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.

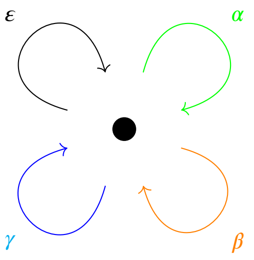


Figure 1: The category in question can be visualised as a commutative diagram.

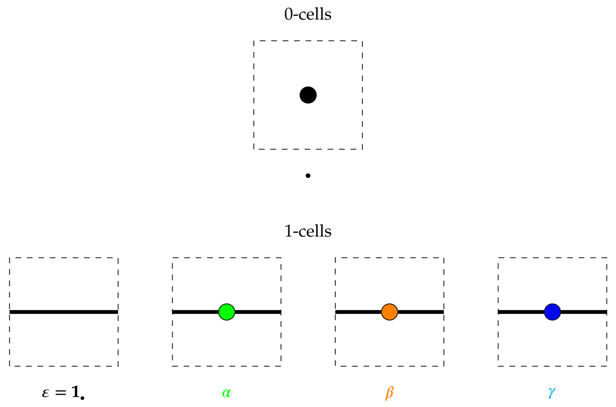


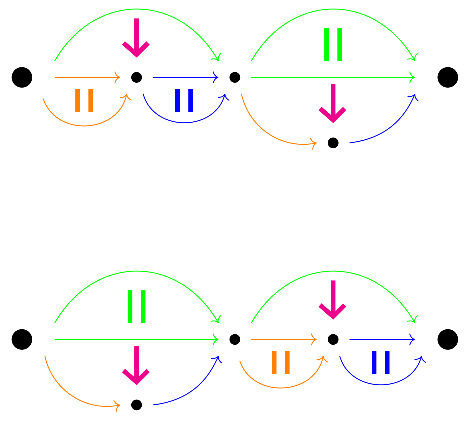
Figure 2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from 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 $\epsilon \cdot x = x = \epsilon \cdot x$.

$$\begin{tikzcd} \bullet \arrow[r, "\alpha"] & \bullet \arrow[r, "\gamma"] & \bullet \arrow[r, "\gamma"] & \bullet \arrow[r, "\beta"] & \bullet \end{tikzcd}$$

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.

We consider rewrites to be equivalent, but not equal. 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.

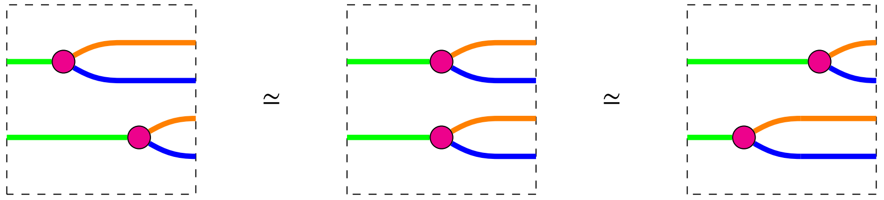
$$\begin{tikzcd} & \beta\gamma\alpha \arrow[dr, bend left, "\mathbf{1}_{\beta\gamma} \cdot R"] & \\ \alpha\alpha \arrow[ur, bend left, "R \cdot \mathbf{1}_\alpha"] \arrow[dr, bend right, "\mathbf{1}_\alpha \cdot R"'] & & \beta\gamma\beta\gamma \\ & \alpha\beta\gamma \arrow[ur, bend right, "R \cdot \mathbf{1}_{\beta\gamma}"'] & \end{tikzcd}$$



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 identities. 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.

$$\begin{tikzcd}
 & \beta\gamma\alpha \arrow[dr] & \\
\alpha\alpha \arrow[ur] \arrow[rr] \arrow[dr] & & \beta\gamma\beta\gamma \\
 & \alpha\beta\gamma \arrow[ur] &
\end{tikzcd}$$

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.

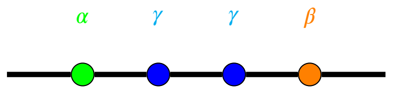


Figure 4: The string-diagrammatic view, where ⋆ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

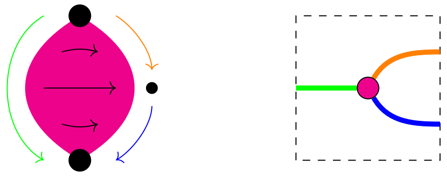


Figure 5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual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

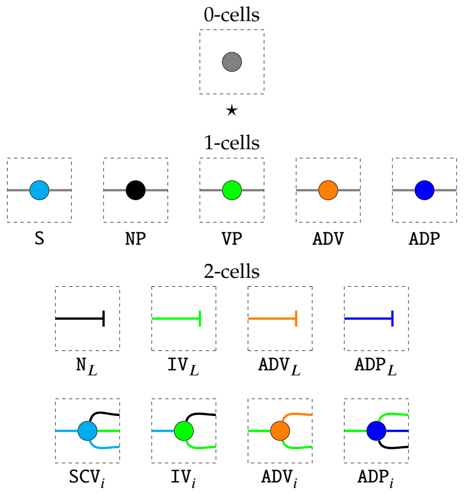


Figure 6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for `Alice sees Bob quickly run to school`.

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 $\downarrow : \mathcal{N} \rightarrow \mathcal{N}^{\downarrow}$ 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 $* : \mathcal{N} \rightarrow \mathcal{N}^{*}$ – 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.

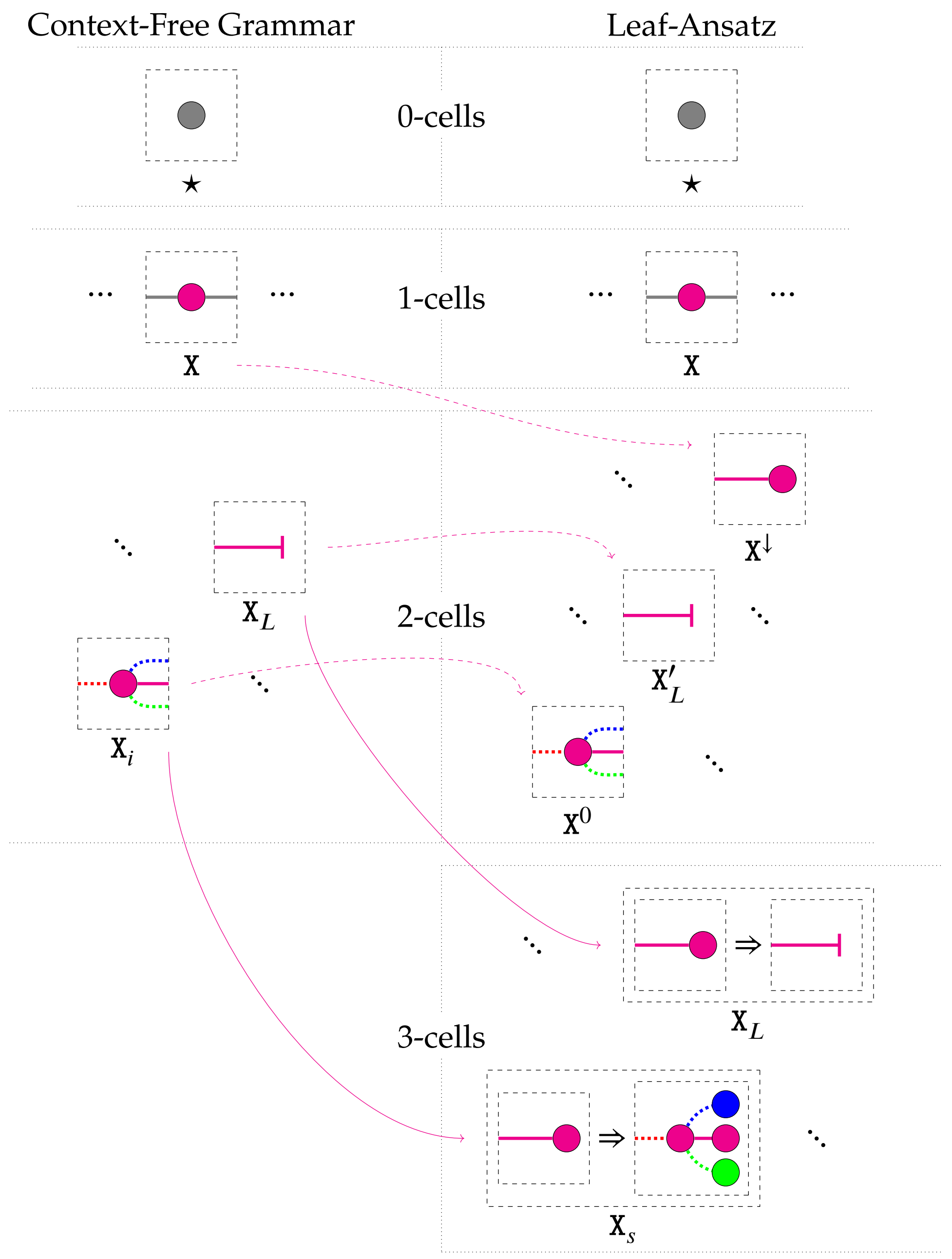


Figure 7:

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 $\texttt{X}$ of $\mathfrak{G}$, we introduce a *leaf-ansatz* 2-cell $\texttt{X}^{\downarrow}$. For each leaf 2-cell $\texttt{X}_L$ in $\mathfrak{G}$, we introduce a renamed copy $\texttt{X}'_L$ 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 $\texttt{X}$ of $\mathfrak{G}$ become substitution marked symbols $\texttt{X}^{\downarrow}$ in $\mathfrak{G}'$. The trees $\texttt{X}_i$ in $\mathfrak{G}$ become initial trees $\texttt{X}^0$ in $\mathfrak{G}'$. The 3-cells $\texttt{X}_s$ of $\mathfrak{G}'$ are precisely substitution operations corresponding to appending the 2-cells $\texttt{X}_i$ 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.

