STRING DIAGRAMS FOR TEXT

VINCENT WANG-MAŚCIANICA

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

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
2023

Contents

(Acknowledgements will go in a margin note here.)

1
Text circuits for syntax

We want to establish that there is a systematic correspondence between text circuits and grammatically acceptable text, which would allow us to use text circuits as a generative grammar without further justification. First we show that context-free grammars, string-rewrite systems, and tree-adjoining grammars are all special cases of higher-dimensional rewriting systems called weak n-categories. Then we provide a "circuit-growing" grammar in terms of a weak n-categorical signature that simultaneously generates strings of grammatically acceptable text and its "deep structure" in terms of text circuits, from which we obtain the desired correspondence between text circuits and text. We close with demonstrations of how the syntactic theory of text circuits may be modified and expanded.

1.1 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. 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.

In regular or 1-category theory we are interested in objects up to isomorphism rather than equality - because isomorphic objects are as good as one another. Concretely, two objects X and Y are isomorphic when there exist morphisms $f : X \to Y$ and $g : Y \to X$ such that $g \circ f = 1_X$ and $f \circ g = 1_Y$. Lifting this philosophy

to n-categories, we can replace equalities $f = g$ of k-morphisms (for $k < n$) with $(k+1)$-isomorphisms $f \simeq g$. This extends also to the equalities in the definition of a $(k+1)$-isomorphism, all the way up to dimension n at which point we are content with ordinary equality again. The idea of replacing equality with isomorphism can even extend to the associativity, unitality and interchange laws governing the composition operations. If left to be ordinary equalities, we speak of a *strict* n-category. If only witnessed by a coherent system of isomorphisms, we have a *weak* n-category. Unsurprisingly the weak variant is much harder to formalise, but also much more expressive once n > 2.

Mathematicians, computer scientists, and physicists may have good reasons to work with n-categories (**?**), but what is the value proposition for formal linguists? 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. Here, we work with a semi-strict model in which associativity and unitality remain strict, while interchange is weak. This fact is hidden behind a convenient graphical notation, and we do not miss out on any of the expressivity of fully weak higher categories. Additionally, this system has an online proof assistant `homotopy.io`. For formal details the reader is referred to (**????**). In this setting we will demonstrate how weak n-categories 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.

1.1.1 *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 1.1.1.

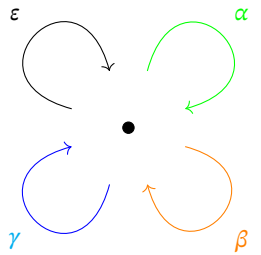


Figure 1.1: The category in question can be visualised as a commutative diagram.

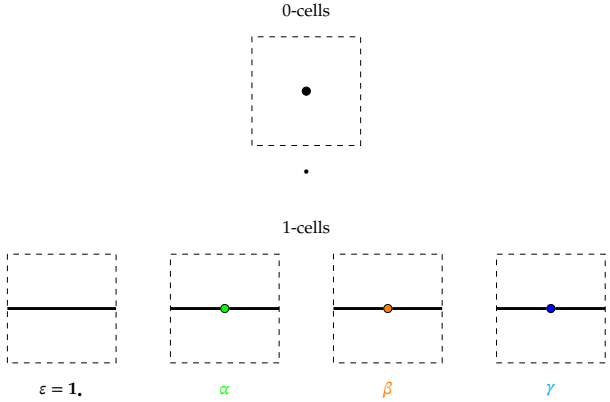
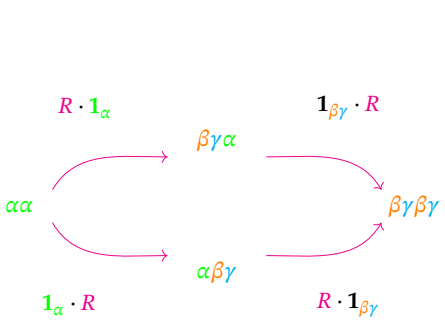
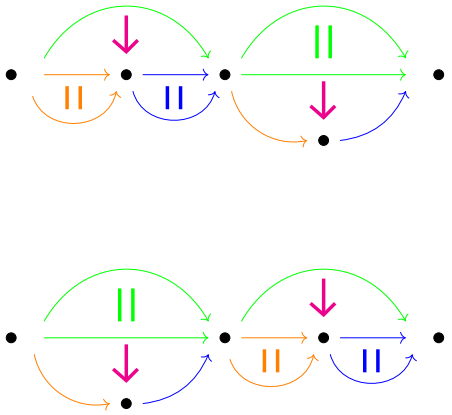


Figure 1.2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from 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 $\epsilon \cdot x = x = \epsilon \cdot x$.

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.





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; between any such pair of 2-cells there is only one

Figure 1.3: For a concrete example, we can depict the string $\alpha \cdot \gamma \cdot \gamma \cdot \beta$ as a morphism in a commuting diagram.

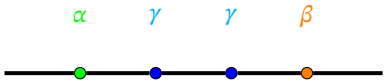


Figure 1.4: The string-diagrammatic view, where $\star$ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

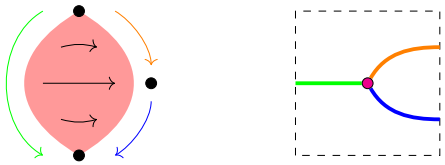
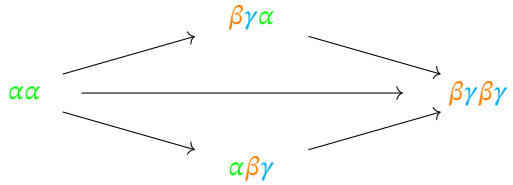
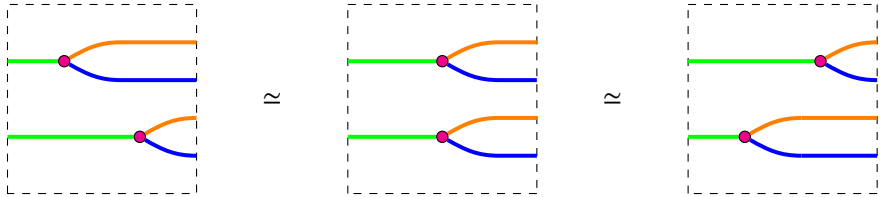


Figure 1.5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual of this view gives us a string diagram for the 2-cell R.

3-cell, which is the identity. 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. In this case, the demotion of interchange equalities to isomorphisms means that Bob is dealing with a *weak* 1-object-2-category (in general the demotion of equalities to isomorphisms is not synonymous to weakness.) 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 equalities, rather than isomorphisms.

1.1.2 *Tree Adjoining Grammars*

Definition 1.1.1 (*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 $\downarrow : \mathcal{N} \to \mathcal{N}^{\downarrow}$ 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 $* : \mathcal{N} \to \mathcal{N}^{*}$ – 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 auxiliary 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 β's foot node $\mathtt{X}^{\star}$.

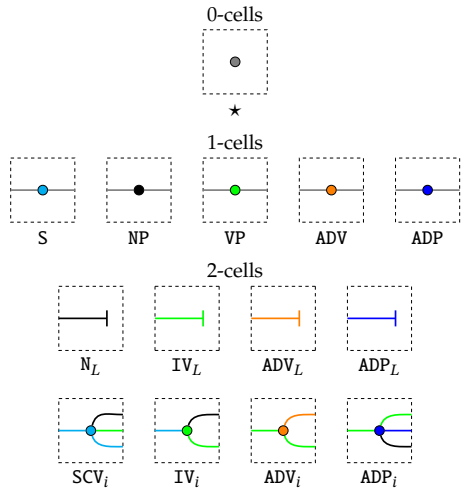


Figure 1.6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for `Alice sees Bob quickly run to school`.

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.

