduced immediately preceding any kind of verb. The position of adpositions in English is context-sensitive. To capture this, the $\mathtt{ADP}_\mathtt{V}$-tendril rule allows an unsaturated adposition to appear immediately after a verb; a bulb may travel by homotopy to the right, seeking an unsaturated noun. Conversely, the bidirectional $\mathtt{ADP}_\mathtt{N}$-tendril rule sends a mycelic tendril to the left, seeking a verb. The two pass-rules allow unsaturated adpositions to swap past saturated nouns and adjectives; note that by construction, neither verbs nor adverbs will appear in a simple sentence to the right of a verb, so unsaturated adpositions will move right until encountering an unsaturated noun. In case it doesn't, the tendril- and pass- rules are bidirectional and reversible.

Definition 0.2.2 (CSG for simple sentences)**.** We may gauge the expressivity of simple sentences with the following context sensitive grammar.

For verbs, adjectives, and modifiers, depicted unsaturated nouns as dotted and saturated with solid black lines, we have:

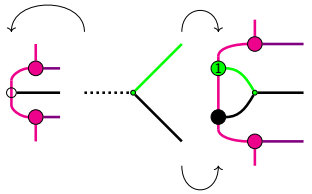


Adpositions require several helper-generators; we depict for example the beginning of the sequence of derivations that result from appending adpositions to an intransitive verb (the generators are implicit in the derivations):



Proposition 0.2.3. Up to labels, the simple-sentence rules yield the same simple sentences as the CSG for simple sentences.

Proof. By graphical correspondence; viewing nodes on the pink surface as 1-cells, each rewrite rule yields a 2-cell. For example, for the IV-intro:



□

0.2.3 *Complex sentences*

Now we consider two refinements; conjunctions, and verbs that take sentential complements. we may have two sentences joined by a conjunction, e.g. `Alice dances while Bob drinks`. We may also have verbs that take a sentential complement rather than a noun phrase, e.g. `Alice sees Bob dance`; these verbs require nouns, which we depict as wires spanning bubbles.

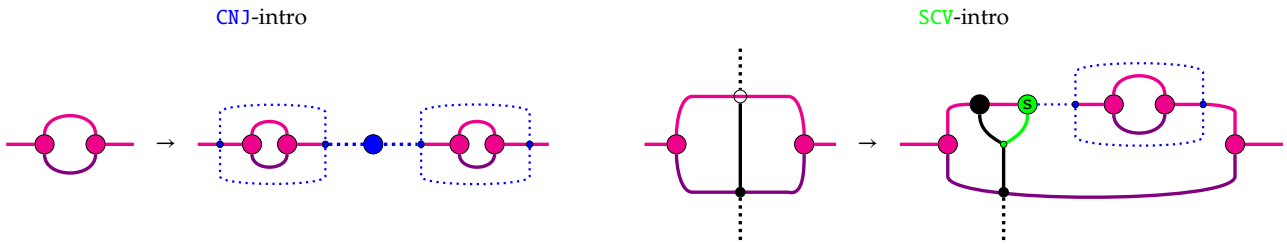
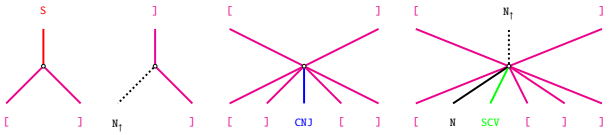


Figure 24: The dotted-blue wires do not contentfully interact with anything else, but this noninteraction disallows overgeneration cases where adpositional phrases might interject between SCV verbs and their sentential complement, e.g. `Alice sees at lunch Bob drink`. The dotted-blue wires also indicate a diagrammatic strategy for extensions to accommodate noun phrases, to be explored later.

Definition 0.2.4 (Sentence structure)**.** A sentence can be:

- a simple sentence, which...
- ... may generate unsaturated nouns from the right.
- a pair of sentences with a conjunction in between.
- (if there is a single unsaturated noun) a sentence with a sentential-complement verb that scopes over a sentence.

As a CSG, these considerations are respectively depicted as:



Proposition 0.2.5. Up to labels, the rules so far yield the same sentences as the combined CSG of Definitions 0.2.2 and 0.2.4.

Proof. Same correspondence as Proposition 0.2.3, ignoring the dotted-blue guards. □

Figure 25:

Example 0.2.6 (`sober` α `sees drunk` β `clumsily dance.`). Now we can see our rewrites in action for sentences. As a matter of convention – reflected in how the various pass- rules do not interact with labels – we assume that labelling occurs after all of the words are saturated. We have still not introduced rules for labelling nouns: we delay their consideration until we have settled coreferential structure. For now they are labelled informally with greeks.

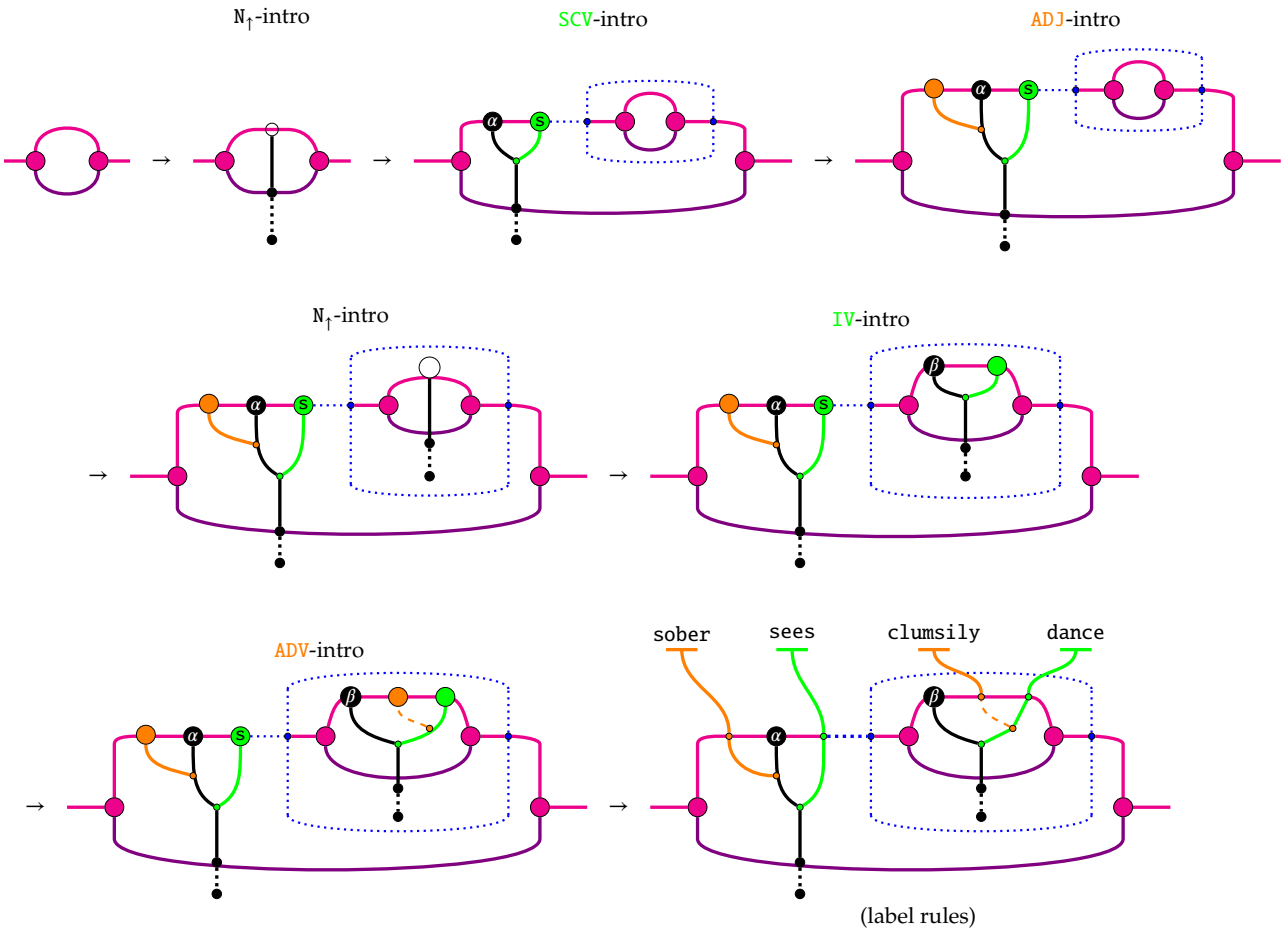
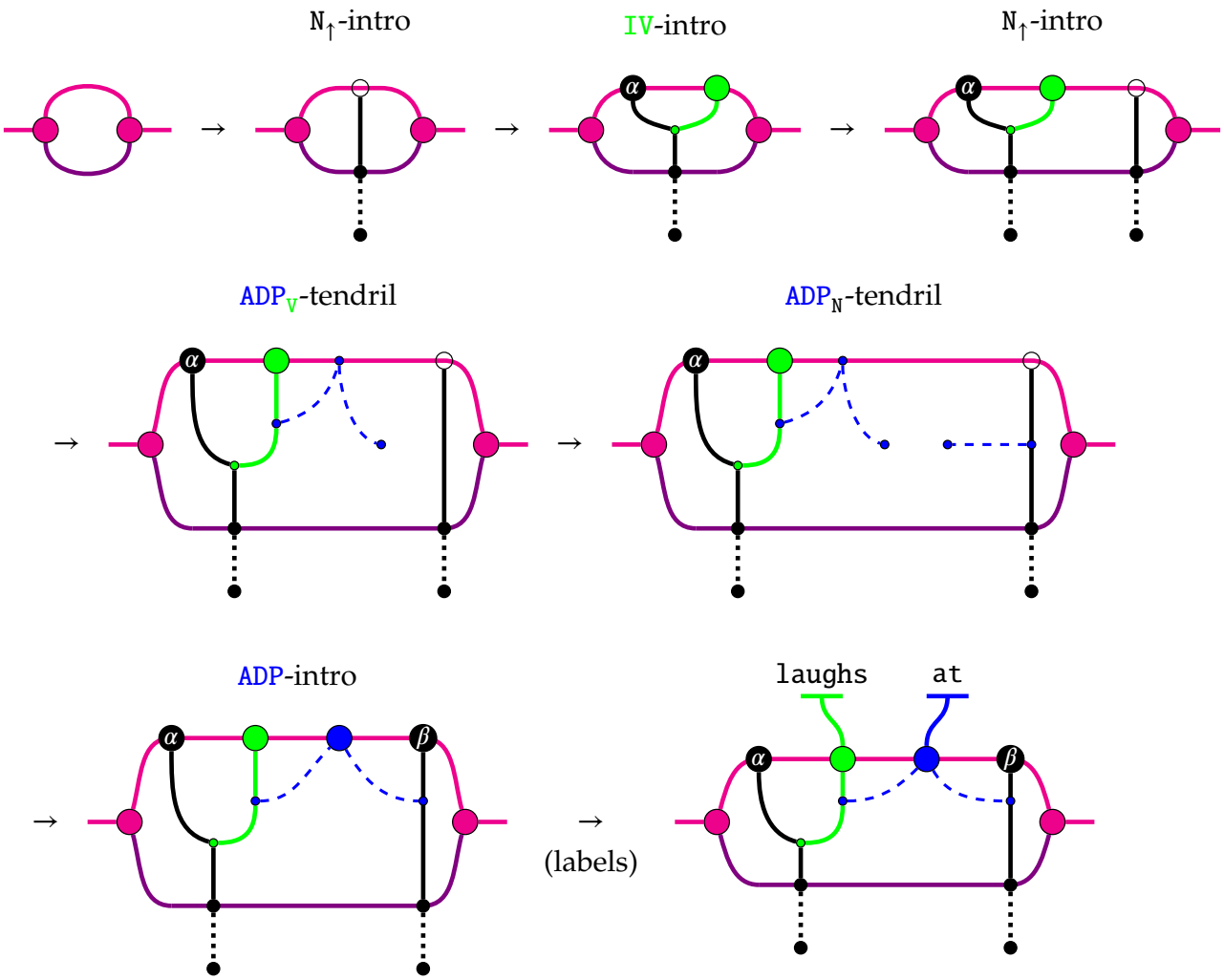


Figure 26:

Example 0.2.7 (α `laughs at` β)**.** Adpositions form by first sprouting and connecting tendrils under the surface. Because the tendril- and pass- rules are bidirectional, extraneous tendrils can always be retracted, and failed attempts for verbs to find an adpositional unsaturated noun argument can be undone. Though this seems computationally wasteful, it is commonplace in generative grammars to have the grammar overgenerate and later define the set of sentences by restriction, which is reasonable so long as computing the restriction is not computationally hard. In our case, observe that once a verb has been introduced and its argument nouns have been saturated, only the introduction of adpositions can saturate additionally introduced unsaturated nouns. Therefore we may define the finished sentences of the circuit-growing grammar to be those that e.g. contain no unsaturated nodes on the surface, which is a very plausible linear-time check by traversing the surface.