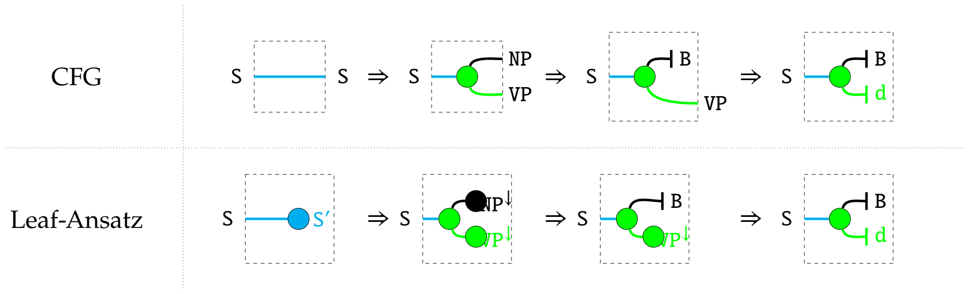


Figure 8: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction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

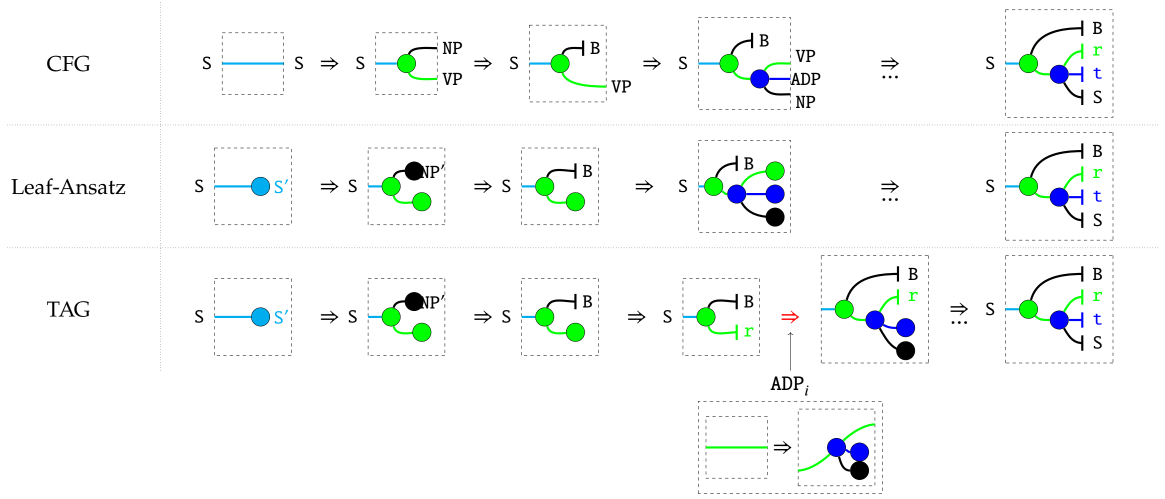
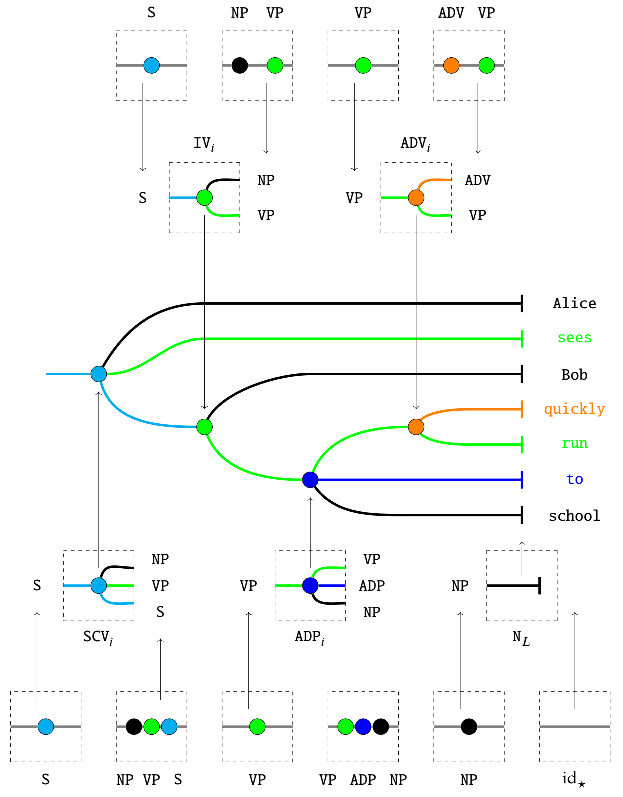
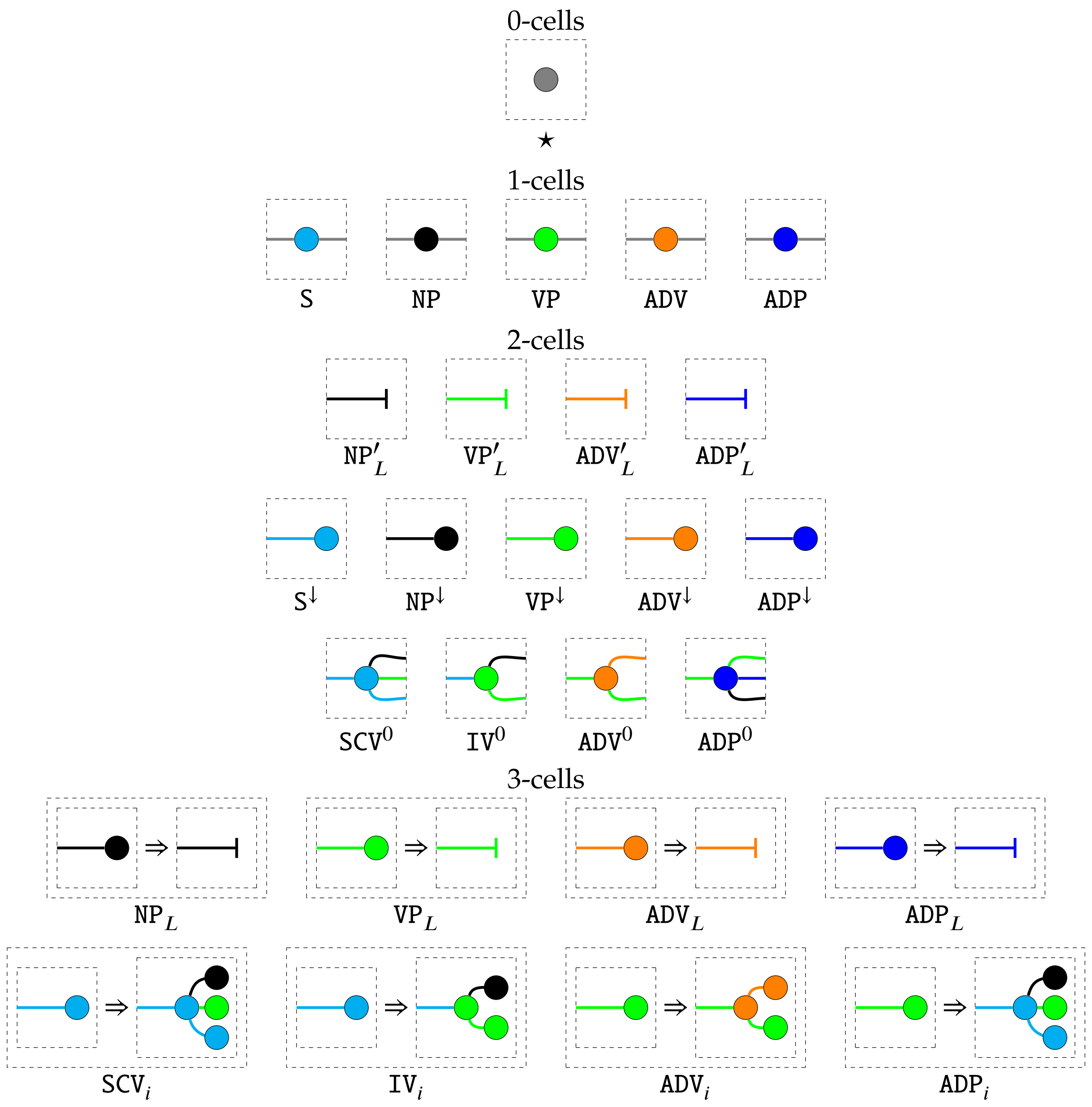


Figure 9: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted 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.