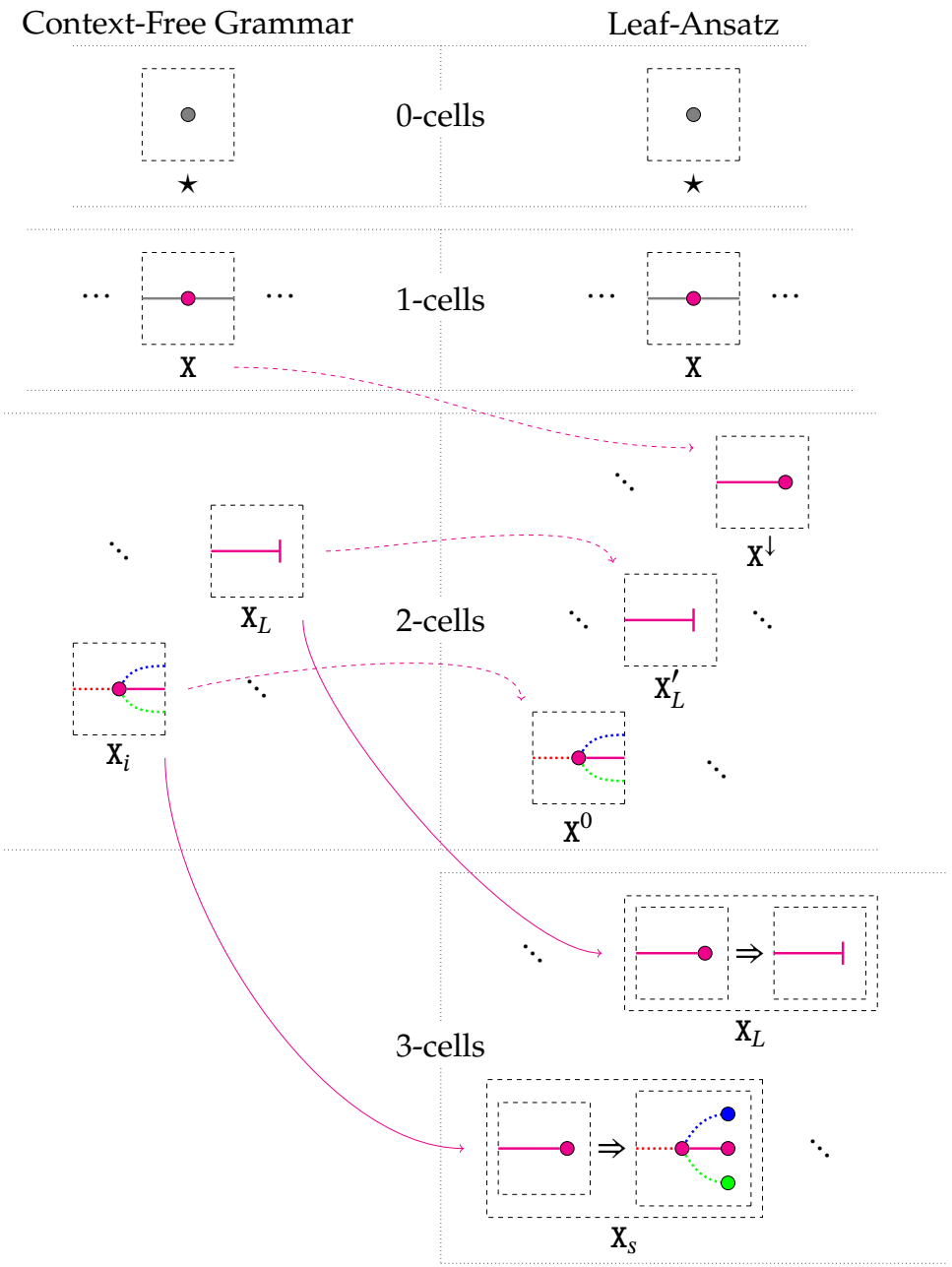


Figure 1.7:

Construction 1.1.4 (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 1.1.5. Leaf-ansatzes of CFGs are precisely tree adjoining grammars (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.

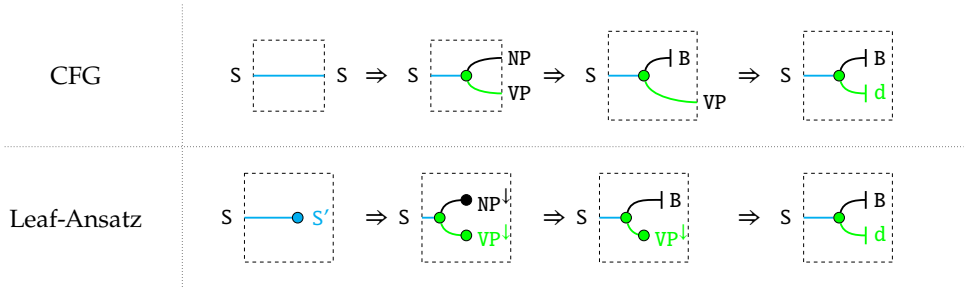


Figure 1.8: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction between nonterminals and terminals.

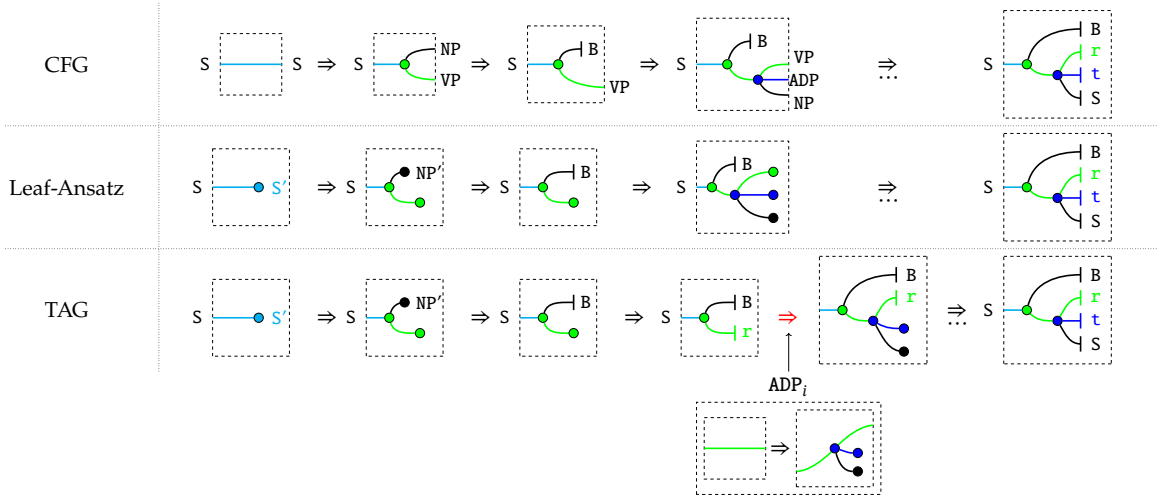


Figure 1.9: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted 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.