0.2.4 *Text structure and noun-coreference*

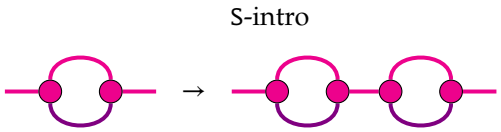


Figure 27: Only considering words, text is just a list of sentences. However, for our purposes, text additionally has *coreferential structure*. Ideally, we would like to connect "the same noun" from distinct sentences as we would circuits.

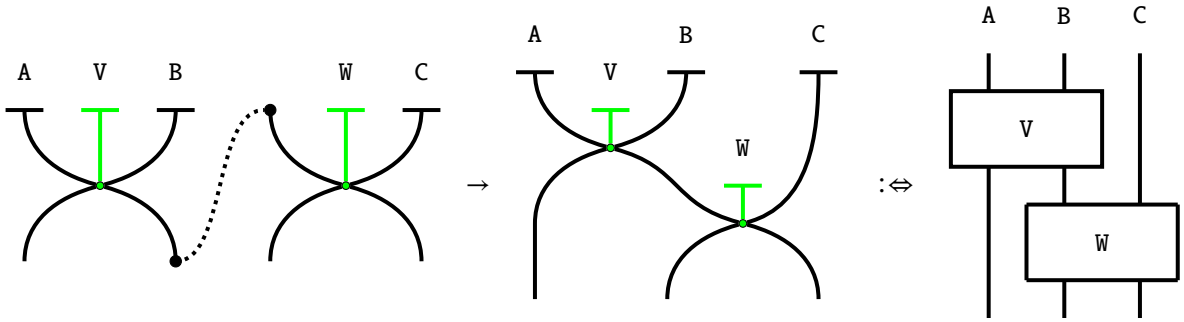


Figure 28: We choose the convention of connecting from left-to-right and from bottom-to-top, so that we might read circuits as we would text: the components corresponding to words will be arranged left-to-right and top-to-bottom. Connecting nouns across distinct sentences presents no issue, but a complication arises when connecting nouns within the same sentence as with reflexive pronouns e.g. `Alice likes herself`.

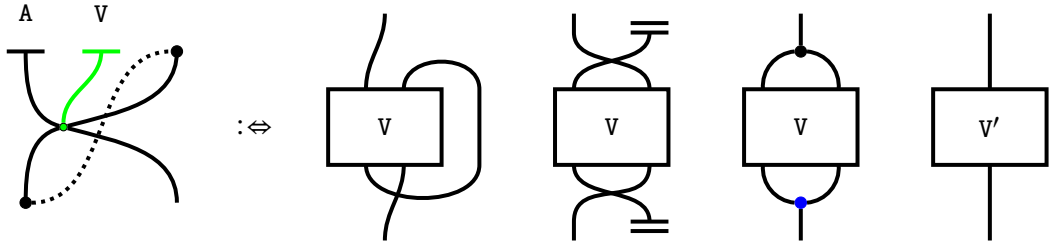
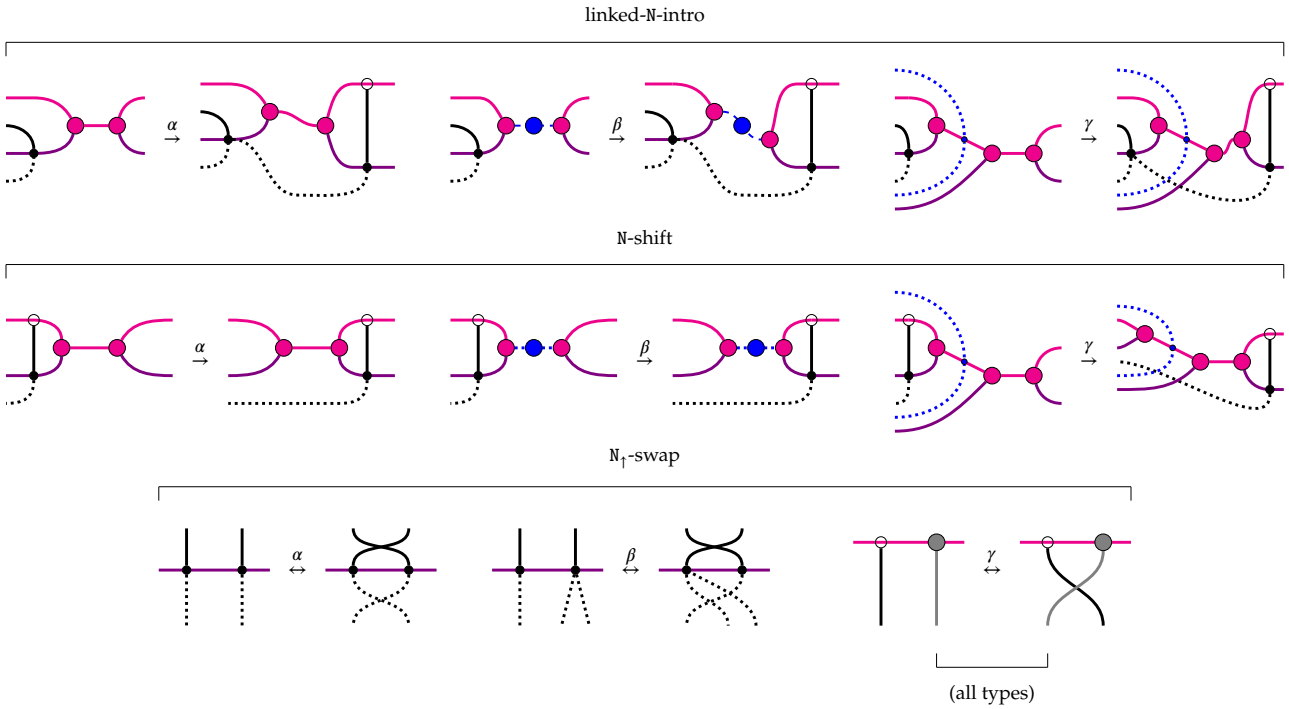


Figure 29: Reflexive coreference would violate of the processivity condition of string diagrams for symmetric monoidal categories. Not all symmetric monoidal categories possess the appropriate structure to interpret such reflexive pronouns, but there exist interpretative options. From left to right in roughly decreasing stringency, compact closed categories are the most direct solution. More weakly, traced symmetric monoidal categories also suffice. If there are no traces, so long as the noun wire possesses a monoid and comonoid, a convolution works. If all else fails, one can just specify a new gate. We will define coreference structure to exclude such reflexive coreference and revisit the issue as an extension.

Now we will deal with coreferential structure and noun-labels. TODO: need more linked-intro variants to handle leaving right-side of conjunction, nested SCV, CNJ to SCV, and SCV to CNJ. Consider using the same graywire convention to simplify case analysis?



The linked-N-intro rules introduce a new unsaturated noun in the next sentence that coreferences the noun in the previous sentence that generated it. The N-shift rules allow any unsaturated noun to move into the next sentence. For both of the previous rules, the β variant handles the case where the next sentence is related to the first by a conjunction. Observe that nouns with a forward coreference have two dotted-black wires leaving the root of their wires, which distinguishes them from nouns that only have a backward coreference or no coreference at all, which only have a single dotted-black wire leaving the root of their wire.

The N-swap rule variants allow a unsaturated noun with no forward coreferences to swap places with any unsaturated noun that immediately succeeds it.

When the structure of coreferences is set, we propagate noun labels from the head of each list. The rules for noun-label propagation are as follows:

Example 0.2.8 (`sober Alice sees Bob clumsily dance.  She laughs at him.`).

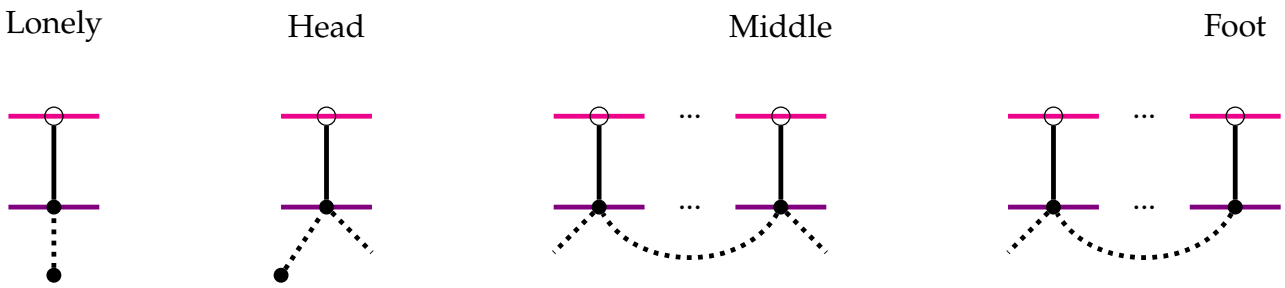


Figure 30: At this point, it is worth establishing some terminology about the kinds of unsaturated nouns we have in play. The kinds of nouns are distinguished by their tails. *Lonely* nouns have no coreferences, their tails connect to nothing. *Head* nouns have a forward coreference in text; they have two tails, one that connects to nothing and the other to a noun later in text. *Middle* nouns have a forward and backward coreference; they have two tails, one that connects to a noun in some preceding sentence, and one that connects forward to a noun in a succeeding sentence. *Foot* nouns only have a backward coreference; they have a single tail connecting to a noun in some preceding sentence.

0.2.5 Text circuit theorem

Definition 0.2.9 (Text Circuits)**.** *Text circuits* are made up of three ingredients:

- wires
- boxes, or gates
- boxes with holes that fit a box, or 2nd order gates

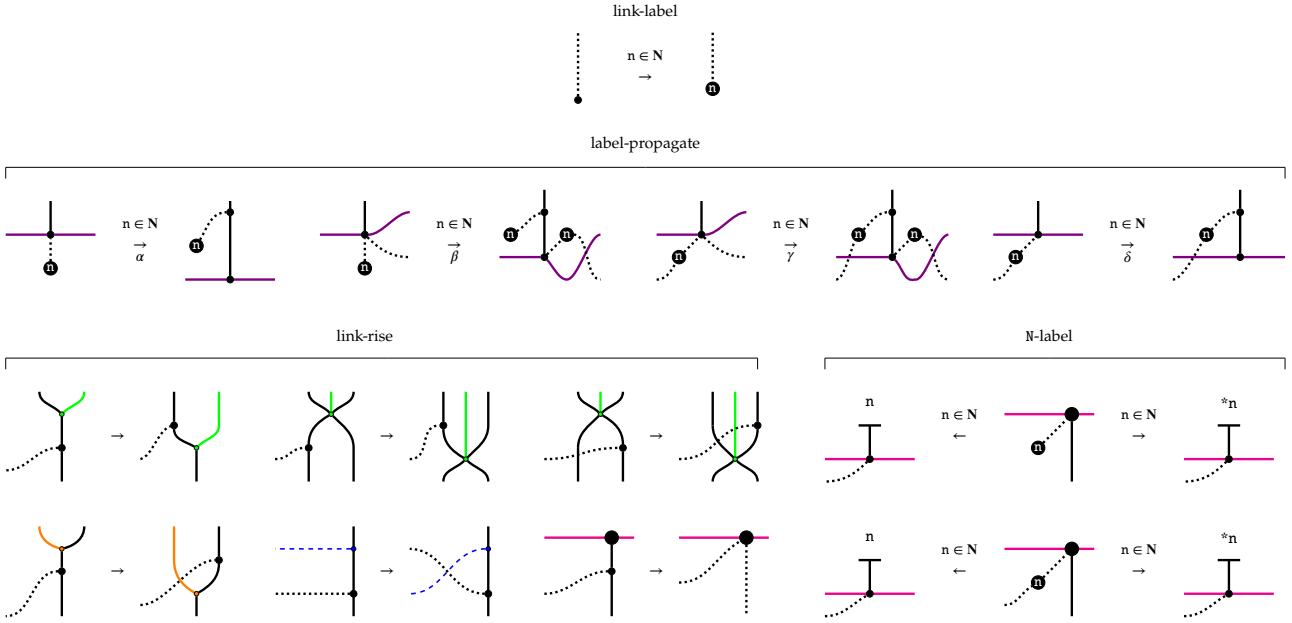


Figure 31: The $n \in \mathbf{N}$ notation indicates a family of rewrites (and generators) for each noun in the lexicon. Link-label assigns a noun to a diagrammatically linked collection of coreferent nouns, and link-propagation is a case analysis that copies a link label and distributes is across coreferent nouns. Link-rise is a case analysis to connect labels to the surface, and finally N-label allows a saturated noun to inherit the label of its coreference class, which may either be a noun n or a pronoun appropriate for the noun, notated *n

Figure 32: We start the derivation by setting up the sentence structure using S- and SCV-intro rules, and two instances of N-intro, one for Alice, and one for Bob. Observe how the N-intro for Bob occurs within the sub-sentence scoped over by the SCV-rule.