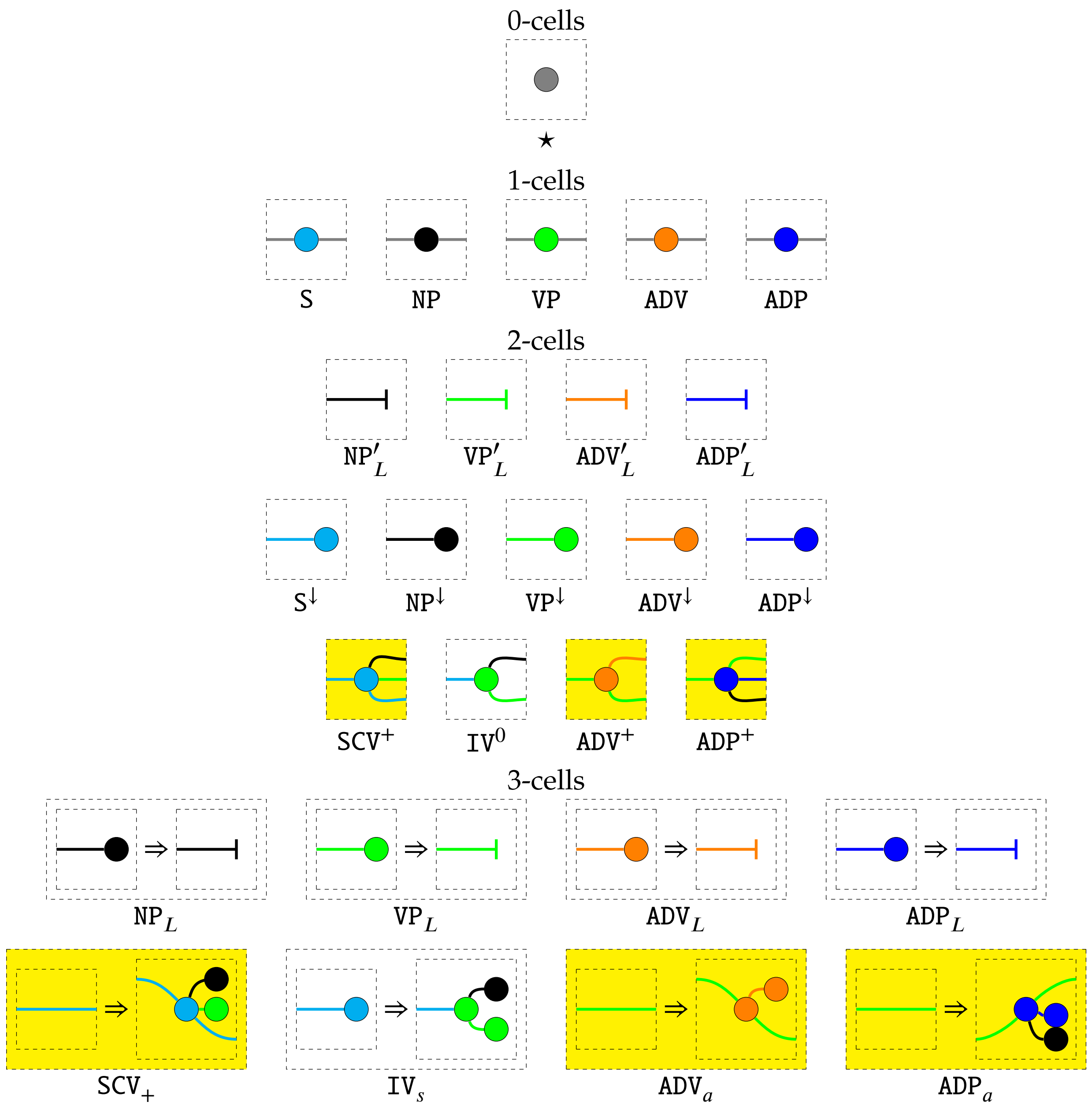


Figure 11: TAG signature of `Alice sees Bob quickly run to school`. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction yields 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*

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.

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.

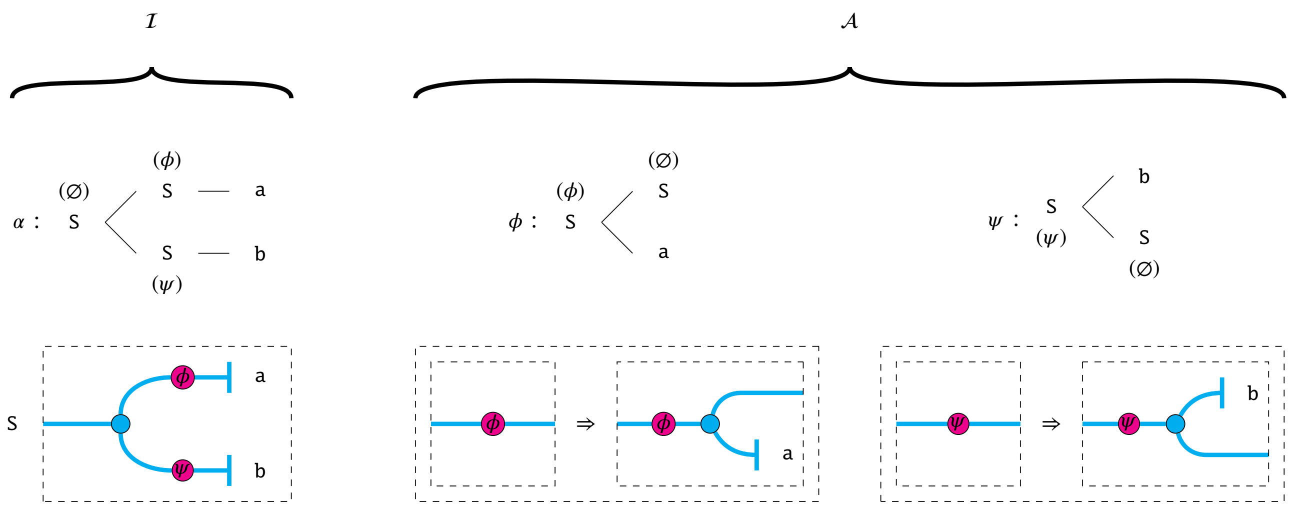


Figure 12: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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.