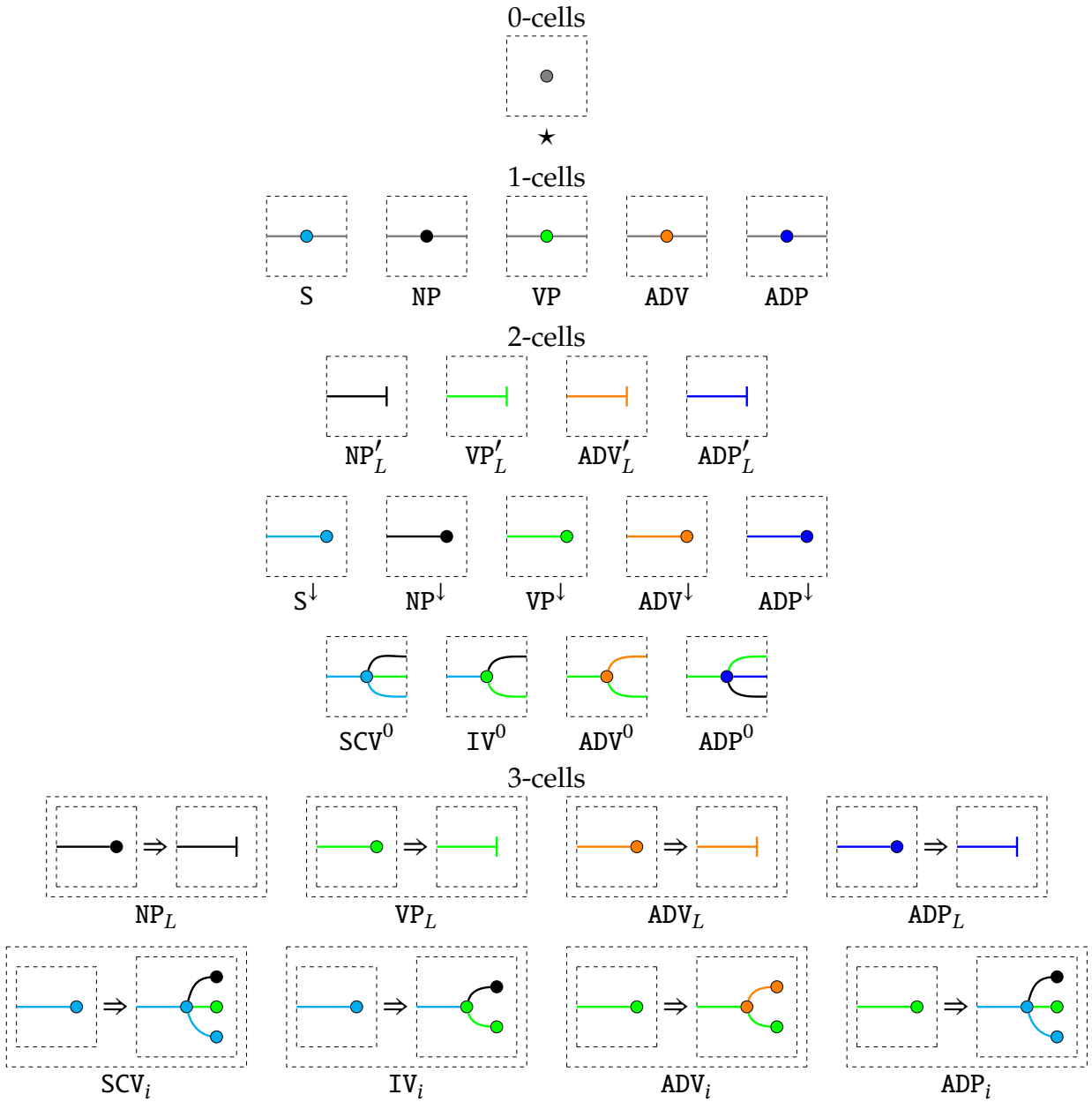


Figure 1.10: Leaf-ansatz signature of `Alice sees Bob quickly run to school` CFG.
1-cells correspond to types. There are three kinds of 2-cells organised in rows; terminals, substitutable ansätze that convert wires into bulbs; and the symbolic rewrites of CFGs respectively. There are two kinds of 3-cells similarly organised in rows; terminal-rewrites that replace a bulb with a terminal, and rewrites that mimic the CFG rewrites on ansätze. Note the one-to-one correspondence between the 2-cells and 3-cells for CFG rewrites and terminals.
In more detail, 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.

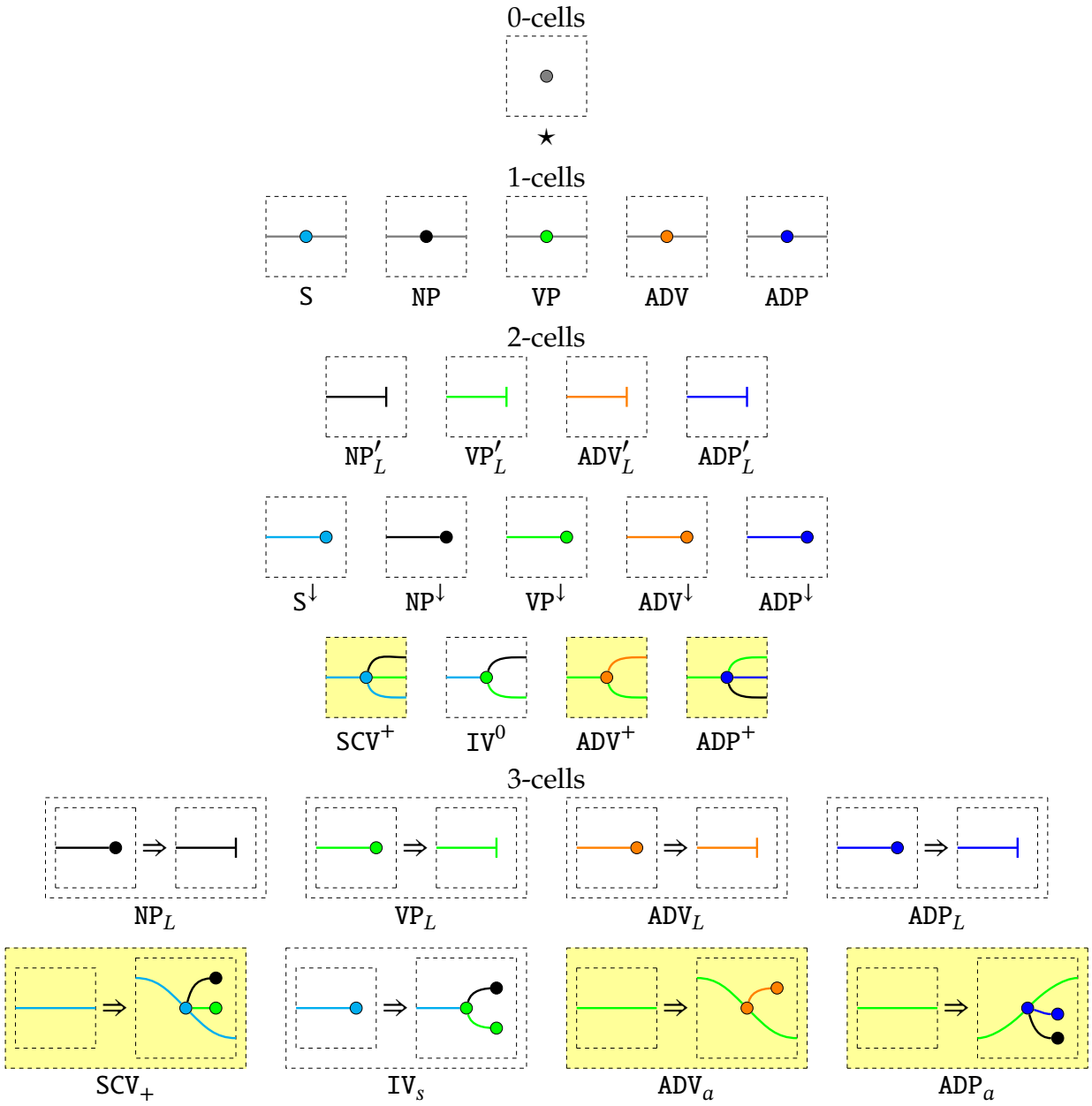
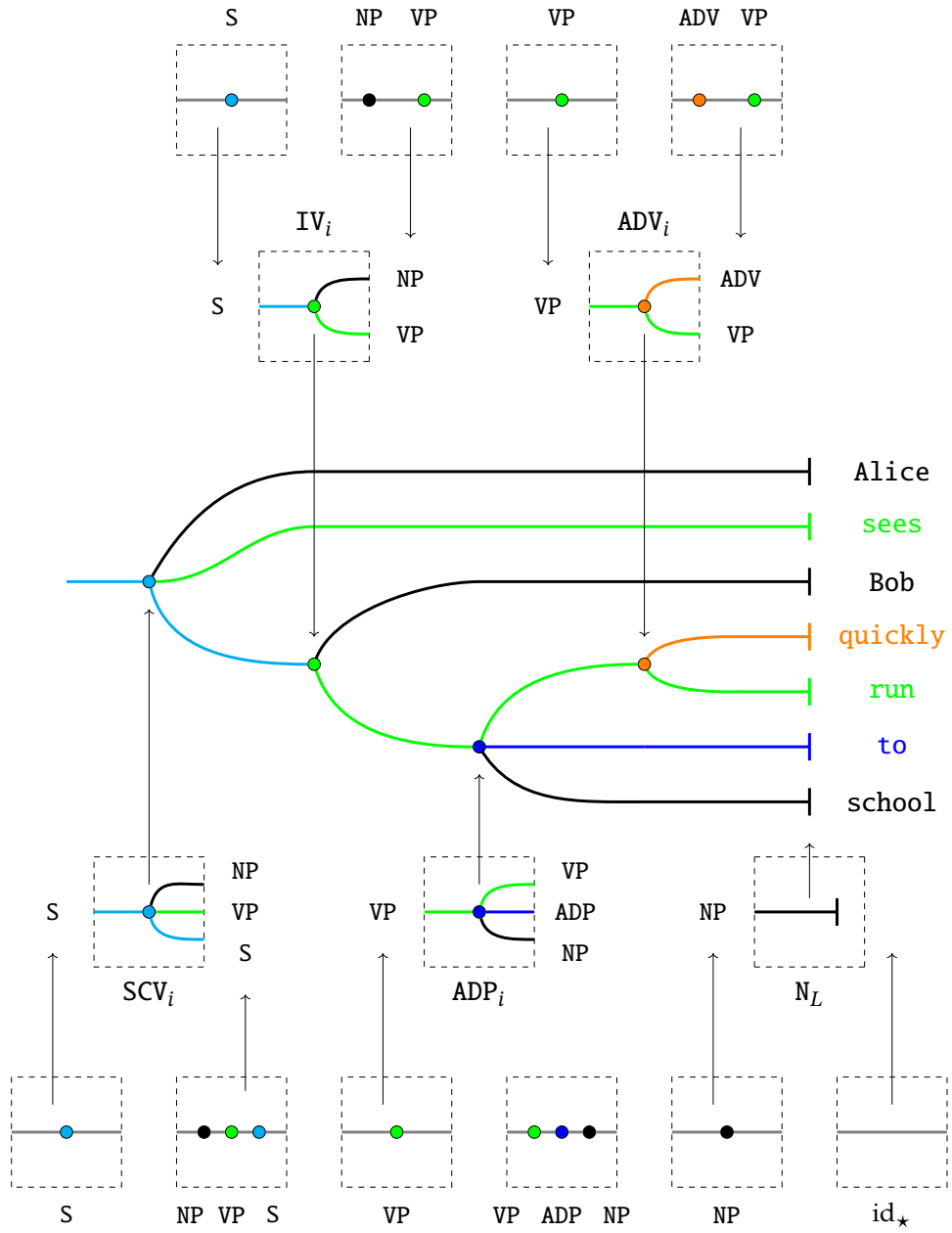


Figure 1.11: TAG signature of `Alice sees Bob quickly run to school`. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction yields as a corollary an alternate proof of Theorem 6.1.1 of (**?**) that recovers CFGs as TAGs.