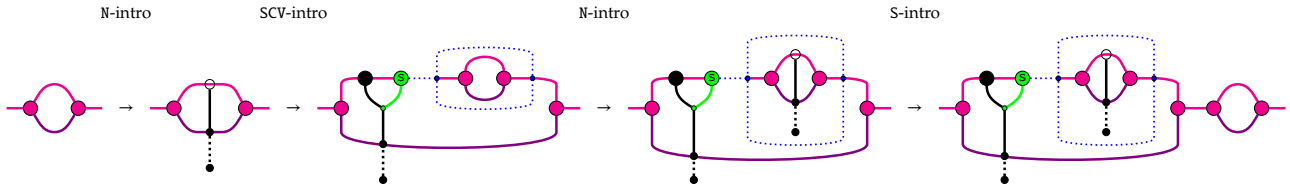


Figure 33: By homotopy, we can rearrange the previous diagram to obtain the source of the linked-N-intro rewrite in the dashed-box visual aid. Observe how we drag in the root of what is to be Alice's wire. Then we use the IV-intro in the second sentence, which sets up the surface structure `she laughs`, and the deep structure for bookkeeping that `she` refers to `Alice`.

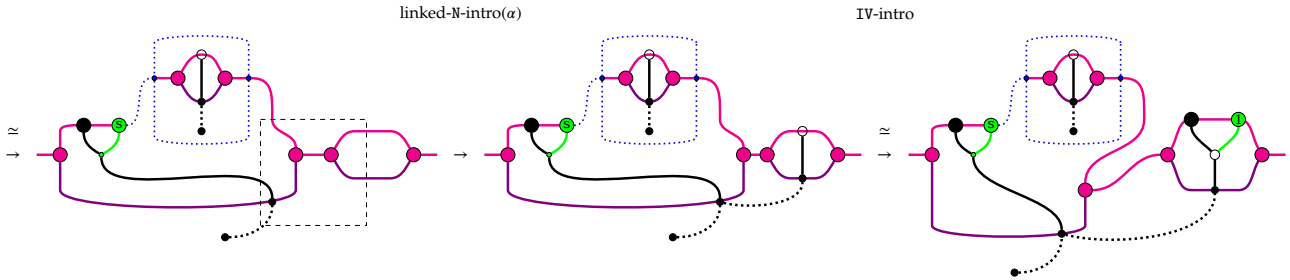


Figure 34: By homotopy again, we can do the same for Bob, this time setting up for the γ variant of linked-N-intro which handles the case when the spawning noun is within the scope of an SCV. Then by applying a series of $\mathtt{N}_{\uparrow}$-swaps, the unsaturated noun is placed to the right of the intransitive verb phrase.

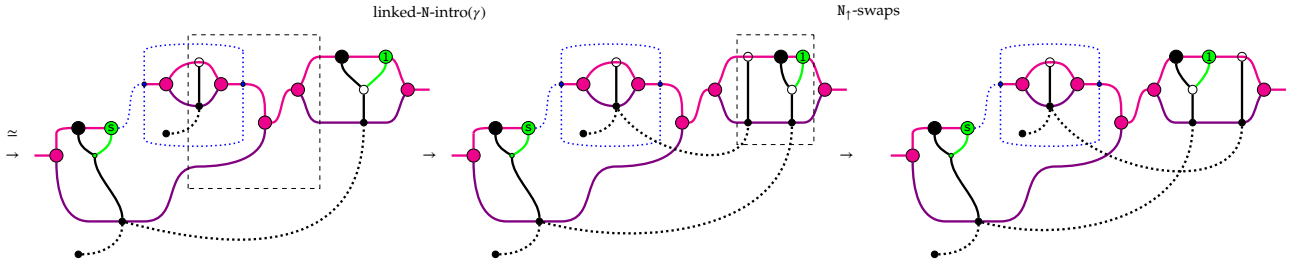
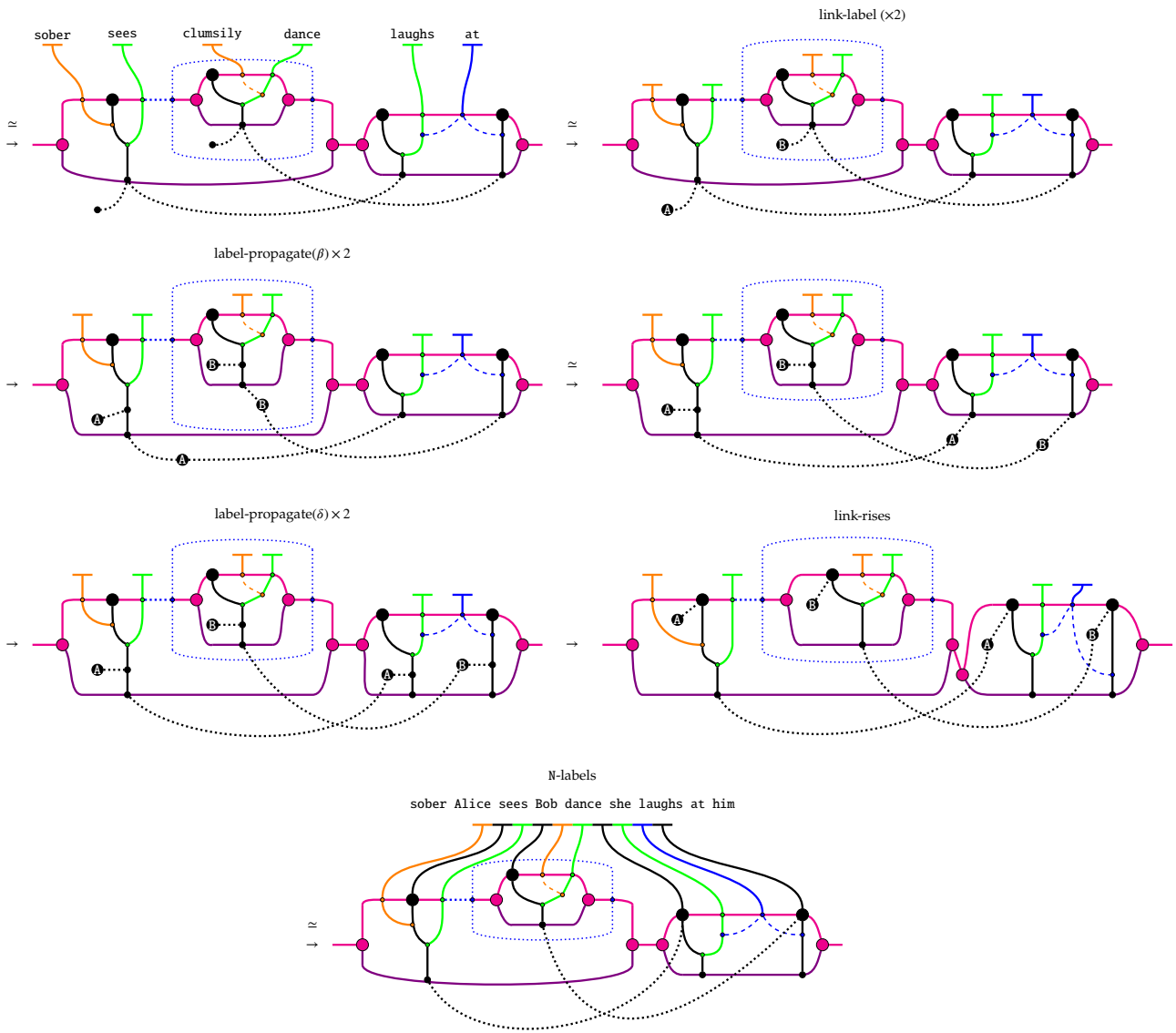


Figure 35: We've already done the surface derivation for the two sentences separately in Figures 25 and 26; since neither of those derivations touch the roots of noun-wires, we can emulate those derivations and skip ahead to the first diagram.



Now we demonstrate that *finished* text diagrams yield unique text circuits up to homotopy. Text circuits are presented again in the margins as a reminder. The strategy will be to present new rewrites that transform coreferential structure into symmetric monoidal wire-connectivity.

Definition 0.2.10 (Finished text diagram)**.** The circuit-growing grammar produces *text diagrams*. We call a text diagram *finished* if all surface nodes are labelled.

Proposition 0.2.11. Finished text diagrams yield text, up to interpreting distinct sentences as concatenated with punctuation `.`, `,`, contentless conjunctions or complementisers – such as `and`, or `that` respectively.

Proof. Sentence-wise grammaticality is gauged by Propositions 0.2.3 and 0.2.5. When multiple sentences occur within the scope of a `SCV` we might prefer the use of contentless complementisers and conjunctions, e.g. `Alice sees( Bob draws Charlie drinks ) Dennis dances` is grammatically preferable but meaningfully equivalent to `Alice sees that Bob draws and Charlie drinks , and Dennis dances .` For our purposes it makes no difference whether surface text has these decorations, as the deep structure of text diagrams encodes all the information we care to know. □

Lemma 0.2.12. The unsaturated noun kinds listed in Figure 30 are exhaustive, hence nouns that share a coreference are organised as a diagrammatic linked-list.

Proof. The `N`-intro rule creates lonely nouns. Head nouns can only be created by the linked-`N`-intro applied to a lonely noun. Any new noun created by linked-`N`-intro is a foot noun. The linked-`N`-intro rule turns foot nouns into middle nouns. These two intro- rules are the only ones that introduce unsaturated nouns, so it remains to demonstrate that no other rules can introduce noun-kinds that fall outside our taxonomy. The `N`-shift rule changes relative position of either a lonely or foot noun but cannot change its kind. The `N`-swap rule may start with either a lonely or foot noun on the left and either a head or middle noun on the right, but the outcome of the rule cannot change the starting kinds as tail-arity is conserved and the local nature of rewrites cannot affect the ends of tails. □

Lemma 0.2.13. No nouns within the same sentence are coreferentially linked.

Proof. Novel linked nouns can only be obtained from the linked-`N`-intro rule, which places them in succeeding sentences. The swap rules only operate within the same sentence and keep the claim invariant. The `N`-shift rules only apply to nouns with no forward coreferences; nouns with both forward and backward coreferences cannot leave the sentence they are in. Moreover, `N`-shift is unidirectional and only allows the rightmost coreference in a linked-list structure to move to later sentences. So there is no danger of an `N`-shift breaking the invariant. □

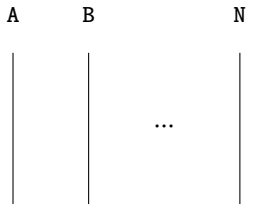


Figure 36: Nouns are represented by wires, each 'distinct' noun having its own wire.

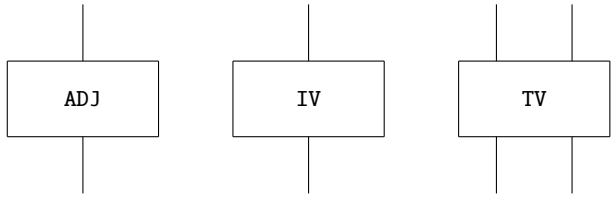


Figure 37: We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires. Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

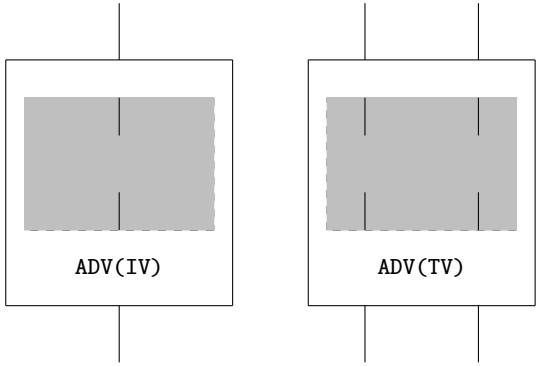


Figure 38: Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicating the shape of gate expected, and these should match the input- and output-wires of the box with the whole.

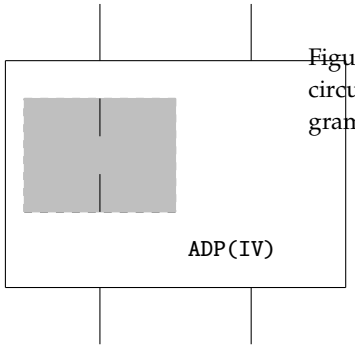


Figure 39: Similarly, adpositions also modify verbs, by moreover adding another noun-wire to the right.

Construction 0.2.14 (Text to circuit)**.**

Figure 44: We turn finished text diagrams into text circuits by operating *in situ*, with extra rules outside the grammatical system that handle connecting noun wires.