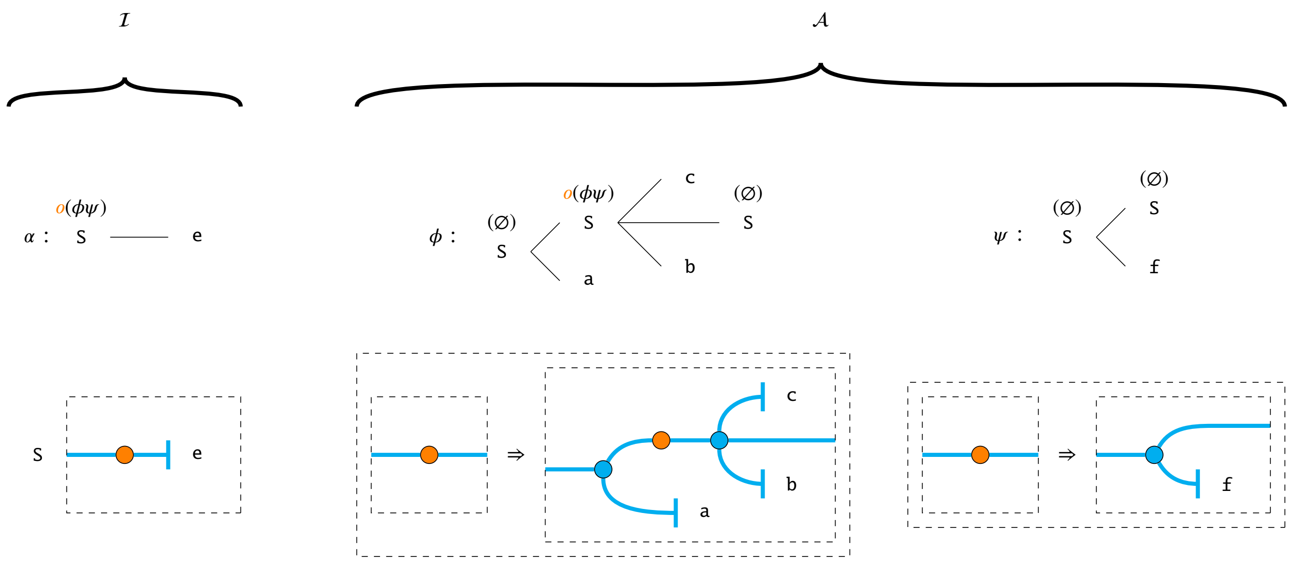
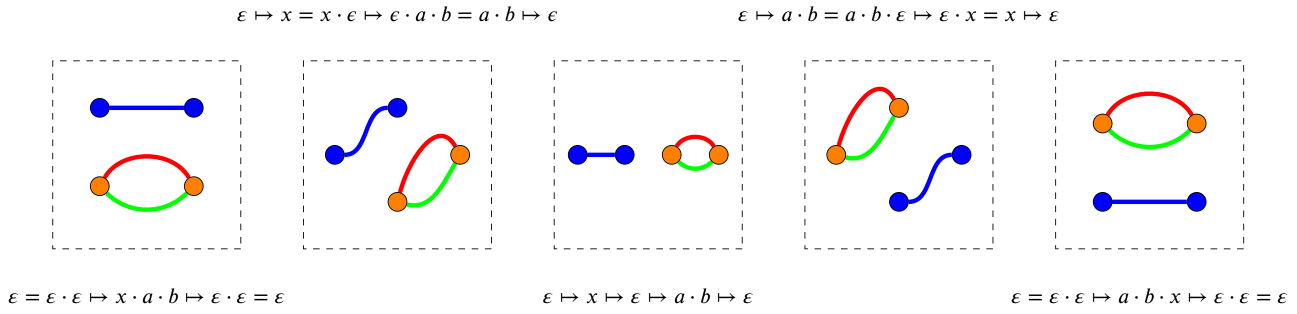


Figure 13: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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.

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



$\varepsilon \mapsto x = x \cdot \epsilon \mapsto \epsilon \cdot a \cdot b = a \cdot b \mapsto \epsilon$
$\varepsilon \mapsto a \cdot b = a \cdot b \cdot \varepsilon \mapsto \varepsilon \cdot x = x \mapsto \varepsilon$
$\varepsilon = \varepsilon \cdot \varepsilon \mapsto x \cdot a \cdot b \mapsto \varepsilon \cdot \varepsilon = \varepsilon$
$\varepsilon \mapsto x \mapsto \varepsilon \mapsto a \cdot b \mapsto \varepsilon$
$\varepsilon = \varepsilon \cdot \varepsilon \mapsto a \cdot b \cdot x \mapsto \varepsilon \cdot \varepsilon = \varepsilon$

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.

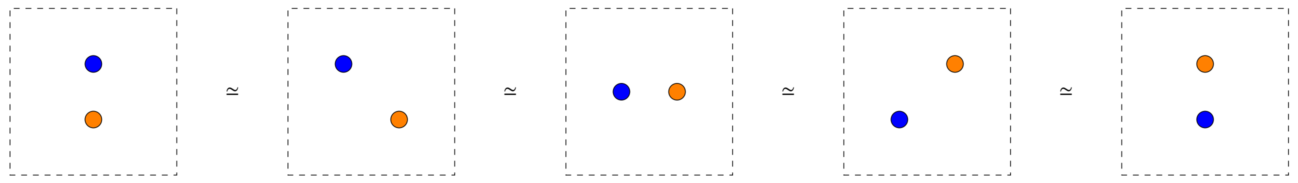


Figure 15: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument CITE is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting 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.

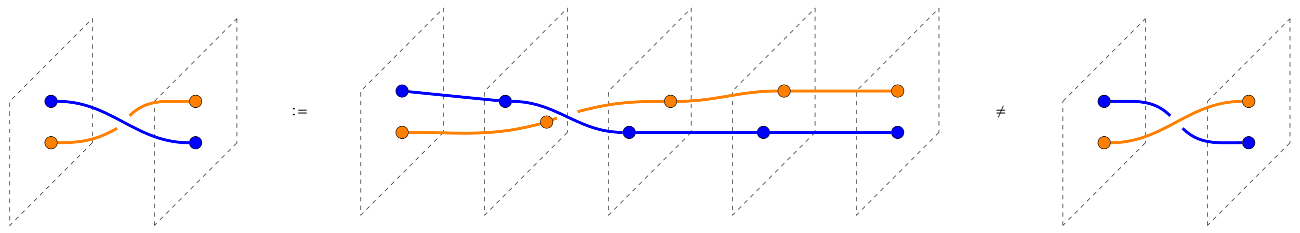


Figure 16: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies CITE , which are continuous bijective transformations that don't let wires double back on themselves, we can identify 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).

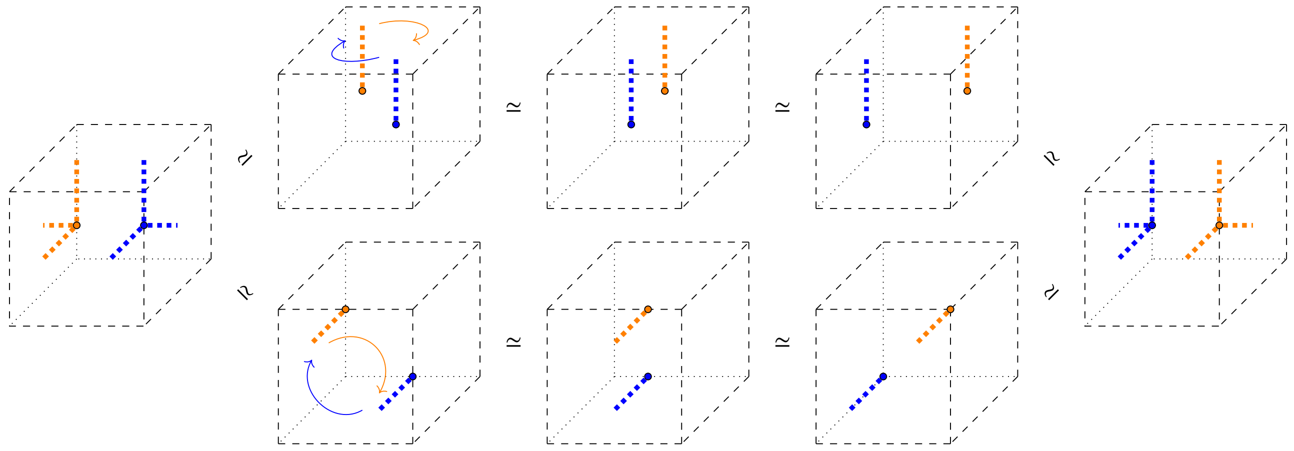


Figure 17: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise 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:

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 ambient 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 $\mathbf{1}_{\mathbf{1}_\star}$, in essence turning dots on a line into dots in a volume; this procedure is called *suspension* CITE .