Corollary 1.1.7. 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 1.7 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 $\mathfrak{G}'$. □

Figure 1.12: Reading the central diagram in the main body from left-to-right, we additionally depict the breakdown of the complete derivation in terms of the constitituent 2-cells, and the source and target 1-cells. Evidently, all context-sensitive grammars may be viewed as finitely presented 1-object-2-categories by considering multi-input-multi-output rewrites. More broadly, any string rewriting system is recoverable in the presence of higher dimensional cells. My source for that is that I made a Turing machine in `homotopy.io` and executed busy-beaver on it as a homework exercise when Jamie Vicary taught Categorical Quantum Mechanics at Oxford.



1.1.3 *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 1.1.8 (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.

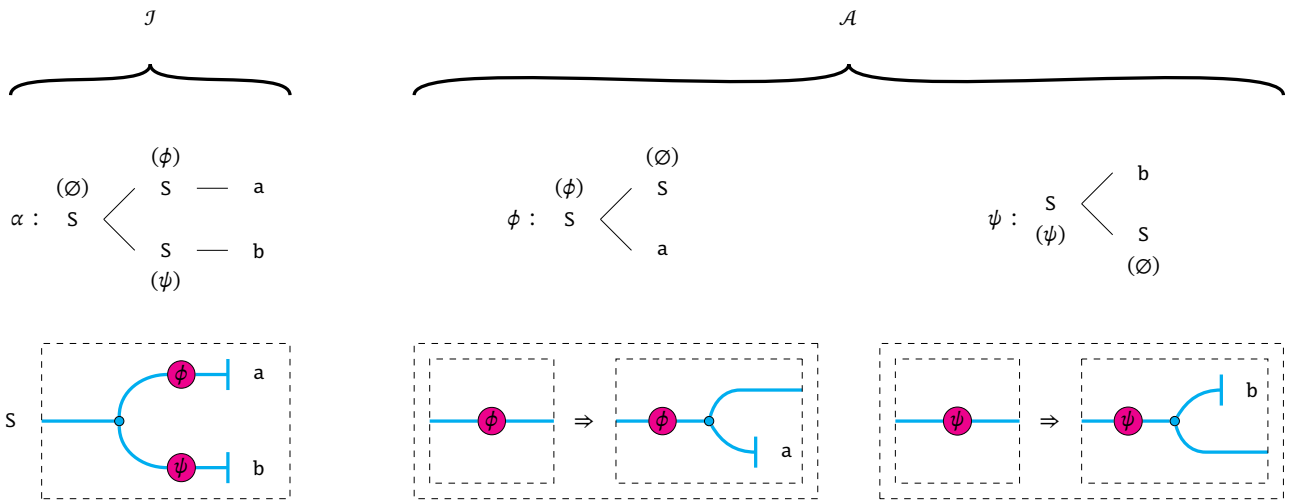


Figure 1.13: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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.

1.1.4 *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.

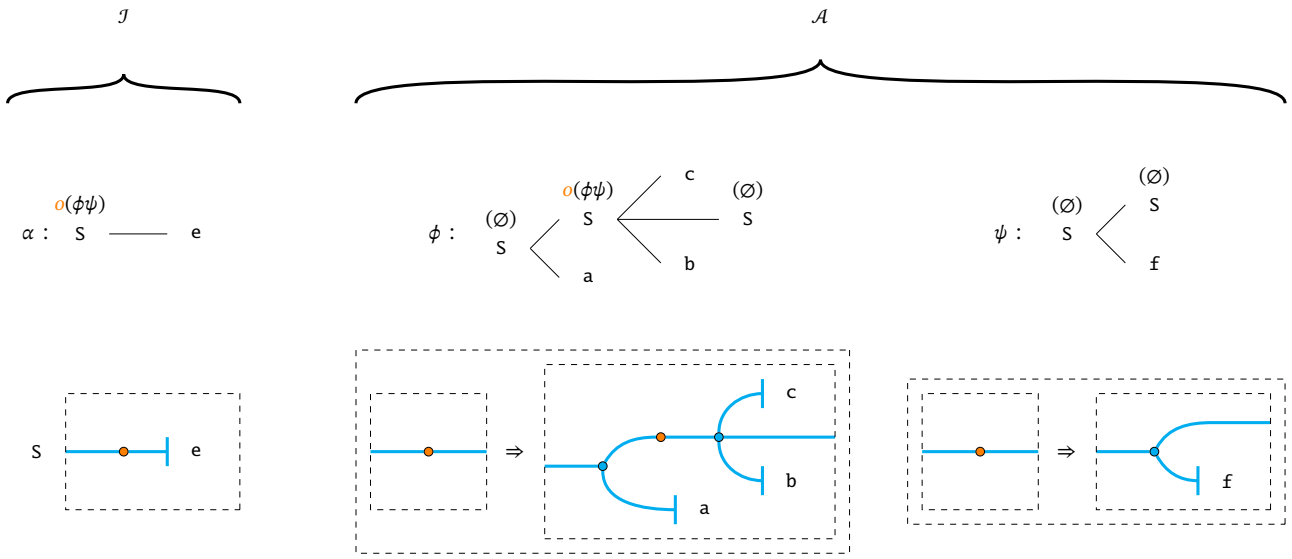
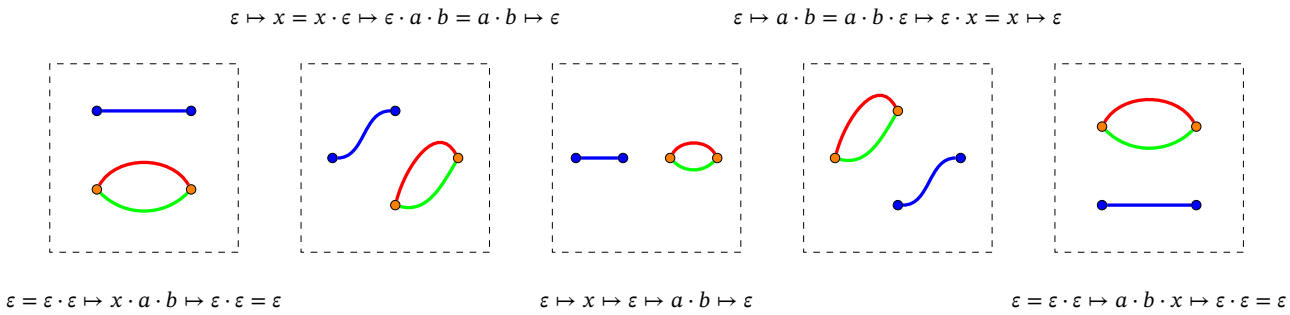


Figure 1.14: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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. Such global acceptance conditions are hacky but common, and in this case it is efficiently verifiable that diagrams do not contain certain generators.

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 1.15: 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.



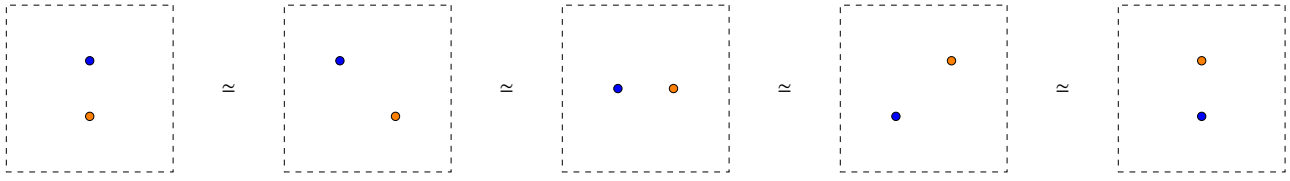


Figure 1.16: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting 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.