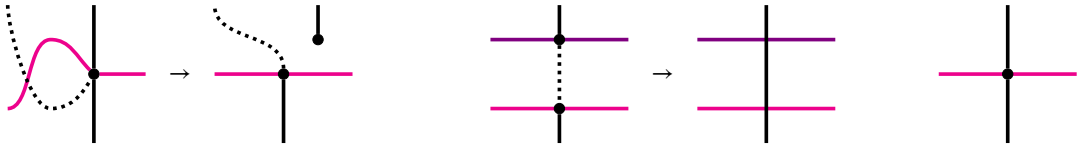


Figure 45: In the first step, by Lemmas 0.2.12 and 0.2.13, we can always rearrange a finished text diagram such that the noun wires are processive.

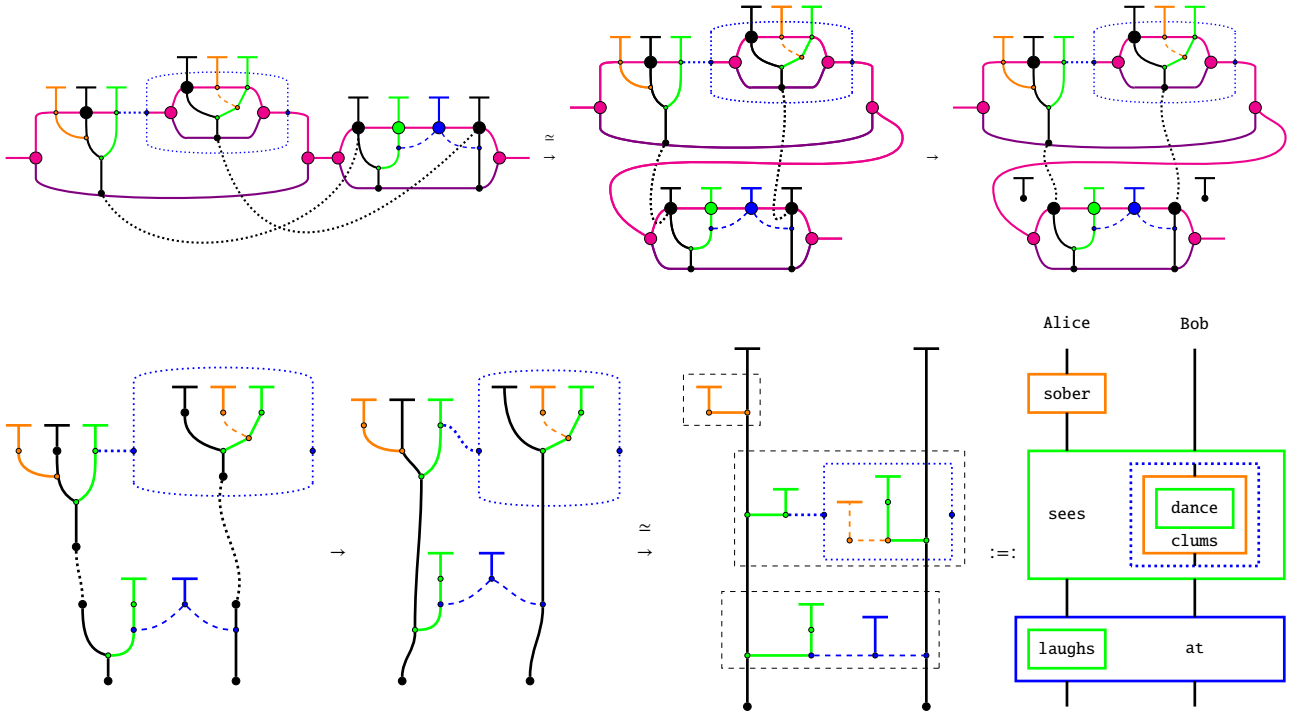
In the second step, use the first rewrite of Construction 0.2.14 to prepare the wires for connection.

In the third step, we just ignore the existence of the bubble-scaffolding and the loose scalars. We could in principle add more rewrites to melt the scaffolding away if we wanted, but who cares? ...

In the fourth step, we apply the second and third rewrites of Construction 0.2.14 to connect the wires and eliminate nodules underneath labels. We can also straighten up the wires a bit and make them look proper.

At this point, we're actually done, because the resulting diagram *is already a text circuit up to a choice of notation*.

Figure 40: For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit.



Proposition 0.2.15 (Finished text diagrams yield unique-up-to-processive-isotopy text circuits)**.** *Proof.* Every sentence corresponds to a gate up to notation, and we have a handle on sentences via Propositions 0.2.3 and 0.2.5. Lemmas 0.2.12 and 0.2.13 guarantee processivity. Uniqueness-up-to-processive-isotopy is inherited: text diagrams themselves are already specified up-to-connectivity, which is strictly more general than processive isotopy. Therefore, for any circuit C obtained from a text diagram T by Construction 0.2.14, T can be

modified up to processive-isotopy on noun wires to yield T' and another circuit C' that only differs from C up to processive isotopy, and all C' can be obtained in this way. □

The converse of Proposition 0.2.15 would be that any text circuit that can be formed by the composition of symmetric monoidal categories and of plugging gates into boxes yields a text diagram. This would mean that text circuit composition is acceptable as a generative grammar for text. Establishing this converse requires elaboration of some conventions.

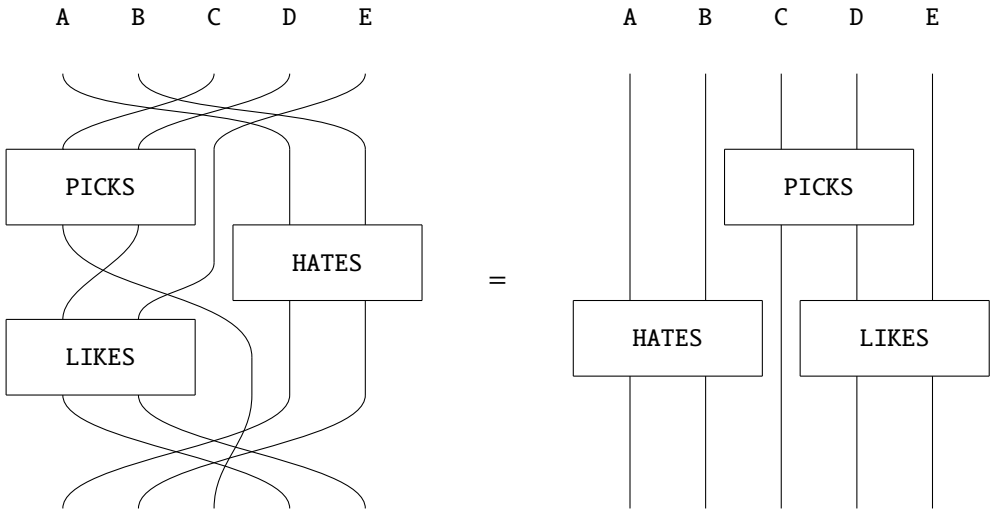


Figure 41: Conjunctions are boxes that take two circuits which might share labels on some wires.

Convention 0.2.17 (Wire twisting)**.**

Wires are labelled by nouns. We consider two circuits the same if their gate-connectivity is the same. In particular, this means that we can eliminate unnecessary twists in wires to obtain diagrammatically simpler representations.

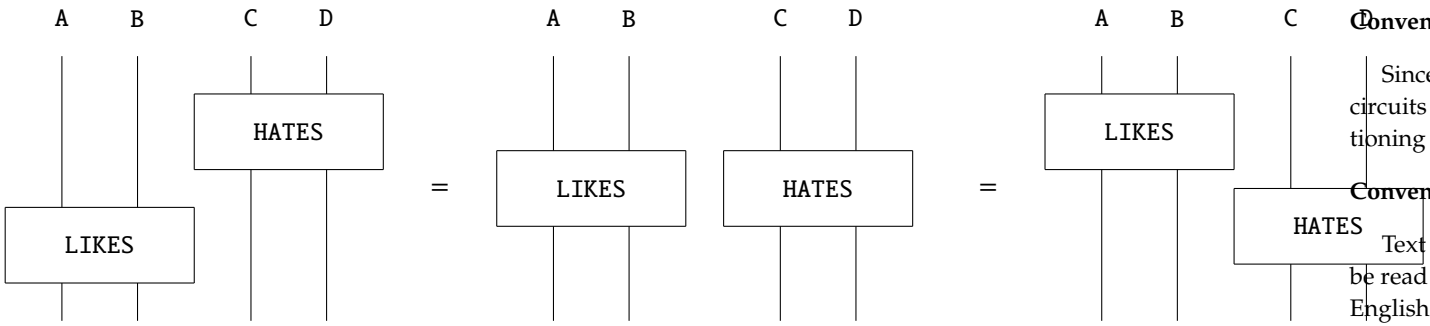


Figure 47:

Convention 0.2.18 (Sliding)**.**

Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel.

Convention 0.2.19 (Reading text circuits)**.**

Text circuits ought to be presented so that they can be read from top to bottom and from left to right, like English text.

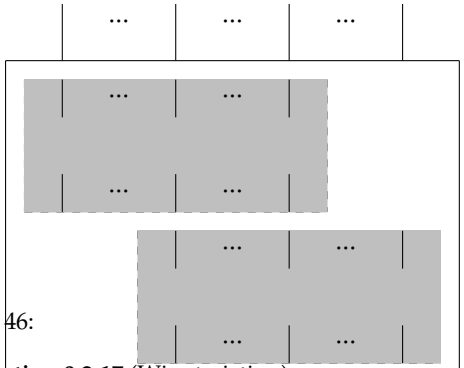


Figure 46:

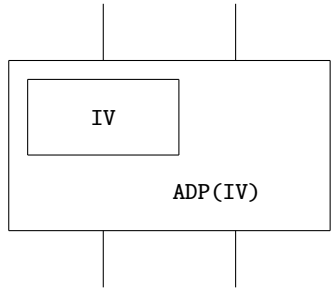


Figure 42: Of course filled up boxes are just gates.

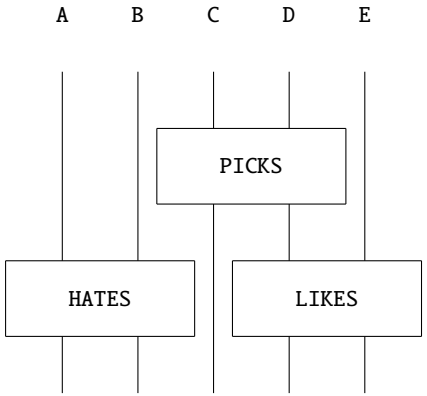
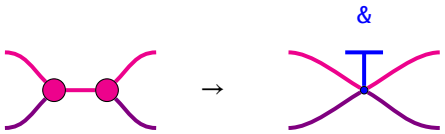


Figure 43: Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits.

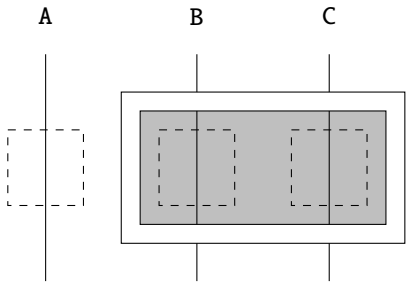
Remark 0.2.16. There are some oddities about our conventions that will make sense later when we consider semantics. For example, Convention 46 an acceptable thing to ask for syntactically but quite odd to think about at the semantic level, where we would like to think that distinct nouns manifest as different states on the same noun-wire-type. A semantic interpretation that makes use of this convention will become clearer in Section configspace . Similarly, Convention 0.2.22 wouldn't be true if we consider the order of text to reflect the chronological ordering of events, in which case there are implicit `...and then ...` conjunctions that distinguish ordered gates from parallel gates conjoined by an implicit `...while ...`; this particular complication is handled at the semantic level in Section statesactionmanner . The distinction in Convention 0.2.20 between typed and "untyped" higher-order processes will be given a suitable semantic interpretation in Section lassos .

Convention 0.2.20 (Arbitary vs. fixed holes)**.** Diagrammatically, adverbs and adpositions are depicted with no gap between the bounding box and their contents, whereas conjunctions and verbs with sentential complement are depicted with a gap; this is a visual indication that the former are type-sensitive, and the latter can take any circuit.