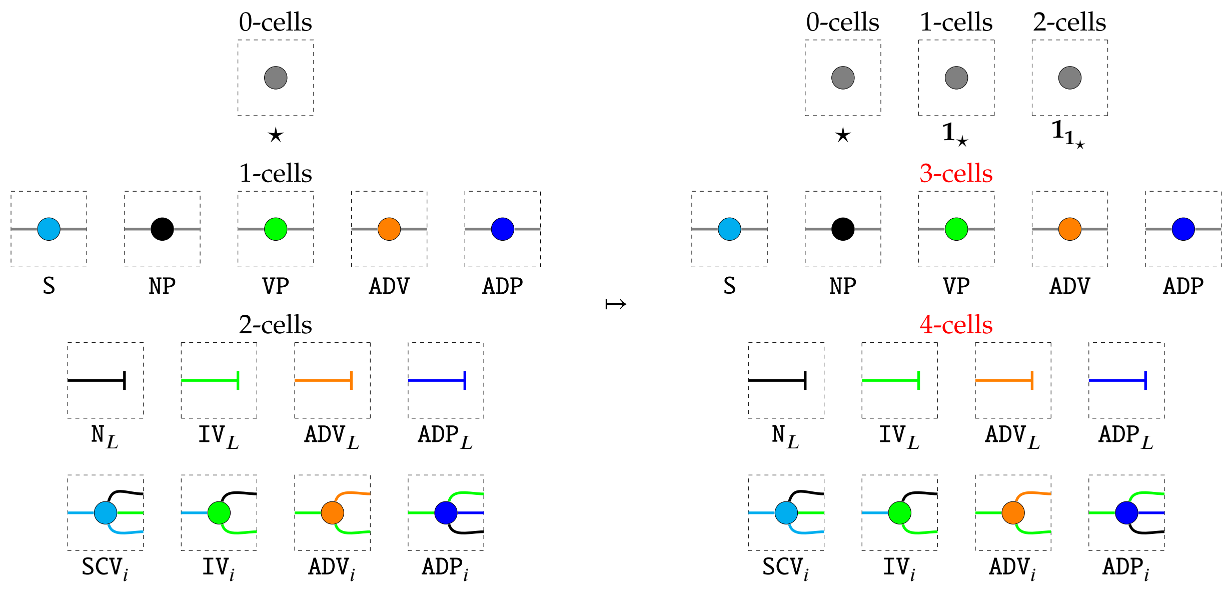


Figure 18: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with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

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.

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.

0.1.5 TAGs with links

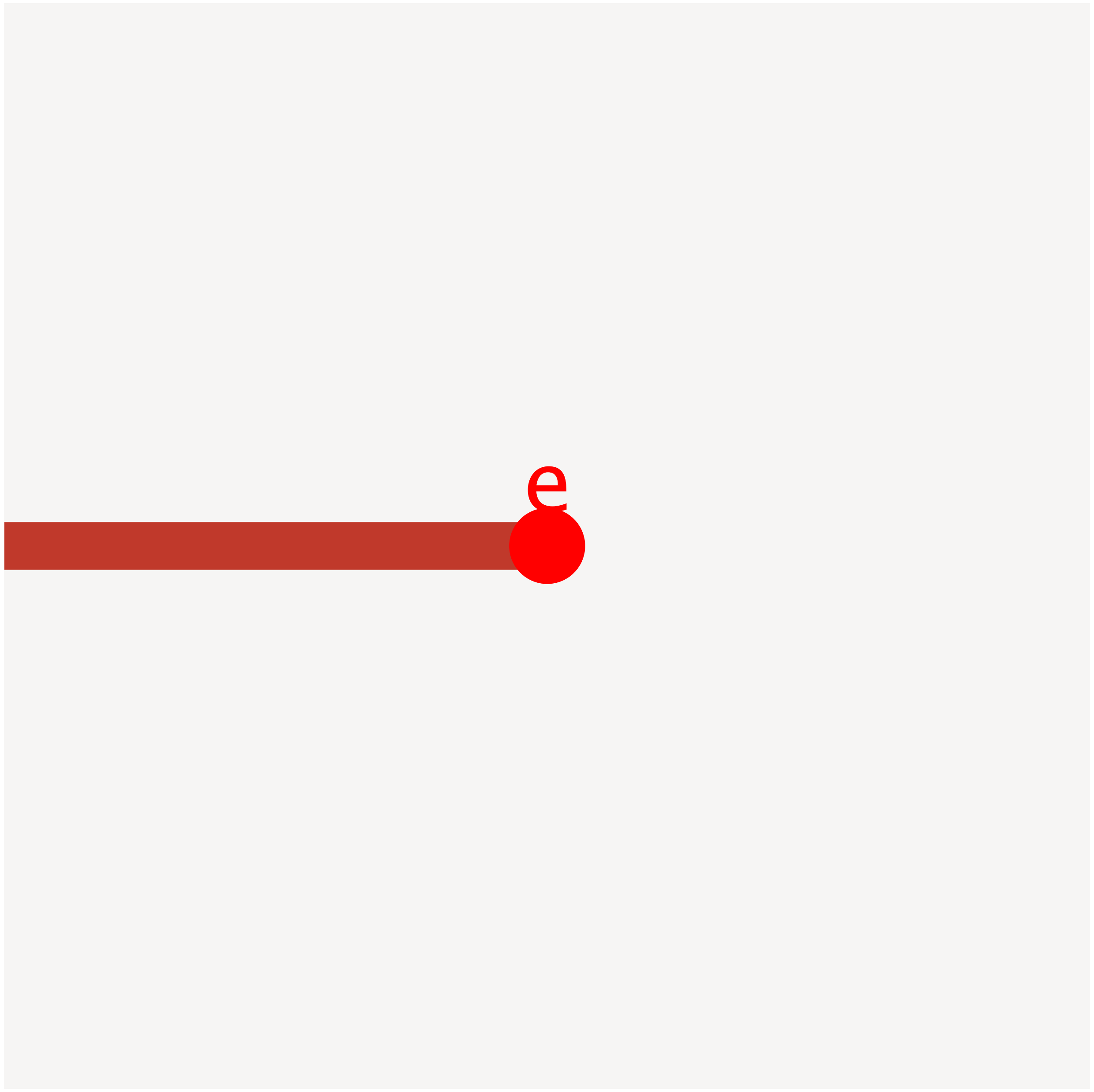
Example 0.1.8. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n-categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out. The trees given in Examples 2.4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