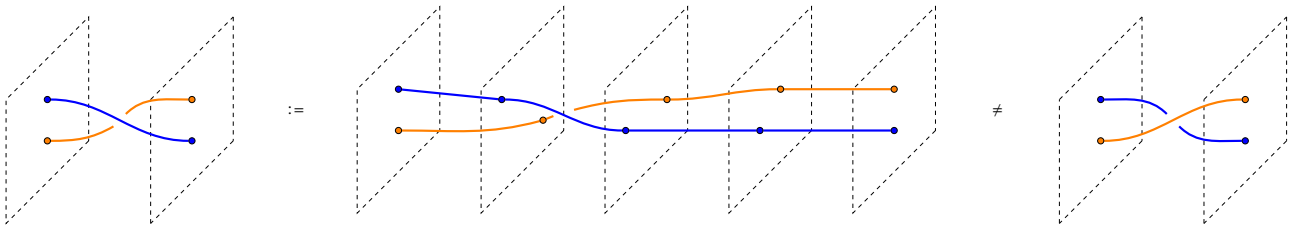


Figure 1.17: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies, which are continuous bijective transformations that don't let wires double back on themselves, we can identify two 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 am-

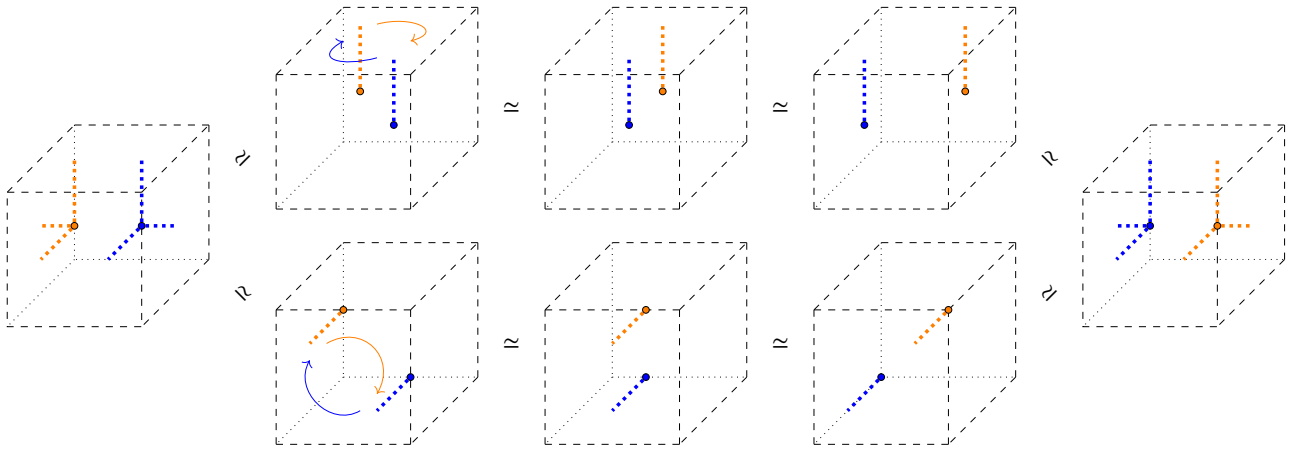


Figure 1.18: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise 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:

bient 2-dimensional space. In a (1× 0-cell)-(1× 1-cell)-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 $(1 \times 0, 1 \times 1, 1 \times 2)$-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 $(1 \times 0, 1 \times 1, 1 \times 2)$-4 category by promoting all 2-cells and higher to sit on top of $\mathbf{1}_{1_\star}$.

In general, the process of going from an n-category to an $(n+1)$-category with just one 0-cell is called *suspension*. So we have essentially that suspending the combinatorial data of planar string diagrams gives braidings, and suspending again gives symmetric monoidal twists: by appropriately suspending the signature, we power up planar diagrams to permit twisting wires.

Remark 1.1.9 (**THE IMPORTANT TAKEAWAY!**). Now have a combinatoric way to specify string diagrams (that generalises PROPs, modulo weak interchange) 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.

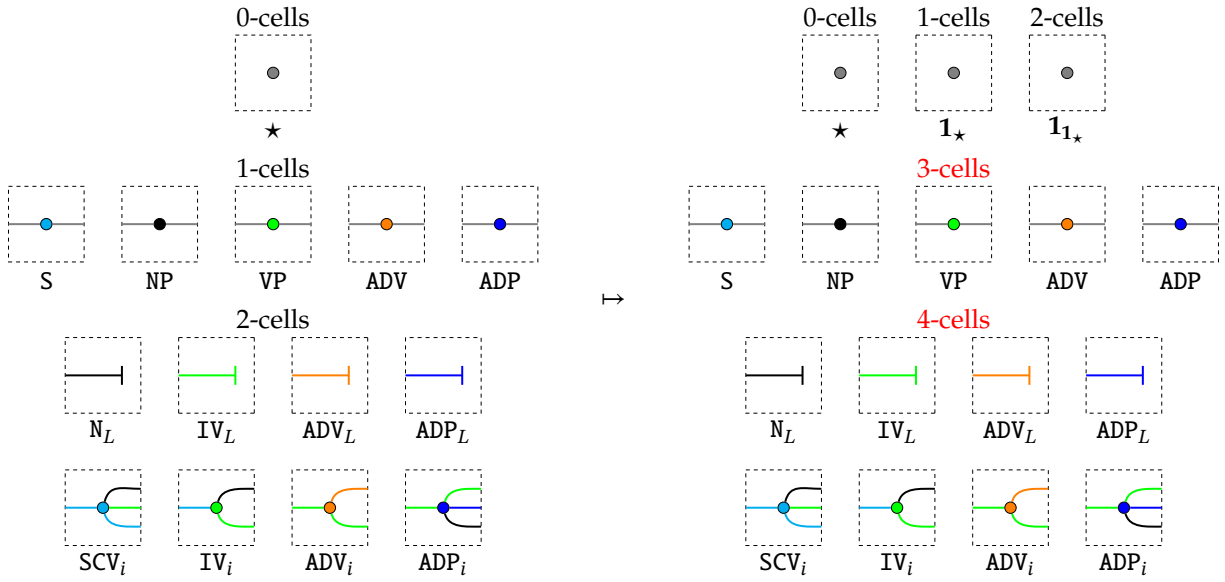


Figure 1.19: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with free swapping of symbol order.

1.1.5 TAGs with links

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 1.1.11. 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.

Definition 1.1.10 (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.