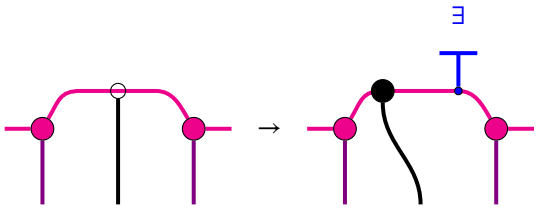
Convention 0.2.21 (Contentless conjunctions)**.** Conventions **??** and 47 require something else to allow them to work at the same time. The left text circuit may be read `C hates D. A likes B.`, and the right as `A likes B. C hates D.` The middle text circuit can be rewritten as either the left or right, which can be done if we're happy to make such choices. A choiceless approach requires something to distinguish the fact that the gates are parallel: a "contentless" conjunction, such as `and`, or `while`. In terms of text diagrams, we want a rewrites that introduces such contentless conjunctions and witnesses their associativity, like this:



Convention 0.2.22 (Lonely wires)**.** There's really only a single kind of text circuit we can draw that doesn't obviously correspond to a text diagram, and that's one where gates are missing.



In process theories, wires are identity processes that do nothing to their inputs. We require a text diagram analogue, and an intransitive "null-verb" in English that seems to work is `is`, in the sense of `exists`. In terms of text diagrams, we want a rewrite that introduces such contentless verbs, like this:



Construction 0.2.23 (Circuit to text)**.** In the presence of additional rewrites from Conventions 0.2.21 and 0.2.22, every text circuit is obtainable from some text diagram, up to Conventions 46 and **??**.

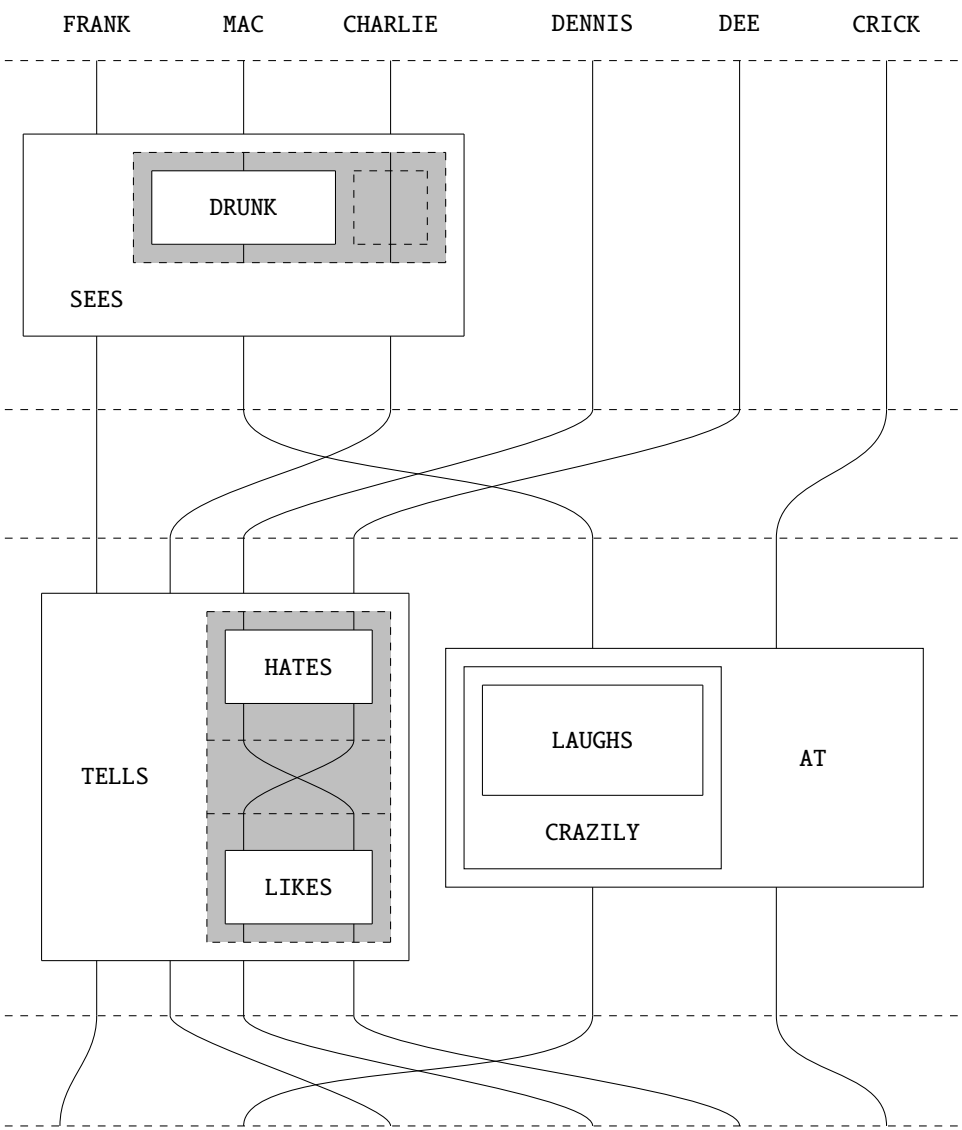


Figure 48: Starting with a circuit, we may use Convention 46 to arrange the circuit into alternating slices of twisting wires and (possibly tensored) circuits, and this arrangement recurses within boxes. Slices with multiple tensored gates will be treated using Convention 0.2.21. By convention 0.2.22, we decorate lonely wires with formal `exists` gates, as in the `Frank sees` box. Observe how verbs with sentential complement are depicted with grey gaps, whereas the adverb and adposition combination of `Mac crazily laughs at Cricket` is gapless, according to Convention 0.2.20.

TODO: insert text circuit theorem

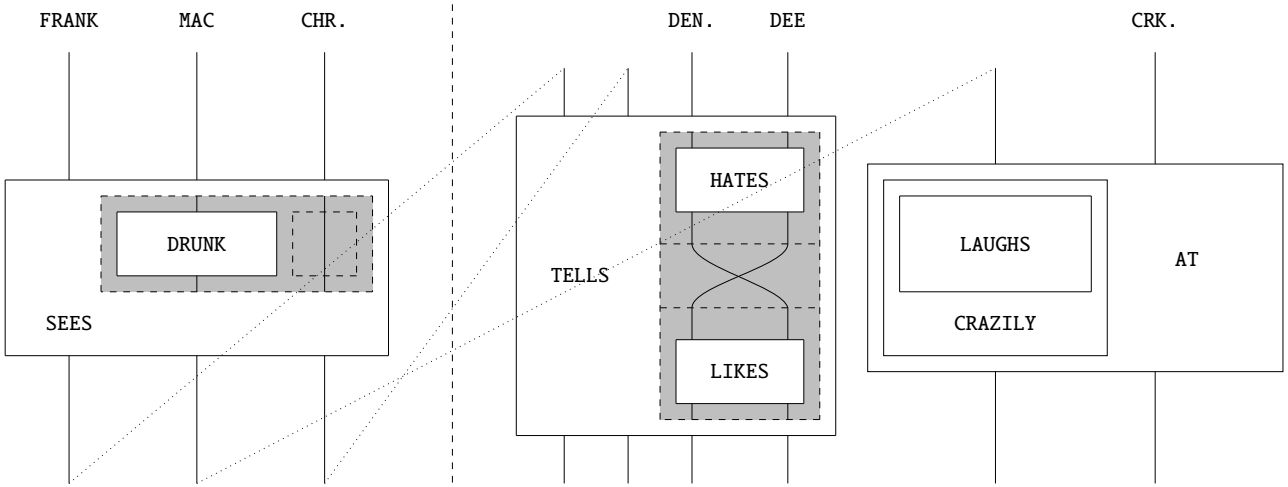


Figure 49: We then linearise the slices, representing top-to-bottom composition as left-to-right. Twist layers are eliminated, replaced instead by dotted connections indicating processive connectivity. The dashed vertical line distinguishes slices. This step of the procedure always behaves well, guaranteed by Proposition 0.2.12. Noun wires that do not participate in earlier slices can be shifted right until the slice they are introduced.

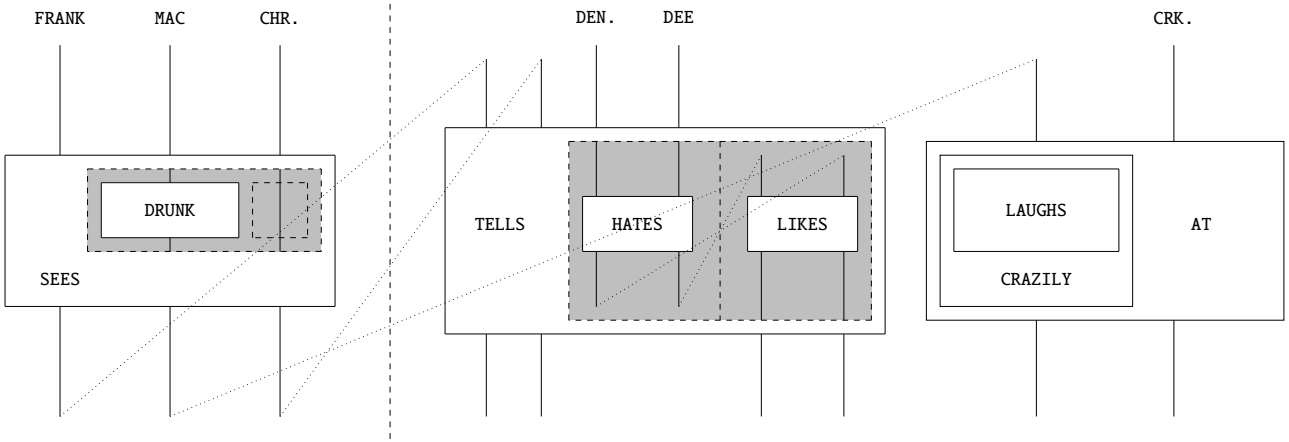


Figure 50: We recurse the linearisation procedure within boxes until there are no more sequentially composed gates. The linearisation procedure evidently terminates for finite text circuits. At this point, we have abstracted away connectivity data, and we are left with individual gates.

0.2.6 Extensions I: relative and reflexive pronouns

Relative pronouns

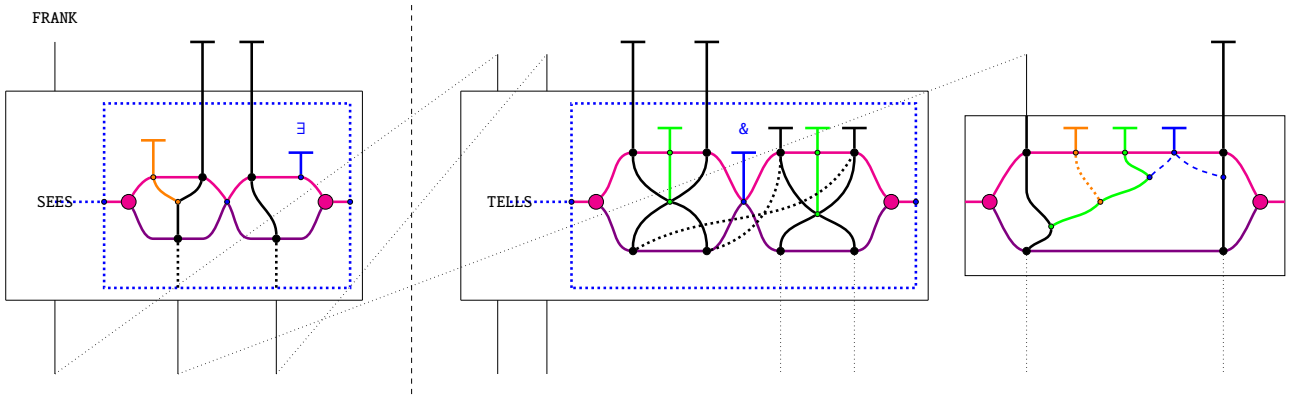


Figure 51: By Proposition 0.2.5, gates are equivalent to sentences up to notation, so we swap notations *in situ*. Conventions 0.2.21 and 0.2.22 handle the edge cases of parallel gates and lonely wires. Observe that the blue-dotted wiring in text diagrams delineates the contents of boxes that accept sentences.