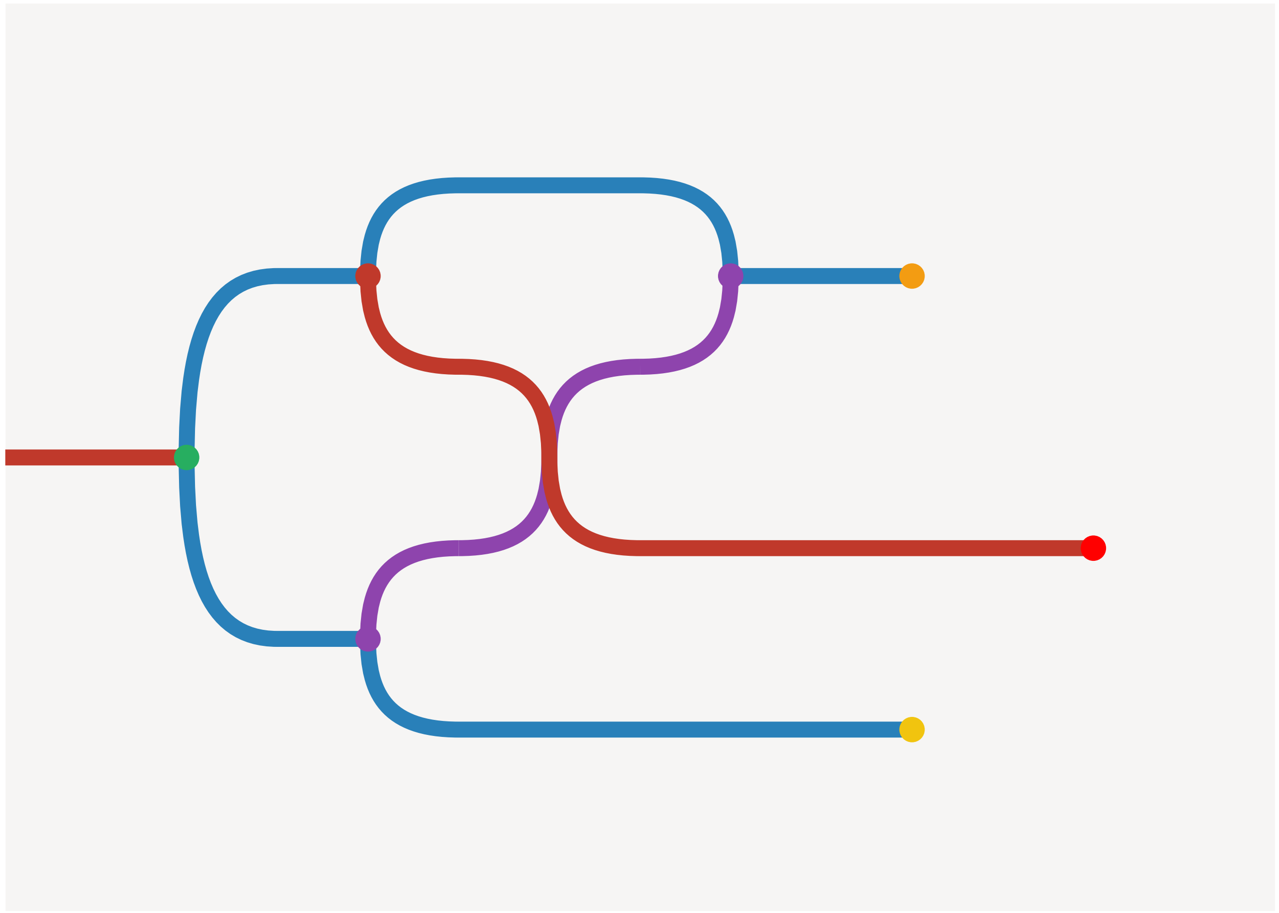
$$placeholder$$

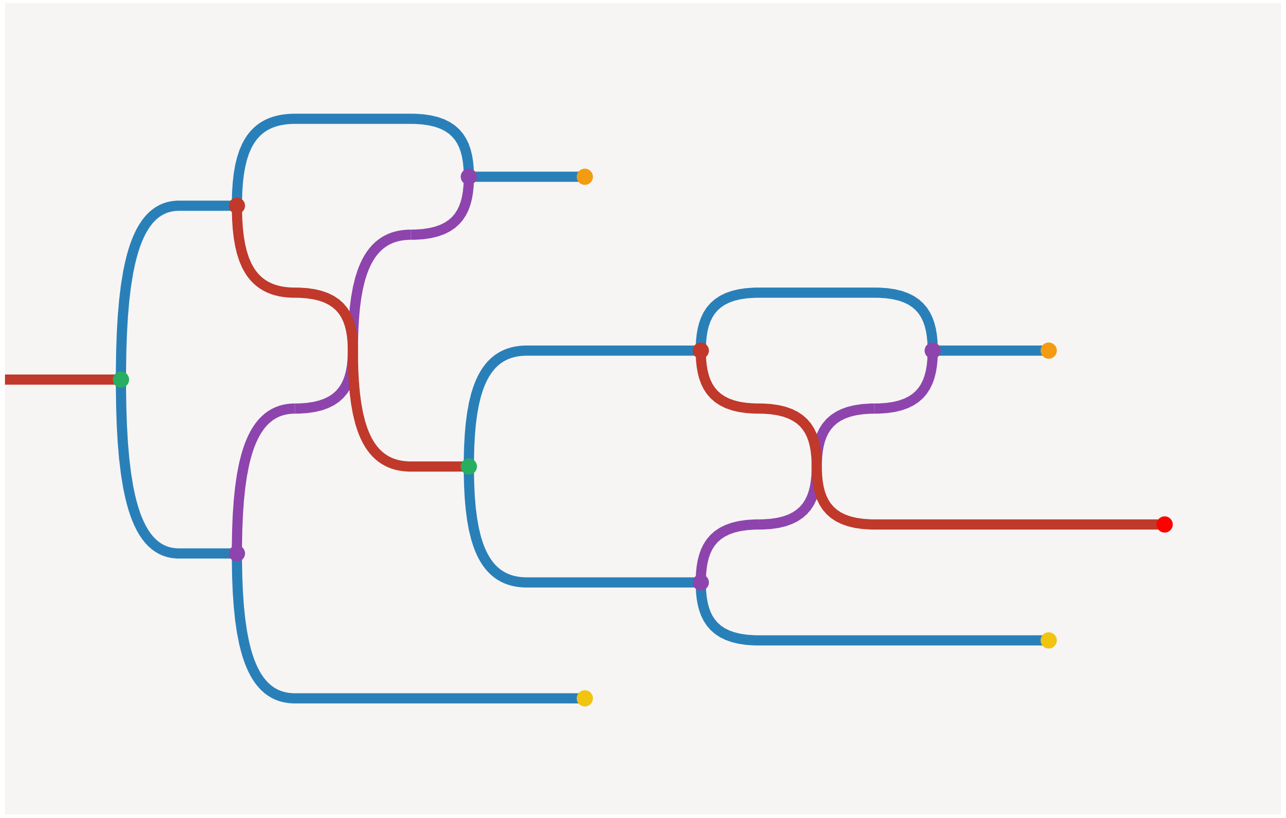
Which we interpret as expressions comprised of the following signature:

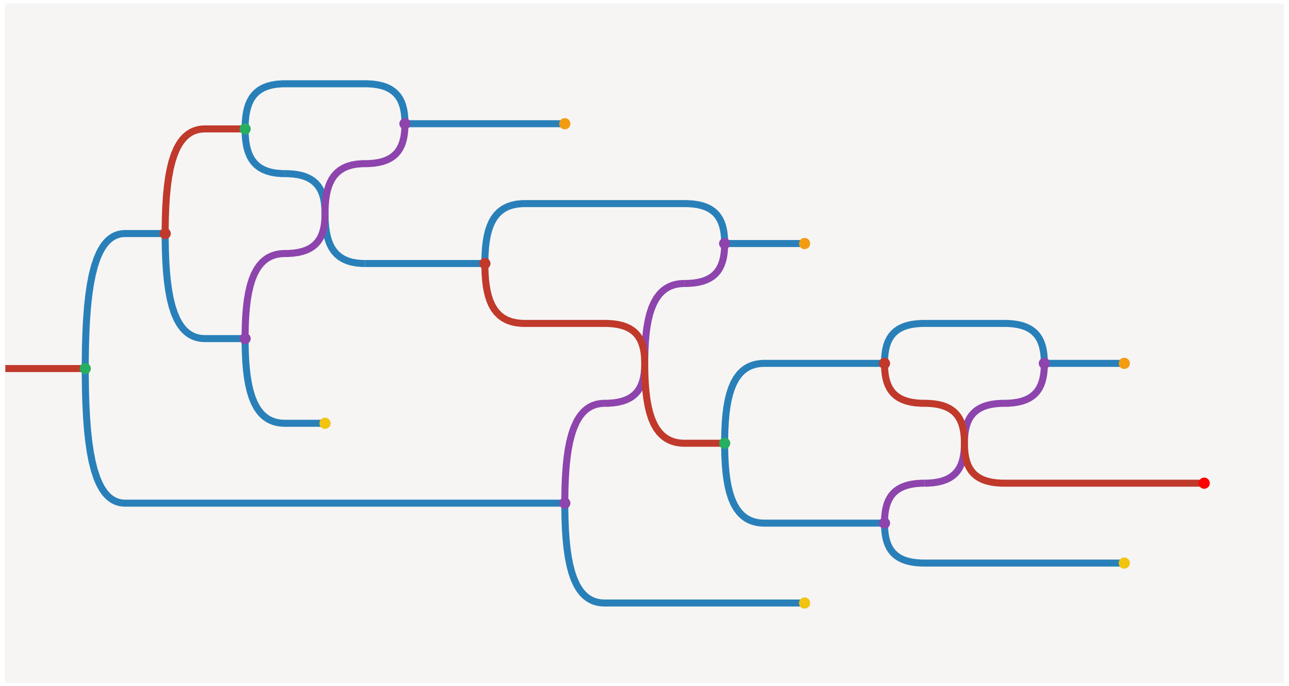
$$placeholder$$

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.


e







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.

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.

Definition 0.1.9 (Tree Adjoining Grammars with local constraints and links in `homotopy.io`)**.** A *Linguistic Tree Adjoining Grammar* is given by the following data:

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

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.
- $\Box, \Diamond$ are fresh symbols. $\Box$ 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 $\mathcal{N} \times \mathfrak{S} \times \{\Box, \Diamond\} \times \{*, \bar{*}\}$, and each leaf is an element of $\mathcal{N} \times \mathfrak{S} \times \{\Box, \Diamond\} \times \{*, \bar{*}\} \cup \Sigma$. 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.

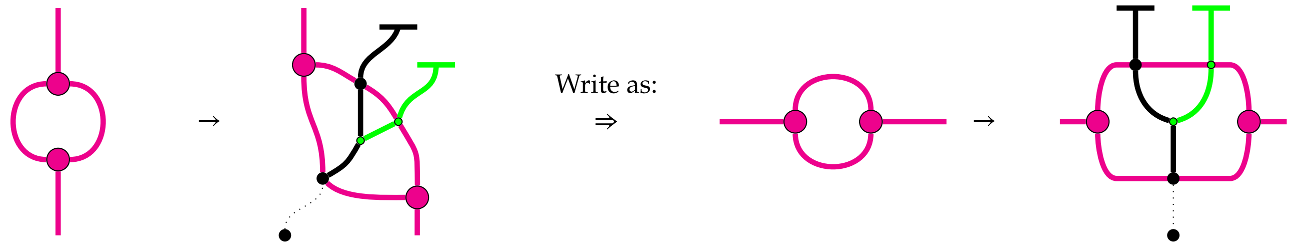
0.2 A generative grammar for text circuits

0.2.1 A circuit-growing grammar

There are many different ways to write an n-categorical signature that generates circuits. Mostly as an illustration of expressive capacity, 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 (roughly) syntactic order, and like mushrooms on soil, the circuits will appear as the mycelium underneath words.