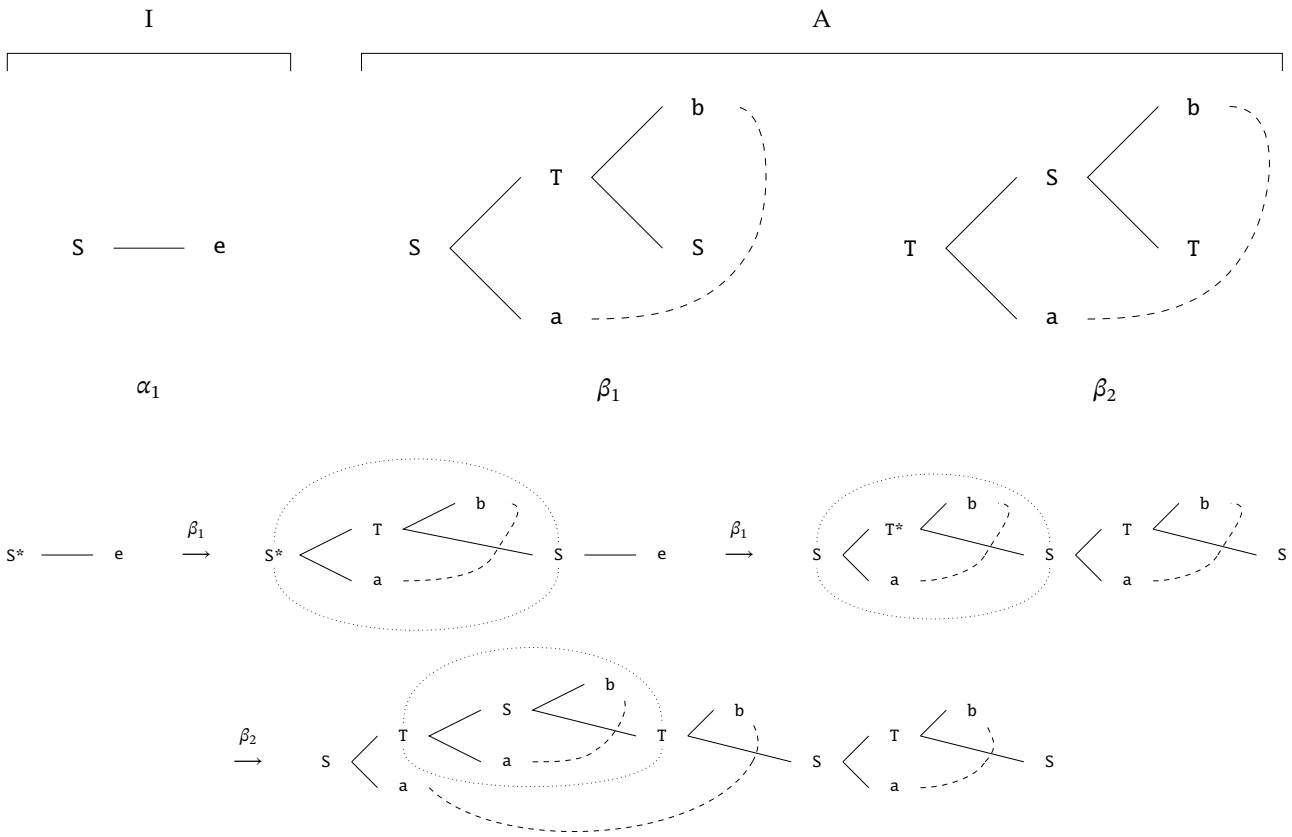


Figure 1.20: TAG signature and example derivation. Joshi stresses that adjoining *preserves* links, and that elementary trees may become *stretched* in the process of derivation, which are fundamentally topological constraints, akin to the "only (processive) connectivity matters" criterion identifying 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 1.2 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

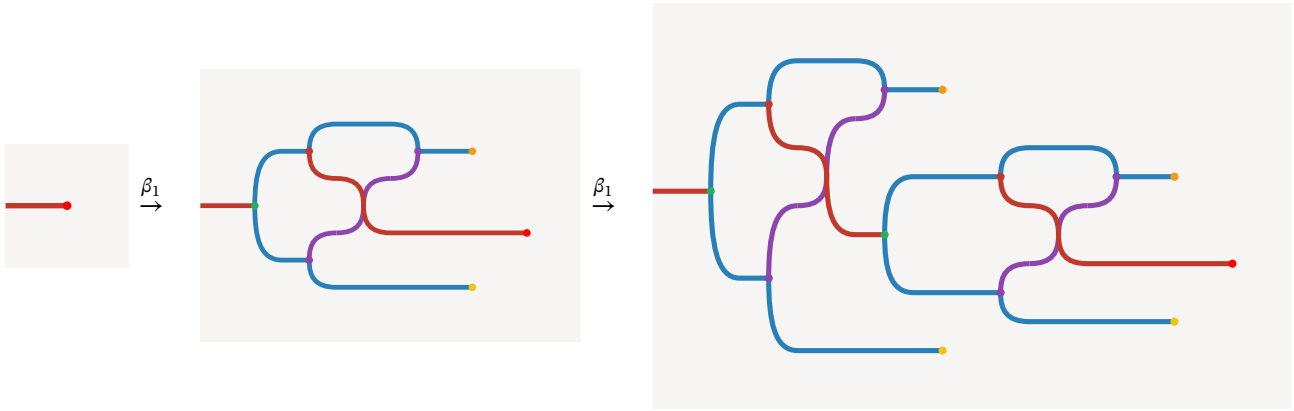


Figure 1.21: With our interpretation of TAGS as weak n-categorical signatures, We can recover each step of the same example derivation automagically in `homotopy.io`; just clicking on where we want rewrites allows the proof assistant to execute a typematching tree adjunction. In the process of interpretation, 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.

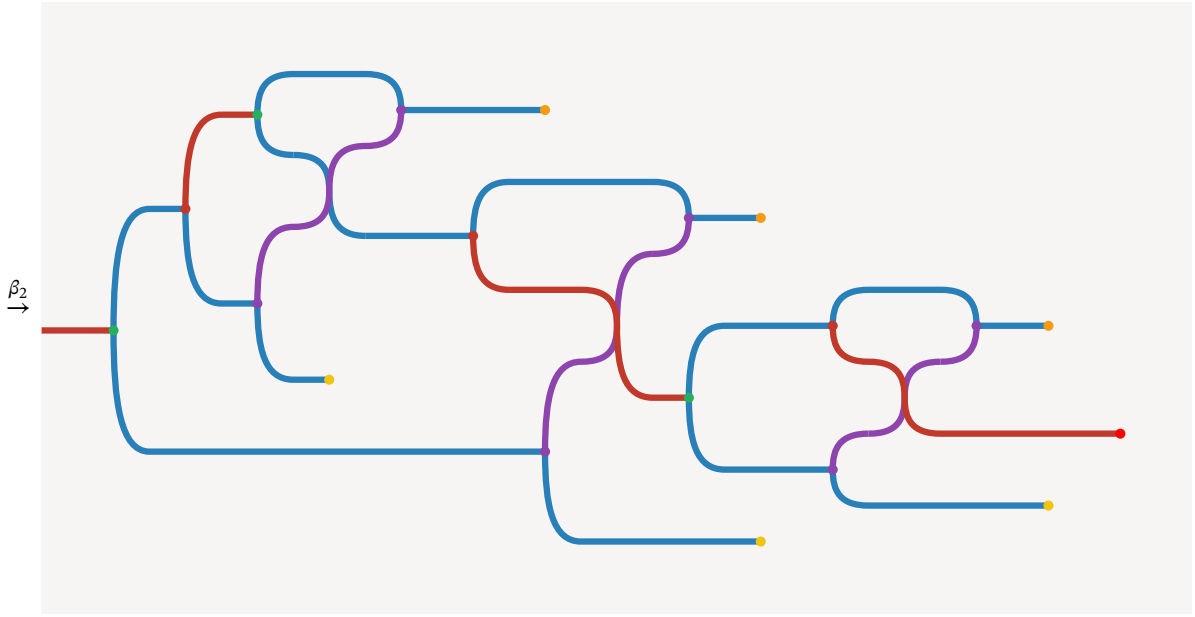


Figure 1.22: The intended takeaway is that even if you don't buy the necessity or formality of weak n-categories, there is always the fallback epistemic underpinning of a formal proof assistant for higher dimensional rewriting theories, which is rather simple to use if I have succeeded in communicating 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 (**?**)), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

Definition 1.1.12 (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.

1.1.6 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 1.1.13 (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 $n = (\mathbf{T}, \mathbf{S}, \dagger, \divideontimes)$, 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 $C_p := \langle c_1, c_2, \cdots c_i, c \ldots c_n \rangle$ 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.

1.2 A generative grammar for text circuits

1.2.1 A circuit-growing grammar

Definition 1.2.1 (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*

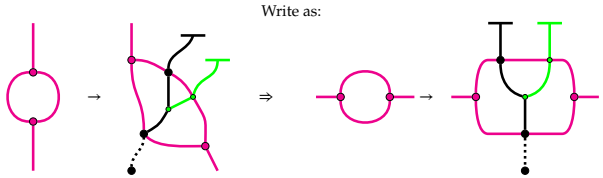


Figure 1.23: **How to read the diagrams in this section:** we will be making heavy use of pink and purple bubbles as frames to construct circuits. We will depict the bubbles horizontally, as we are permitted to by compact closure, or by reading diagrams with slightly skewed axes.

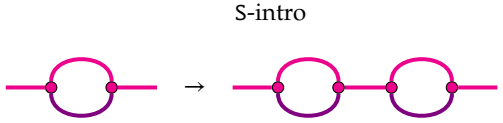


Figure 1.24: Every derivation starts with a single blank sentence bubble, to which we may append more blank sentences.

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. As a note to the reader, there are a lot of worked examples in this section, so if you get confused about why a rule is the way it is, just skip over it and hopefully there will be a clarifying example later on.

Simplifications and limitations: For now we only consider word types as in Definition 1.2.1, 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 and 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 1.1.6 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 1.25. 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.

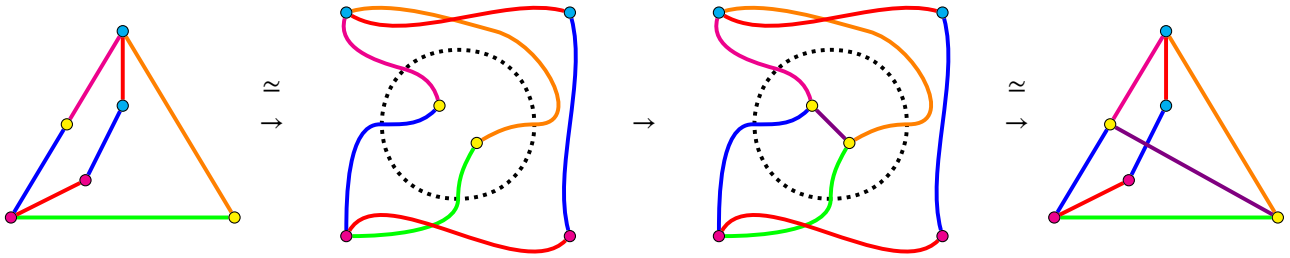


Figure 1.25: In this toy example, obtaining the same rewrite that connects the two yellow nodes with a purple wire using only graph-theoretically-local rewrites could potentially require an infinite family of rules for all possible configurations of pink and cyan nodes that separate the yellow, or would 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.