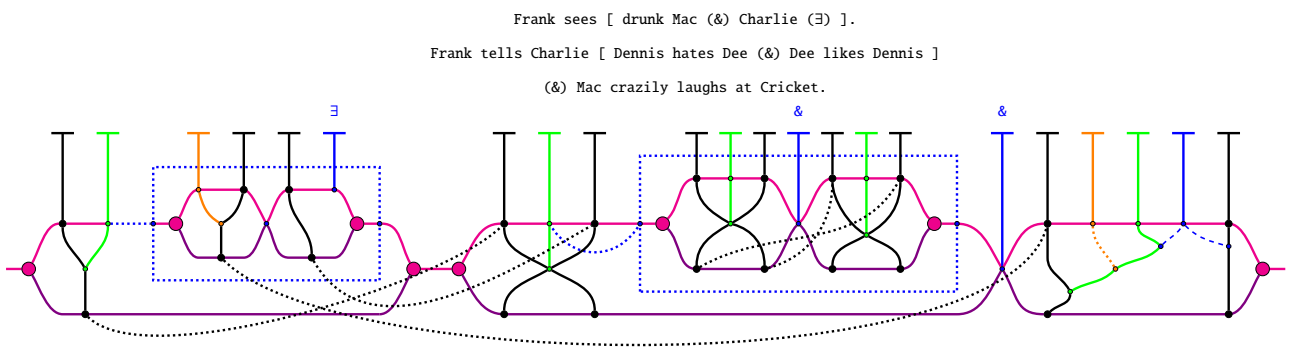
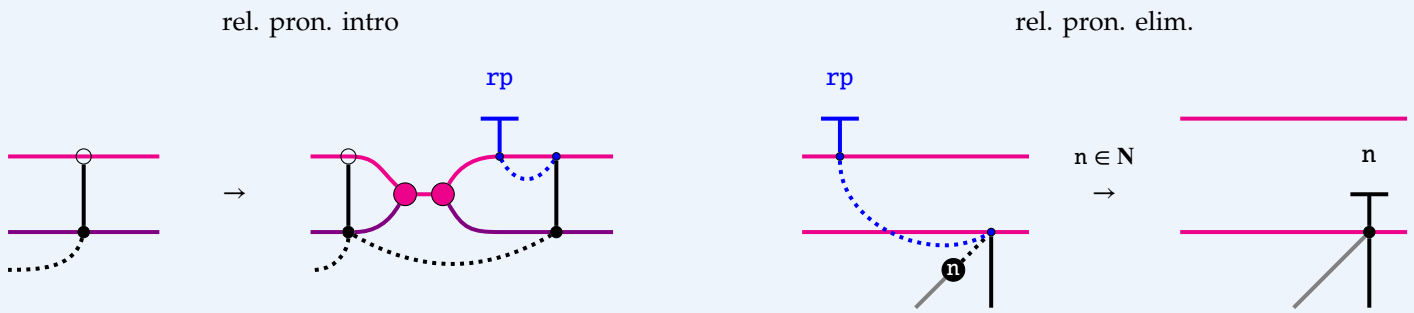


Figure 52: Recursing notation swaps outwards and connecting left-to-right slices as sentence-bubbles connect yields a text circuit, up to the inclusion of rewrites from Conventions 0.2.21 and 0.2.22: applying the reverse of those rewrites and the reverse of text-diagram rewrites yields a valid text-diagram derivation, by Propositions 0.2.5 and 0.2.12. We haven't formally included transitive verbs with sentential complement in our vocabulary, but it should be obvious at this point how they function with our existing machinery.

A relative pronoun glues two sentences across a single noun. For example, what would be a text with two sentences `Alice teaches at school.` `School bores Bob.` can be composed by identifying `school`: `Alice teaches at school that bores Bob.` In this case, `that` is called a subject-relative pronoun, as it stands in for the subject argument in `bores`. In `Alices teaches at school that Bob attends.`, we have an object-relative pronoun, as `that` stands in for the object argument in `attends`. In English, relative pronouns occur immediately after the noun they copy, and are followed immediately by a sentence missing a noun.

Reflexive pronouns are dealt with in a similar fashion as we have with relative pronouns, and various semantic interpretation options have been covered already in Figure 29.

Construction 0.2.24. We can extend our system to accommodate relative pronouns as follows. The introduction of a relative pronoun is considered to start a new sentence with a premade pronominal link. The relative pronoun connects to a new kind of surface noun, which for most intents and purposes behaves just like the nouns we have seen before, except for two caveats: it cannot be labelled (since the relative pronoun is already handling that), and it cannot leave the sentence it starts in by N-shift rules. We can eliminate relative pronouns by forcing the governed noun to be named, which is equivalent to dropping the relative pronoun and recovering distinct sentences.



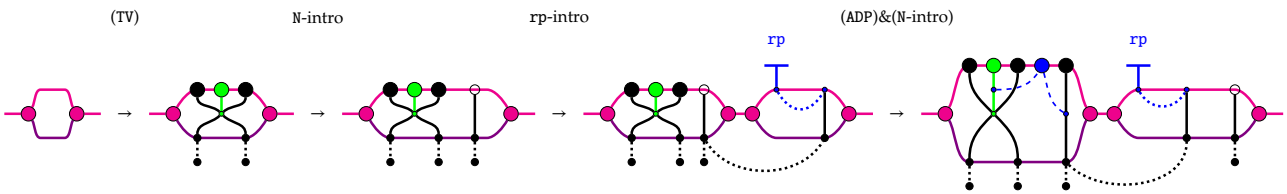


Figure 53:

Example 0.2.25 (Introducing relative pronouns)**.**

Here we demonstrate derivations of `Alice teaches at school that bores Bob` and `Alice teaches at school that Bob attends`. The initial steps in both cases are the same, setting up the `teaches` phrase structure and introducing a new unsaturated noun in the `Bob` phrase to work with the relative pronoun.

0.2.7 Extensions II: grammar equations

One heuristic for extension of text diagrams to larger fragments of text is to devise *grammar equations* that express novel textual elements in terms of known text diagrams. The application of this heuristic is illustrated by example.

0.2.8 Extensions III: Types and Nesting

A heuristic for the extension of text diagrams to novel grammatical categories is the absorption of type-theoretical compositional constraints at the level of sentences into nesting of boxes. A mathematical-historical perspective that justifies this usage of diagrams is in Appendix CITE .

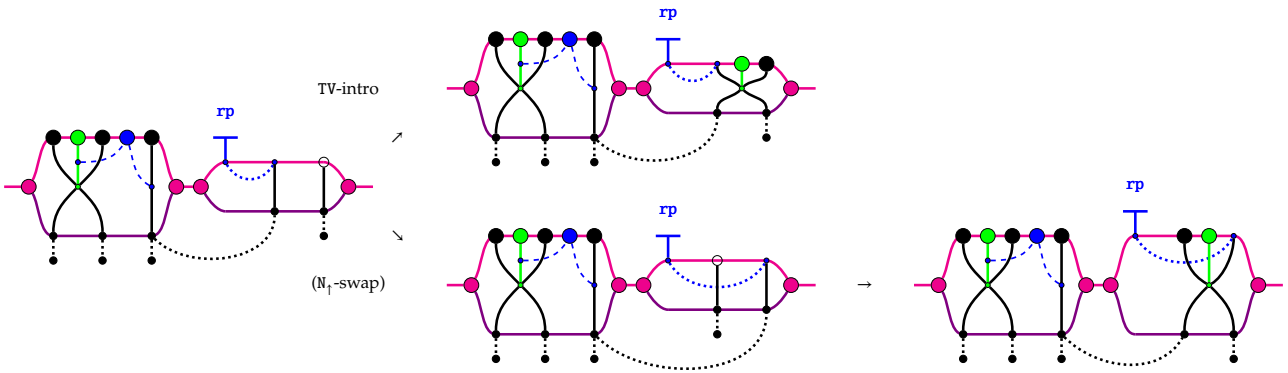


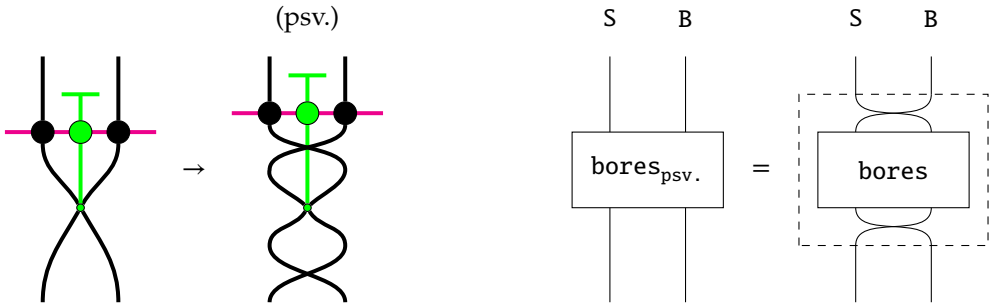
Figure 54: Now have a branching derivation. We may either directly generate a transitive verb treating the relative pronoun as a subject, or we may first perform an $\mathtt{N}_{\uparrow}$-swap first and then generate a transitive verb, treating the relative pronoun as an object. Now the ends of either branch can be labelled to recover our initial examples.

Figure 55:

Example 0.2.26 (Passive voice).

`School bores Bob` = `Bob is bored by school`

Twists in wires can be used to model passive voice constructions, which amount to swapping the argument order of verbs. In the original paper CITE , a more detailed analysis including the flanking words `is bored` by involves introducing a new diagrammatic region, which is modelled by having more than a single 0-cell in the *n*-categorical signature.



Intensifiers

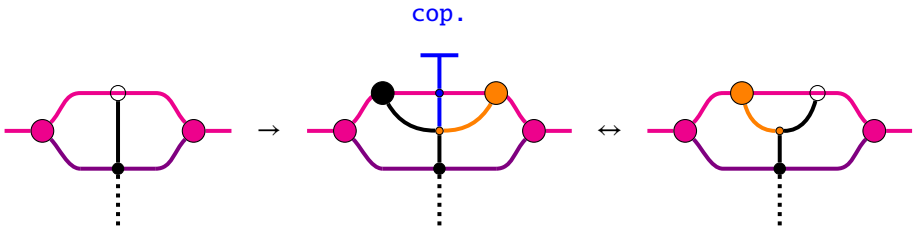
Figure 56:

Example 0.2.27 (Copulas).

`Red car = Car is red`

Modifiers such as adjectives and adverbs when they occur before their respective noun or verb are called *attributive*. When modifiers occur after their respective target, they are called *predicative*. In English, without the aid of `and`, only a single predicative modifier is permissible, e.g. `big red car` and `big car is red` are both acceptable, but textttcar is big red is not. There is no issue in introducing rewrites to handle copular modifier constructions in text diagrams, and in text circuits, there is no distinction between either kind of modifier.

ADJ-intro(pred.)



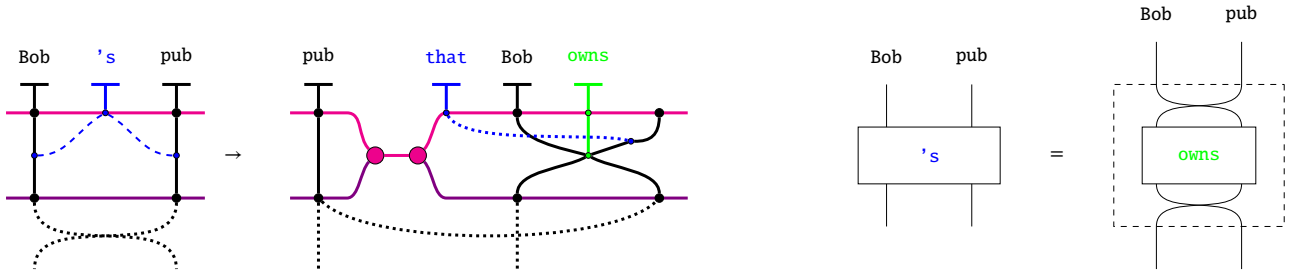
pred/attr swap

Figure 57:

Example 0.2.28 (Possessive pronouns).

`Bob's pub = Pub that Bob owns`

This example, along with other grammar equations, was first introduced in the pregroups and internal wirings context in CITE . Possessive pronouns are placed contiguously in between noun-phrases, for which the diagrammatic technology we developed for placing adpositions can be repurposed. Possessive pronouns may be dealt with by a single rewrite that relies on the presence of a transitive ownership verb in the lexicon, which corresponds to a box-analysis in text circuits.