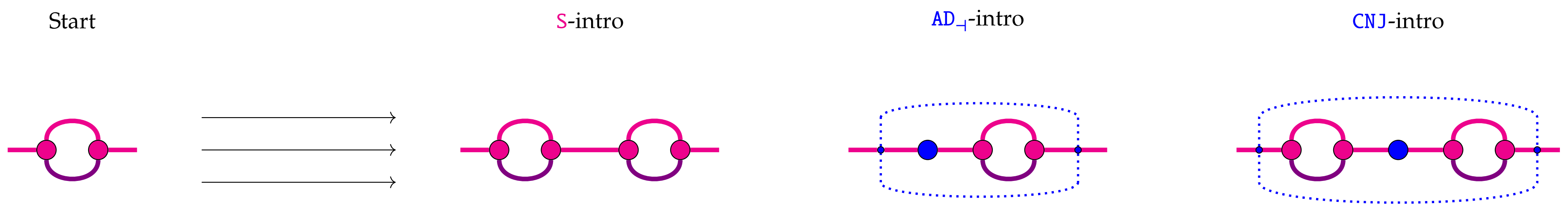
Simplifications: Propositions only, no determiners, only one tense, no morphological agreement between nouns and their verbs and referring pronouns, and we assume that adverbs, adverbial adjunctions, and adjectives stack indefinitely and without further order requirements; e.g. `Yesterday yesterday yesterday Alice happily secretly finds red big toy shiny car that he gives to Bob.` 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 can principle be overcome by the techniques we developed in Section **??** for restricted tree-adjoining and links.

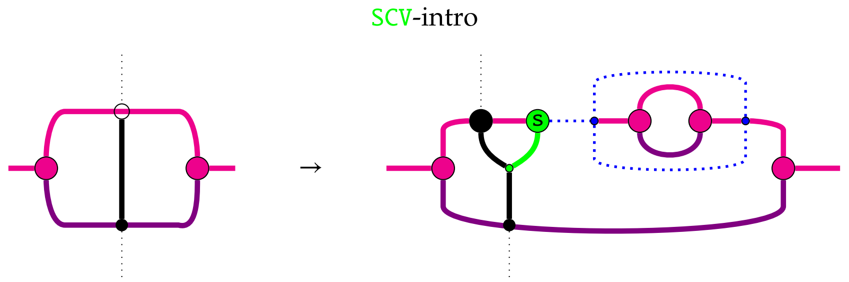
How to read the following diagrams: We work in a dimension where wires behave symmetric monoidally by homotopy, and the signature still works if interpreted in a compact closed setting. We start each derivation with a pink sentence bubble, which we depict for aesthetic purposes as horizontal, which amounts to picking slightly different axes by which to read diagrams. The insides of the sentence bubble will fill up with a circuit, and labelled words will appear on the top surface bubble like stylised mushrooms, to be read off left-to-right. We will only express the rewrite rules; the generators of lower dimension are implicit.



0.2.2 Sentences

Before the contents of a sentence are even decided, we may decide to (top row, left to right) get another sentence ready, introduce an adverbial adjunction (such as `yesterday`), or introduce a conjunction of two sentences (such as `because`). Introducing new words yields dots that carry information of the grammatical category of the word, and there are separate rewrites that allow dots to be labelled according to the lexicon. In the bottom, we have a rewrite that allows a sentence with one unlabelled noun to be the subject of a "sentential complement verb", abbreviated SCV: these are verbs that take sentences as objects rather than other nouns, and they are typically verbs of cognition, perception, and causation, such as Alice suspects Bob drinks. The blue-dotted lines are just syntactic guardrails that correspond to the holes in boxes of circuits later.

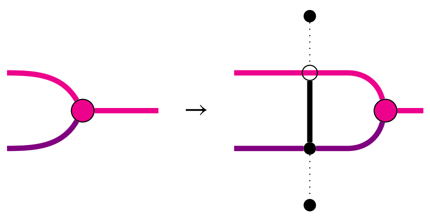




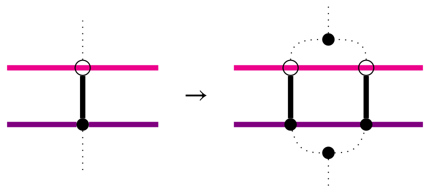
0.2.3 *Simple sentences*

Within each sentence bubble, each derivation starts life as a "simple sentence", which only involves nouns and a single verb, which is either an intransitive verb that takes a single noun argument, or a transitive verb that takes two. You just can't have a (propositional) sentence without at least a noun and a verb. Within each sentence, we may start introducing nouns. From left-to-right; we may introduce a new noun (which comes with tendrils that extrude outside the sentence bubble for later use to resolve pronominal reference); split a noun so that the same noun-label is used multiply; and label a noun *if it is saturated – depicted by a solid black* *circle* which includes a copy of the label for bookkeeping purposes when resolving references.

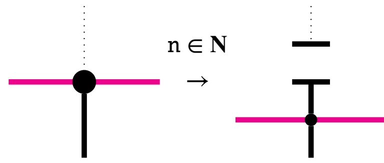
N-intro



N-split

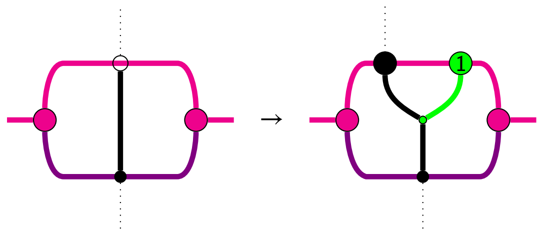


N-label

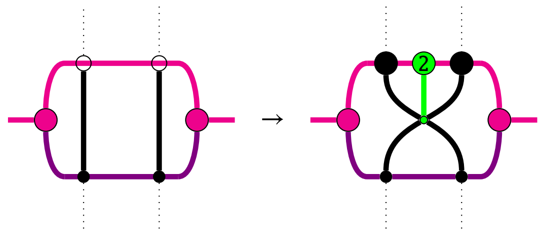


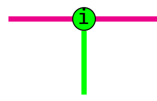
Nouns require verbs in order to be saturated. From left-to-right; if there is precisely one unlabelled noun, we may introduce an unlabelled intransitive verb and saturate the noun so that it is now ready to grow a label; or if there are two unlabelled nouns, we may introduce an unlabelled transitive verb on the surface and saturate the two nouns that will be subject and object; and verbs may be labelled.

IV-intro



TV-intro

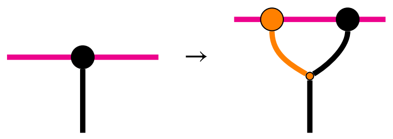




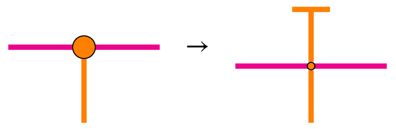
0.2.4 *Modifiers*

Modifiers are optional parts of sentences that modify (and hence depend on there being) nouns and verbs. We consider adjectives, adverbs, and adpositions. From left to right; we allow adjectives to sprout immediately before a saturated noun, and we allow adverbs to sprout immediately before any verb.