0.3 *Text circuits: details, demos, developments*

This section covers some practical developments, conventions, references for technical details of text circuits. The most striking demonstration to date is that circuits are defined over a large enough fragment of language to *leverage* several bAbi tasks CITE , which are a family of 20 general-reasoning text tasks – the italicised choice of wording will be elaborated shortly. Each family of tasks consists of tuples of text in simple sentences concluded by a question, along with an answer. It was initially believed that world models were required for the solution of these tasks, but they have been solved using transformer architectures. While there is no improvement in capabilities by solving bAbi using text circuits, the bAbi tasks have been used as a dataset to learn word gates from data, in a conceptually compliant and compositional manner. Surprisingly, despite the low-data, low-compute regime, the tasks for which the current theory has the expressive capacity to cover are solved better by text circuits than by LSTMs; a proof-of-concept that with the aid of appropriate mathematics, not only might fundamental linguistic considerations help rather than hinder NLP, but also that

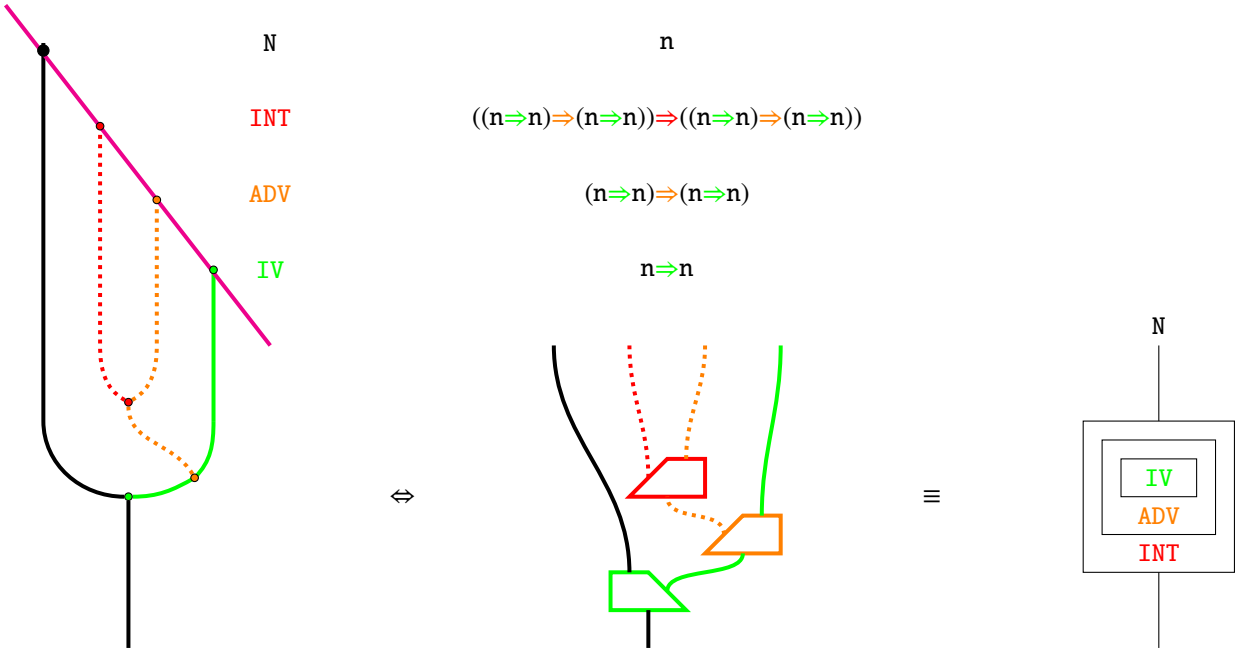


Figure 58:

Example 0.2.29 (Intensifers).

```
Alice very quickly runs
```

The deep nodes of a text diagram may be equivalently viewed as evaluators in a symmetric monoidal closed setting, and the surface nodes as states for the evaluators. By Curry-Howard-Lambek, this view recovers typelogical grammar settings where composition is some variant of modus ponens. So long as the typing rules are operadic or treelike (which is almost always the case for typelogical grammars, as there are rarely gentzen-style sequent rules that generate multiple outputs), we may instead use a notation where parent edges of evaluation branches become nesting boxes.

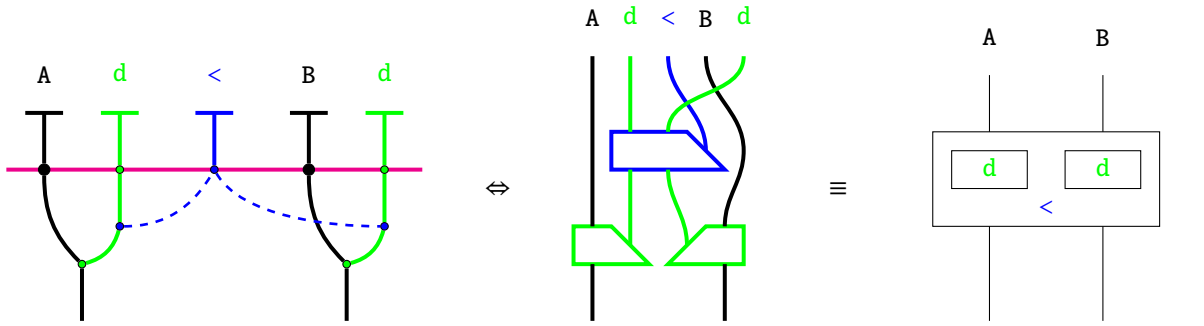


Figure 59:

Example 0.2.30 (Comparatives).

```
Alice drinks less than Bob drinks
```

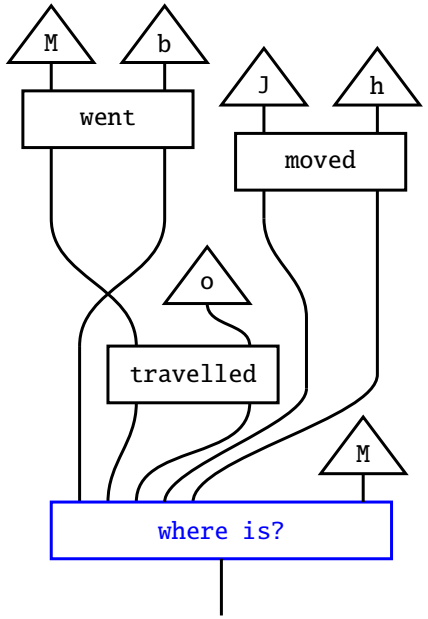
Just as transitive verbs modify two nouns, comparatives are higher-order transitive modifiers that act on the data of verbs or adjectives. A benefit of the symmetric monoidal closed view is that it easily accommodates mixed-order and multi-argument modifiers.

explainability and capability are not mutually exclusive. Experimental details are elaborated in a forthcoming report CITE . While there are expressivity constraints contingent on theoretical development, this price buys a good amount of flexibility within the theoretically established domain: text circuits leave room for both learning-from-data and "hand-coded" logical constraints expressed process-theoretically, and naturally

An example from Task 1, "single supporting fact", is:

```
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
(Query:) Where is Mary?
(Answer:) office.
```

Translating the setup of each task into a circuit of neural nets-to-be-learnt, and queries into appropriately typed measurements-to-be-learnt, each bAbi task becomes a training condition: the depicted composite process ought to be equal to the `office` state.



Solving bAbi in this way also means that each word gate has been learnt in a conceptually-compliant manner, insofar as the grounded meanings of words are reflected in how words interact and modify one another. One may argue that the verb `to go` and synonyms have been learnt-from-data in a way that coheres with human conceptions by appeal to the bAbi dataset.

accommodate previously computed vector embeddings of words.

In practice, the process of obtaining transparently computable text goes through two phases. First, one has to obtain text circuits from text, which is conceptually simple: typelogical parsers for sentences can be modified to produce circuit-components rather than trees, and a separate pronomial resolution module dictates symmetric monoidal compositionality; details are in the same forthcoming report CITE . Second, one implements the text circuits on a computer. On quantum computers, boxes are modelled as quantum combs CITE . On classical computers, boxes are sandwiches of generic vector-manipulation neural nets, and boxes with 'dot dot dot' typing are interpreted as families of processes, which can be factored for instance as a pair of content-carrying gates along with a monoid+comonoid convolution to accommodate multiplicity of wires. The theoretical-to-practical upshot of text circuits when compared to DisCoCat is that the full gamut of compositional techniques, variations, and implementation substrates of symmetric monoidal categories may be used for modelling, compared to the restrictions inherent in hypergraph and strongly compact closed categories.

In terms of underpinning mathematical theory, the 'dot dot dot' notation within boxes that indicates related families of morphisms is graphically formal (**?**), and interpretations of such boxes were earlier formalised in (**???**). The two forms of interacting composition, one symmetric monoidal and the other by nesting is elsewhere called *produoidal*, and the reader is referred to CITE for formal treatment and a coherence theorem. Boxes with holes may be interpreted in several different ways. Firstly, boxes may be considered syntactic sugar for higher-order processes in monoidal closed categories, and boxes are diagrammatically preferable to combs in this regard, since the latter admits a typing pathology where two mutually facing combs interlock. Secondly, boxes need not be decomposable as processes native to the base category, admitting for instance an interpretation as elementwise inversion in linear maps, which specialises in the case of **Rel** (viewed as **Vect** over the boolean ring) to negation-by-complement. In some sense, none of these formalities really matter, on the view that text circuits are algebraic jazz for interpreting text, where facets are open to interpretation and modification. What follows are brief sketches of avenues for extensions of the theory.

0.3.1 Avenues I: syncategorematicity as distributivity

A useful heuristic for the application of text diagrams is to treat individual text circuits as analogous to propositional contents, and certain logical or temporal connectives as structural operations upon circuits – rewrites – that must be applied in order to obtain a purely propositional format. In other words, logical or structural words are to be treated as circuit-manipulation instructions to be executed in order to obtain a circuit, in the same way that 1 + 1 is only an integer expression once addition has been evaluated. It is suggested here that the n-categorical setting is a suitably rich rewriting system to accommodate such rewrites.

Figure 60:

Example 0.3.1 (Syncategorematicity I).

`Alice and Bob drink`

Syncategorematic are roughly words have contextually-dependent semantics. In our terms, since we consider the semantics of text circuits to be underpinned by monoidal functors that reify the circuits in a target category, syncategorematic words such as and may be treated as distributive laws. In this example, and occurs as a conjunction of nouns, and is eliminated by distributive-law rewrites within the deep structure of the text diagram *before translation into circuits*. Note that what is meant by *distributive* here is, in string-diagrammatic terms, precisely the same as that in algebra, for expressions such as $a \times (b + c) \ = \ (a \times b) + (a \times c)$. A new copy-node for verb labels that rewrites for all verbs facilitates distribution, and the deep and nodes come in a tensor-dentensor pair analogous to those for nonstrict string diagrams CITE . Sources of rewrites are outlined in dashed boxes.

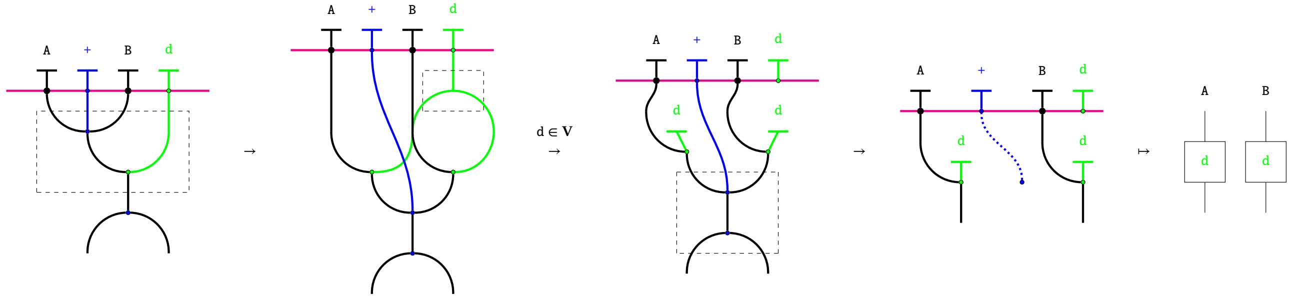
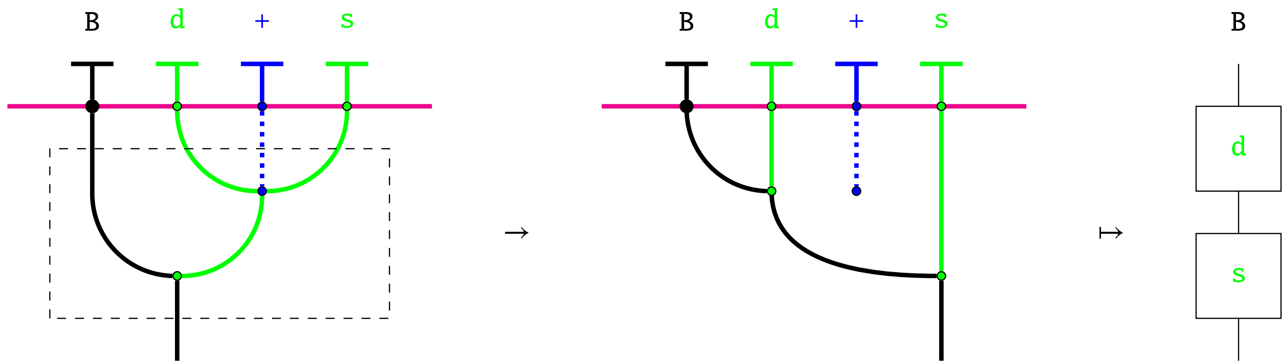


Figure 61:

Example 0.3.2 (Syncategorematicity II).

`Bob drinks and smokes`

In this example, the same word and is a conjunction of verbs. In this case we choose to interpret the conjunction of verbs as sequential composition, so there is no need for a corresponding detensor for the and of verbs.



0.3.2 *Avenues II: determiners and quantifiers in context*

Extending the reach of text circuits to determiners, quantifiers, and conditionals appears to require a contextual diagram or process theory in which to evaluate and enforce constraints upon the purely syntactic content of text circuits. The broad strategy sketched here rests upon three tactics. First, as in the neural approach to bAbi, word-gates are considered to be paired with measurement-processes that return an analog of truth values[1], the latter of which may be generic tests for adjectives as static predicates or verbs as dynamic predicates. These truth-measurements allow conditionals to be expressed as either circuit-rewrites or constraints on truth-measurements, the latter which are in turn interpretable as loss-functions in the process of training gates[2]. Second, we model context as the rest of the text circuit, which is a modifiably finite model. Third, we suppose we have a way to record and relate alternative circuits. These tactics appear sufficient for a first pass. Determiners may be considered to be context-sensitive connectivity. Universal quantifiers[3] may be analysed

[1] If a tree falls in a forest but nobody hears it, does it make a sound? If data is modified but we have no way to check, has an update happened?

[2] For learnability limitations of the gate-test setup, the reader is referred to Appendix REF .

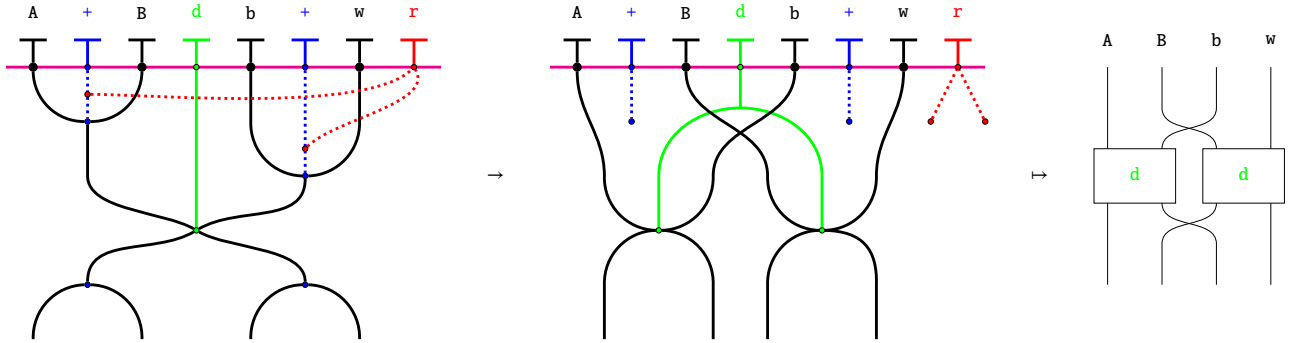
[3] For a short recipe on learning finite first-order models in a cousin of **Vect** with the kronecker product, the reader is referred to Appendix REF .

Figure 62:

Example 0.3.3 (**Coordination**).

`Alice and Bob drink beer and wine respectively`

We stand to win in terms of conceptual economy for modelling; more complex phenomena of text structure such as coordination appear to be resolvable in the same framework of distributivity-law rewrites.



in particular finitary contexts as conditionals and constraints on truth-conditional measurements. Existential quantifiers evaluated in the finitary case yield alternative circuits.

Example 0.3.4 (**Determiners I**).

`Bob drinks the beer` (among drinks)

Here, `drinks` is considered transitive and `the beer` a nesting box for `drinks` that reaches over to contextual wires representing a selection of beverages. In this case (relying on the implicit uniqueness of `the`), a series of `beer?` tests may be computed, and the best match chosen as the resulting argument for `drinks`.

Example 0.3.5 (**Determiners II**).

`Bob drinks a beer` (among drinks)

We take the logical reading of `a` as $\exists x : \texttt{beer?}(x) \wedge \texttt{drinks?}(\texttt{Bob}, x).$*Whentherearetherbeveragesaround*, *wemaycreatealternativ*

Example 0.3.6 (**Determiners III**).

`Bob drinks a beer` (that we didn't know about)

When there are no beers in context, the same statement takes on a dynamic reading: it constitutes the introduction of a beer into discourse. In terms of text circuits, this amounts to introducing a novel beer-state and beer-wire. Determining an appropriate setting to accommodate "arbitrary" vs. "concrete" beers (c.f. Fine's arbitrary objects CITE) requires further research and experimentation, but preliminarily it is known that density matrices are capable of modelling semantic entailment CITE , at the computational cost of adopting the kronecker product.

[Quantifiers I]

`Bob drinks all the beers` (in context)

In a finitary context, drinking all the beers amounts to applying the distributivity of `and`.

[Quantifiers II]

`Bob drinks all beers` (generic)

Without the determiner `the`, this becomes a generic statement, which logically amounts to (analysing the usual conditional as a disjunction) $\forall x : \neg\texttt{beer?}(x) \vee \texttt{drinks?}(\texttt{Bob}, x)$. *We can treat generic universal quantifiers of this kind as rewrite rules : anytime a beverage wire satisfies a beer test, we may freely add on a gate witnessing that Bob drinks that beverage.*

0.3.3 Avenues III: grammar as geometry, geometry as computation

`The example of noun phrases illustrates the higher principle that the particulars of interpretational choices into string diagrams are inessential: any consistent scheme will do, and the string diagrams will give voice to the computational structure of dynamic semantics.`

Example 0.3.7 (**Noun phrases I**)**.**

`The king of France is bald` (new discourse)

Example 0.3.8 (**Noun phrases II**)**.**

`The king of France is bald` (discourse modifier)

.1 A modern mathematician's companion to Montague's "Universal Grammar"

.1.1 What did Montague consider grammar to be?

Summary: Montague considered grammars to be coloured operads; Montague's "algebras" are (multi-sorted) clones, which are in bijection with (multi-sorted) Lawvere Theories, which are equivalently coloured operads.

4
5

Montague semantics/grammar as Montague envisioned it is largely contained in two papers - *Universal Grammar* [4], and *The Proper Treatment of Quantifiers in English* [5] - both written shortly before his murder in 1971. The methods employed were not *mathematically* novel - the lambda calculus had been around since DATE , and Tarski and Carnap had been developing intensional higher-order logics since DATE - but for linguists who, by-and-large, only knew first order predicate logic, these methods were a tour-de-force that solved longstanding problems in formal semantics.

There is a natural division of Montague's approach into two structural components. First, the notion of a structure-preserving map from syntax to semantics. Second, the use of a powerful and expressive logic for semantics. We acknowledge the importance of the latter for formal semantic engineering, but here we will focus on just the former. According to Partee CITE , a formal semanticist, advocate, and torch-bearer for Montague, the chief interest of Montague's approach (as far as his contemporary *linguists* were concerned) lay in the following ideas:

1. Take truth conditions to be the essential data of semantics.

2. (a) Use lambdas to emulate the structure of syntax...

   (b) ...in a typed system of intensional predicate logic, such that composition is function application.

I have split the second point to highlight the role of lambdas. This element was the crux of the Montagovian revolution: according to Janssen in a personal communication with Partee from 1994, lambdas were "...the feature that made compositionality possible at all."

In Section 1 of *Universal Grammar*, Montague's first paragraph establishes common notions of relation and function - the latter he calls *operation*, to distinguish the n-ary case from the unary case which he calls *function*. This is all done with ordinals indexing lists

of elements of an arbitrary but fixed set A, which leads later on to nested indices and redundancy by repeated mention of A. We will try to avoid these issues going forward by eliding some data where there is no confusion, following common modern practice.

Next, Montague introduces his notion of *algebra* and *homomorphism*. He separates the data of the carrier set and the generators from the *polynomial operations* that generate the term algebra.

Section 2 seeks to define a broad conception of 'syntax', which he terms a *disambiguated language*. This is a free clone with carrier set A, generating operations F_γ indexed by $\gamma \in \Gamma$, along with extra decorating information:

1. ($\delta \in$)Δ is an (indexing) set of syntactic categories (e.g. NP, V, etc.). Montague calls this the *set of category indices*. $X_\delta \subseteq A$ form the *basic expressions* of type δ in the language.

2. a set S assigns types among $\delta \in \Delta$ to the inputs and output of - not necessarily all - F_γ.

3. a special $\delta_0 \in \Delta$ is taken to be the type of declarative sentences.

This definition is already considerably progressive. Because there is no condition of disjointness upon the X_δ - a view that permits consideration of the same word playing different syntactic roles - (1) permits the same basic expression $x \in A$ to participate in multiple types $X_\delta \subseteq A$ ($\star$). (2) misses being a normal typing system on several counts. There is no condition requiring all F_γ to be typed by S, and no condition restricting each F_γ to appear at most once: this raises the possibilities that (†) some operations F go untyped, or that (‡) some are typed multiply.

Taking a disambiguated language $\mathfrak{U}$ on a carrier set A, Montague defines a *language* to be a pair $L :=< \mathfrak{U}, R >$, where R is a relation from a subset of the carrier A to a set PE_L, the set of *proper expressions* of the language L. An admirable purpose of R appears to be to permit the modelling of *syntactic ambiguity*, where multiple elements of the term algebra $\mathfrak{U}$ (corresponding to syntactic derivations) are related to the same "proper language expression".

However, we see aspects where Montague was born ahead of his time mathematically: it appears that his intent was to impose a system of types to constrain composition of operations, but the tools were not available for him. Montague addresses (†) obliquely, by defining ME_L to be the image in PE_L of R of just those expressions among A that are typed. Nothing appears to guard against (‡), which causes problems as Montague expresses structural constraints (in the modern view) in terms of constraints on the codomain of an interpreting functor (cf.

Definition .1.1 (Generating data of an Algebra)**.** Let A be the carrier set, and F_γ be a set of functions $A^k \to A$ for some $k \in \mathbb{N}$, indexed by $\gamma \in \Gamma$. Denoted $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$

The following three data are taken to be common among Montague's algebras:

Definition .1.2 (Identities)**.** A family of operations populated, for all $n, m \in \mathbb{N}$, $n \leq m$, by an m-ary operation $I_{n,m}$, defined on all m-tuples as

$$I_{n,m}(a) = a_n$$

where a_n is the n^{th} entry of the m-tuple a.

Definition .1.3 (Constants)**.** For all elements of the carrier $x \in A$, and all $m \in \mathbb{N}$, a constant operation $C_{x,m}$ defined on all m-tuples a as:

$$C_{x,m}(a) = x$$

Definition .1.4 (Composition)**.** Given an n-ary operation G, and n instances of m-ary operations $H_{1 \leq i \leq n}$, define the composite $G(H_i)_{1 \leq i \leq n}$ to act on m-tuples a by:

$$G(H_i)_{1 \leq i \leq n}(a) = G(H_i(a))_{1 \leq i \leq n}$$

N.B. the m-tuple a is copied n times by the composition. Writing out the right hand side more explicitly:

$$G\Big(\big(H_1(a)\,,\ H_2(a)\,,\ \ldots\ ,\ H_n(a) \big) \Big)$$

Definition .1.5 (Polynomial Operations)**.** The polynomial operations over an algebra $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ are defined to be smallest class K containing all $F_{\gamma \in \Gamma}$, identities, constants, closed under composition.

Definition .1.6 (Homomorphism of Algebras)**.** h is a homomorphism from $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ *into* $\langle B, G_\gamma \rangle_{\gamma \in \Delta}$ iff

1. $\Gamma = \Delta$ and $\forall \gamma$:
2.

Montague's notion of *generating* syntactic categories). One consquence, in conjunction with ($\star$), is that every multiply typed operation F induces a boolean algebra where the typings are the generators and the operations are elementwise in the inputs and output. Worse problems occur, as Montague's clone definition include all projectors, and when defined separately from the typing structure, these projectors may be typed in a way that permits operations that arbitrarily change types, which appears to defeat the purpose. We doubt these artefacts are intentional, so we will interpret Montague assuming his intent was a type-system as we would recognise one today.

.1.2 *On the historical inevitability of text diagrams*

Definition .1.2 is equivalent to asking for all projections. Definitions .1.2 and .1.4 together characterise Montagovian algebras as (concrete) clones CITE , which then generalised to (abstract) clones CITE , which were then discovered to be in bijection with Lawvere theories CITE . By an evident extension of [Prop 3.51] to the typed case, a *disambiguated language* is a multi-sorted Lawvere theory without relations, where the sorts are generated from products of a pointed set $(\Delta, \delta_0 : 1 \to \Delta)$.

Lawvere theories are themselves a special case of operadic composition CITE . Operadic composition naturally as that of coloured trees CITE , which is equivalently depicted as nesting expressions according to the tree-structure CITE . Interpreting colours as types, operadic composition subsumes whatever one might wish to do with a typed gentzen-style sequent system where rules are multi-input single-output. While typelogical grammars stop there, PROPs generalise operadic composition to multi-input multi-output CITE , and as combinatorial specifications for string diagrams, weak n-categories generalise PROPs, and we have shown weak n-categories to subsume a distinct evolutionary line of formal syntax from CFGs to TAGs.

In summary, text circuits share a common conceptual and mathematical ancestry with many approaches to formal syntax, hence one ought to expect deep compatibility.