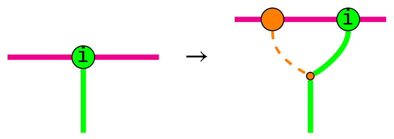
ADJ-intro



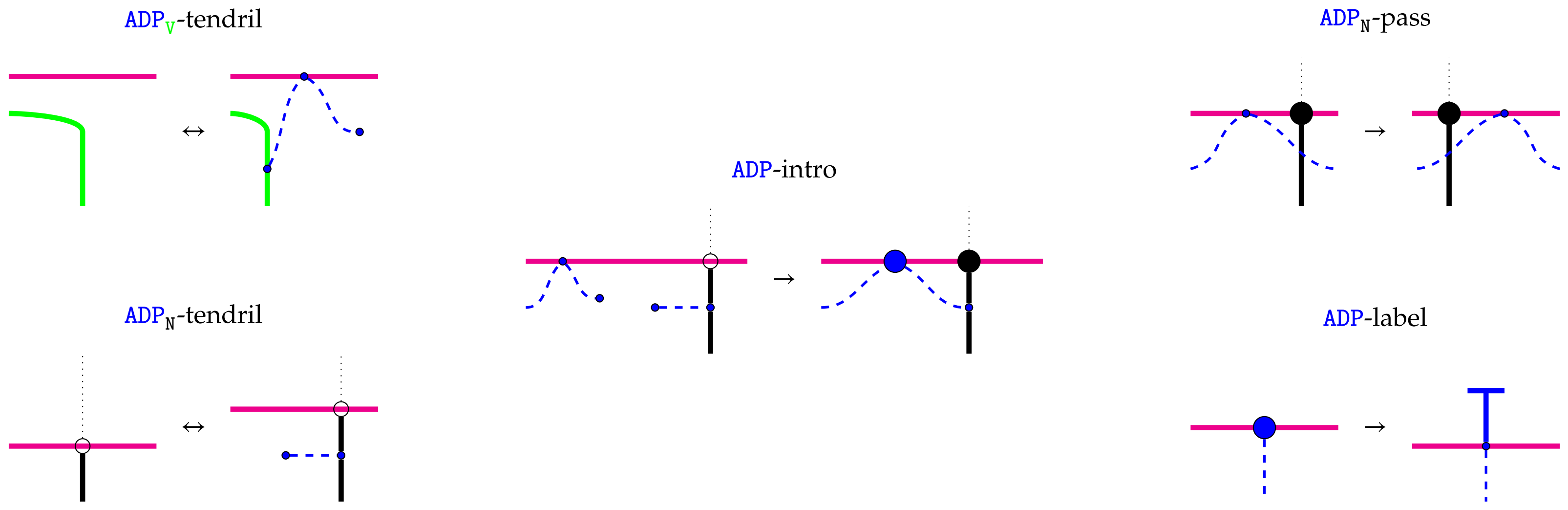
ADJ-label



ADV-intro



Adpositions modify verbs by tying in an additional noun argument; e.g. while runs is intransitive, runs `towards` behaves as a transitive verb. Some more advanced technology is required to place adpositions and their thematic nouns in the correct linear order on the surface. In the left column; an adposition tendril can sprout from a verb via an unsaturated adposition, seeking an unsaturated noun to the right; an unsaturated noun can sprout an tendril seeking a verb to connect to on the left. Both of these rewrites are bidirectional, as tendrils might attempt connection but fail, and so be retracted. In the centre, when an unsaturated adposition and its tendril find an unsaturated noun, they may connect, saturating the adposition so that it is ready to label. In the right column; an unsaturated adposition may move past a saturated noun in the same sentence, which allows multiple adpositions for the same verb; finally, a saturated adposition can be labelled.



0.2.5 *Rewriting to circuit-form*

Resolving references

Connecting circuits

Example 0.2.1.

0.2.6 *Extensions I: relative and reflexive pronouns*

Subject relative pronouns

Example 0.2.2.

Object relative pronouns

Example 0.2.3.

Reflexive pronouns

Example 0.2.4.

0.2.7 Extensions II: grammar equations

Attributive vs. predicative modifiers

Example 0.2.5.

Copulas

Example 0.2.6.

Possessive pronouns

Example 0.2.7.

0.2.8 Extensions III: higher-order modifiers

Intensifiers

Example 0.2.8.

Comparatives

Example 0.2.9.

0.2.9 Equivalence to internal wirings

0.2.10 Text circuit theorem

0.2.11 Related work

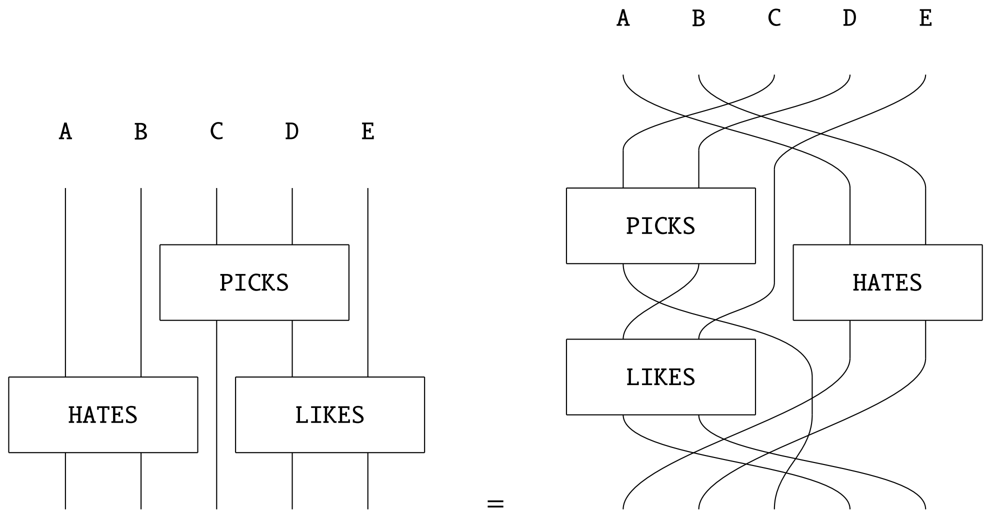
0.3 *Text circuits: details, demos, developments*

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 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 accommodate previously computed vector embeddings of words.

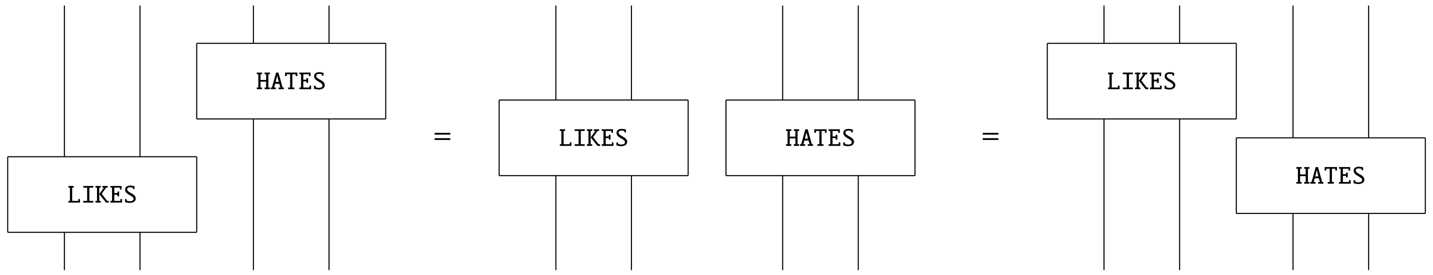
In practice, the process of obtaining transparently computable text goes through two phases. First, one has to obtain text circuits from text, which is conceptually simple: typelogical parsers for sentences can be modified to produce circuit-components rather than trees, and a separate pronomial resolution 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 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.

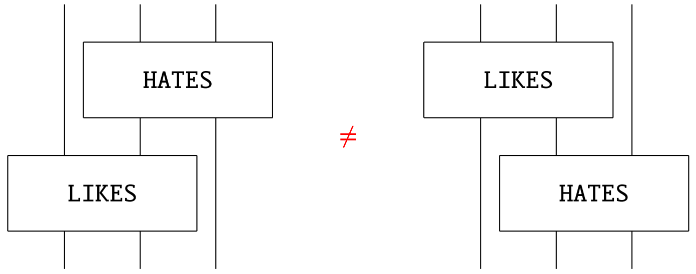
Convention 0.3.1. Sometimes we allow wires to twist past each other, and we consider two circuits the same if their gate-connectivity is the same:



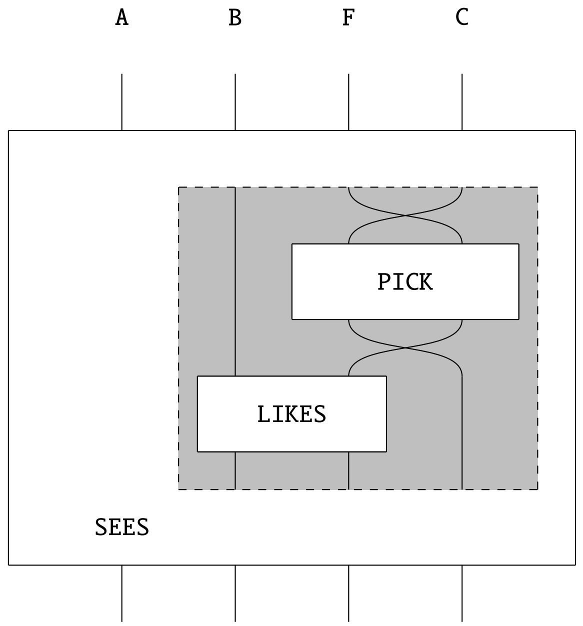
Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel:



We do care about output-to-input connectivity, so in particular, we **do not** consider circuits to be equal up to sequentially composed gates commuting past each other:



Example 0.3.2. The sentence ALICE SEES BOB LIKES FLOWERS THAT CLAIRE PICKS can intuitively be given the following text circuit:


A
B
F
C
PICK
LIKES
SEES