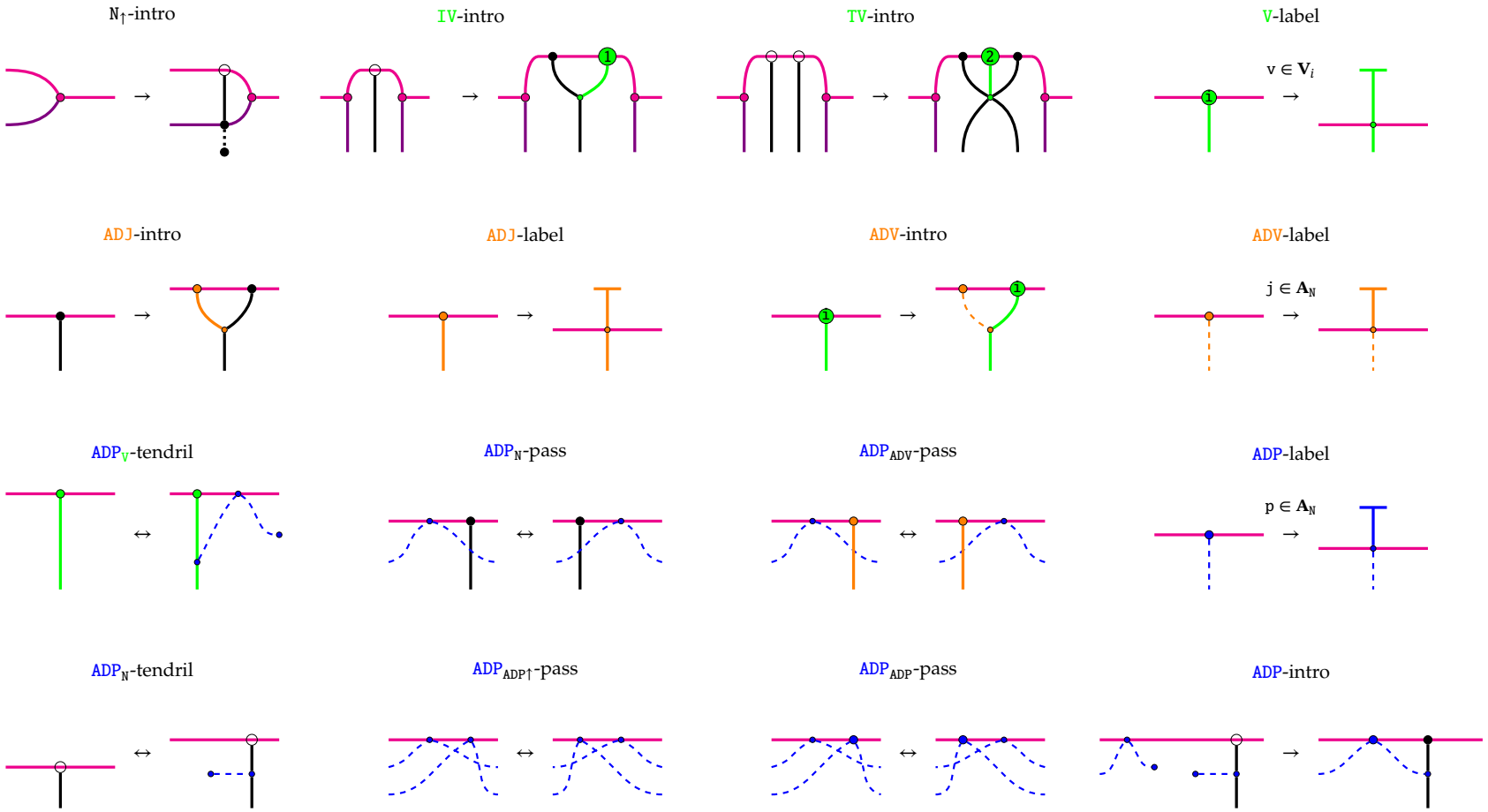
THE BROAD PLAN: We'll first display the rules and demonstrate their usage, then we'll prove the text circuit theorem by relating our rules to text.

THE RULES: We start with simple sentences that only contain a single intransitive or transitive verb, which correspond to gates and typed-boxes. Then we consider more general sentences with nesting sentential structure, which correspond to untyped-boxes. Then we introduce coreferential structure on nouns, which corresponds to symmetric monoidal composition of text circuits.

THE THEOREM: We characterise the expressive capacity of our rules for simple and complex sentences in terms of a context-sensitive grammar that corresponds to the surface structure of the derivations, which tells us that the generated text is sensible. Then we provide a mathematical characterisation of coreferential structure and a completeness result of our rules with respect to processive connectivity, which tells us that all circuit connectivity patterns are achievable. Then we (re)state and prove the text circuit theorem: that the fragment of language generated by the grammar surjects onto text circuits.

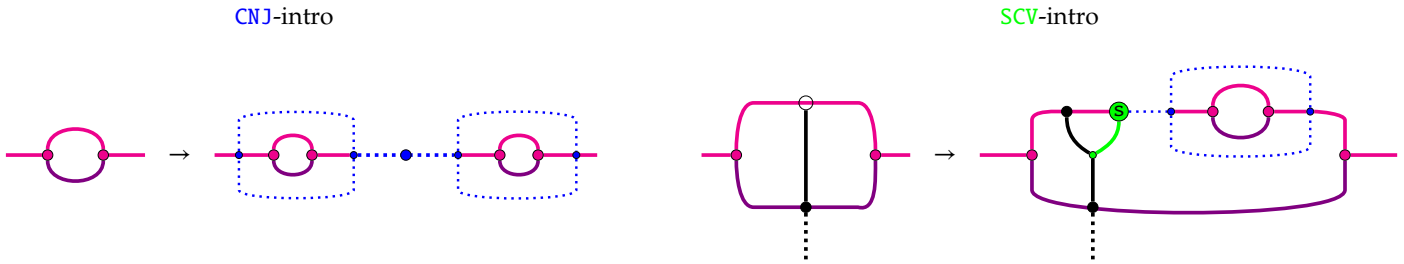
Aftermath: Finally, we examine how we may model extensions to the expressive capacity of text circuits by introduction of new rewrite rules.

Rules 1.2.2 (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:



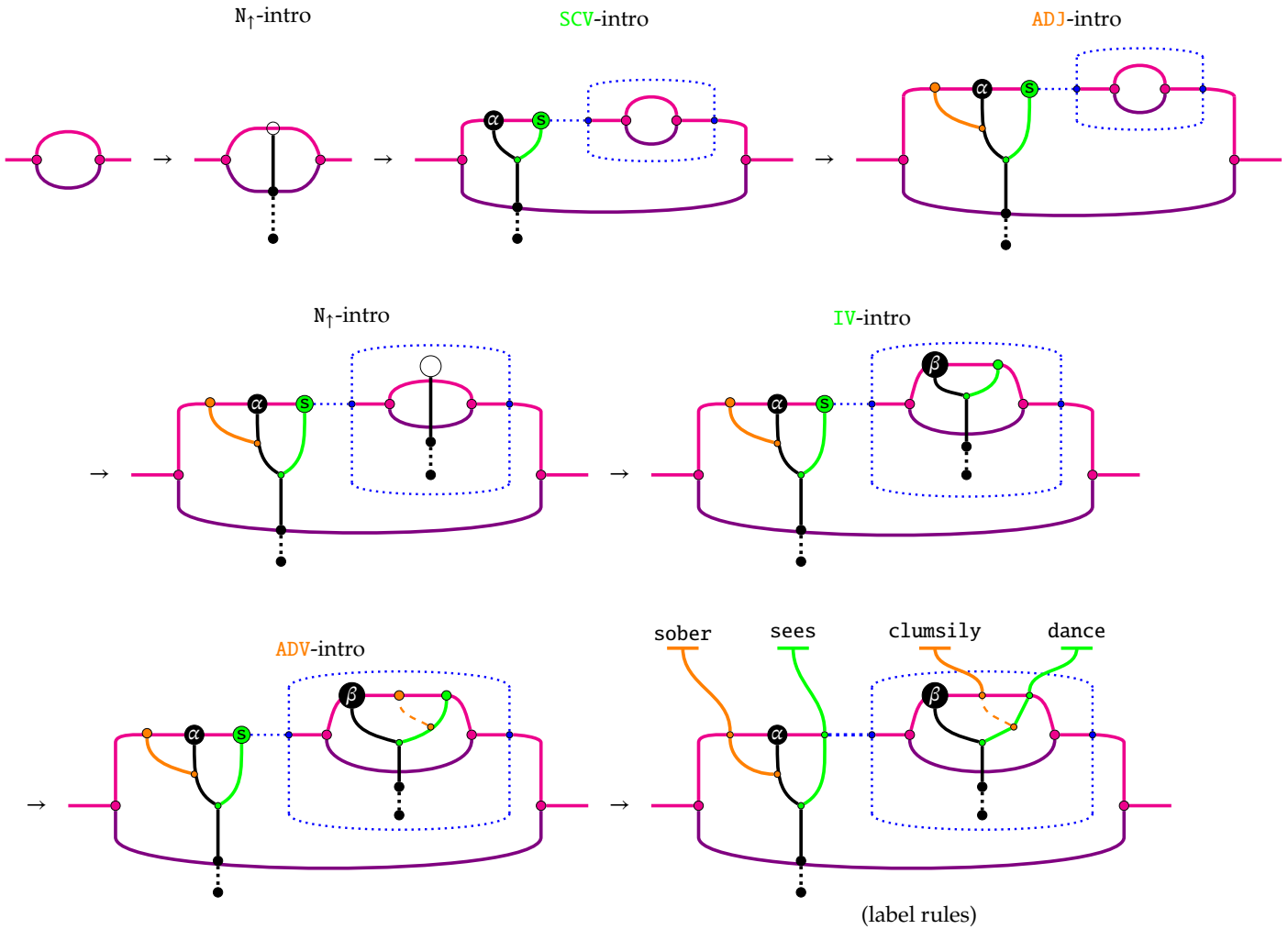
The $\mathtt{N}_\uparrow$-intro rule introduces new unsaturated nouns. The IV-intro and TV-intro rules apply when there are precisely one or two unsaturated nouns in the sentence respectively, saturating their respective nouns. Adjectives may be introduced immediately preceding saturated nouns. Adverbs may be introduced immediately preceding verbs. To capture context-sensitive placement of adposition introductions, the $\mathtt{ADP}_\mathtt{V}$-tendril rule allows an unsaturated adposition to succeed 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. 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 reversible.

Rules 1.2.3 (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.

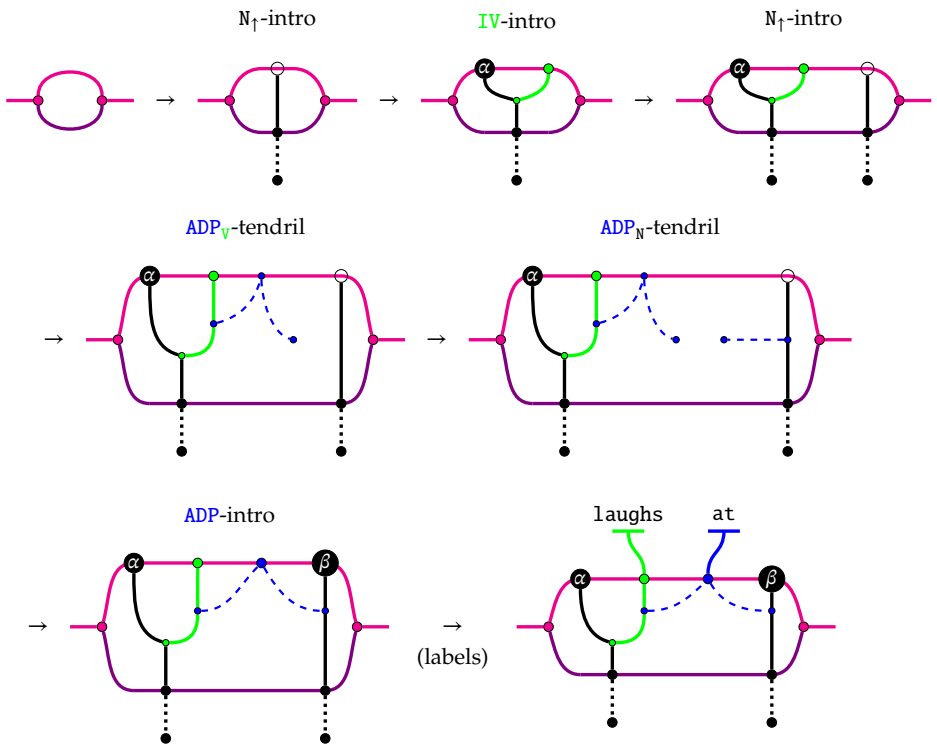


The dotted-blue wires do not contentfully interact with anything else, but the points at which they connect on the surface magenta wire serve as blockers that prevent overgeneration cases where adpositional phrases might interject between SCV verbs and their sentential complement, e.g. `Alice sees at lunch Bob drink`. Later, they serve as visual indicators for the contents of untyped-boxes in text circuits.

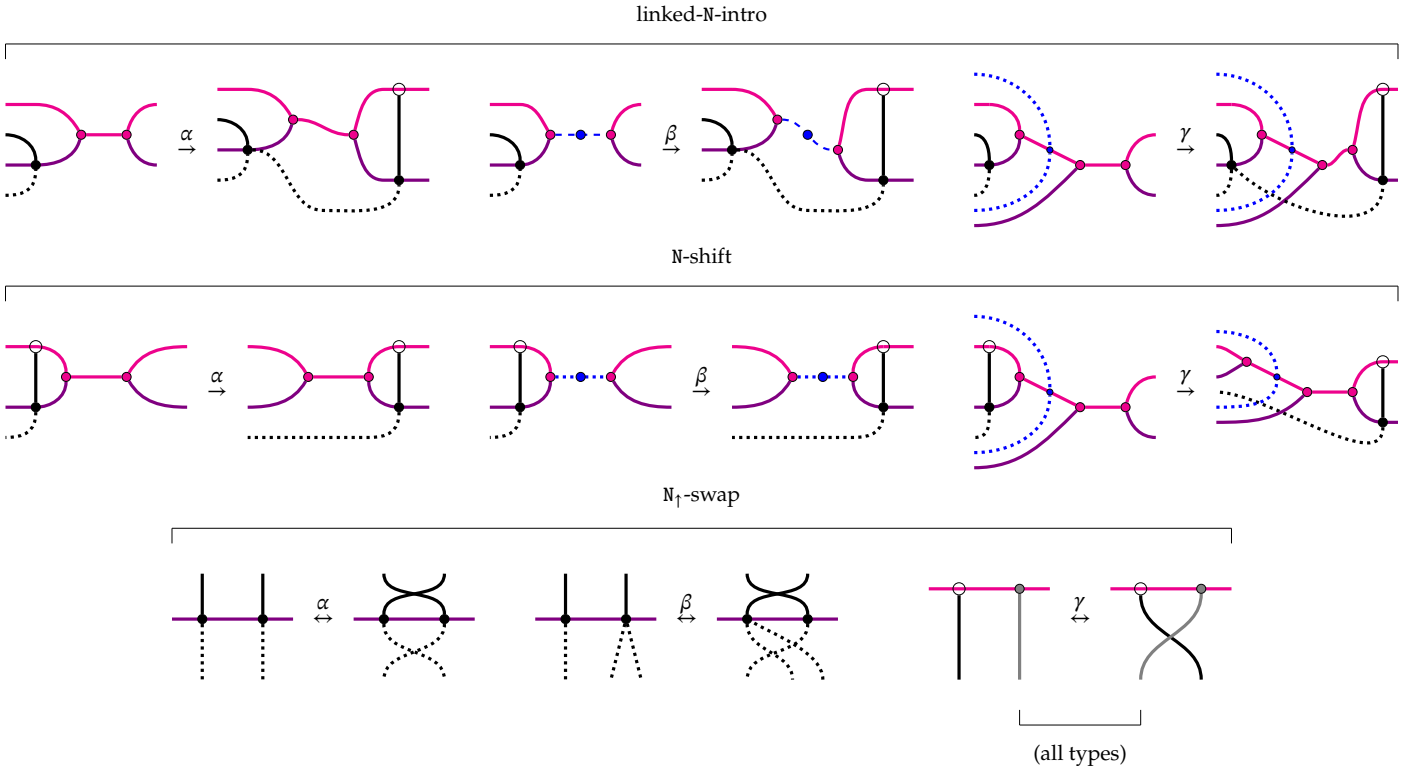
Example 1.2.4 (`sober` α `sees drunk` β `clumsily dance.`). Now we can see our rewrites in action for sentences. As a matter of convention – reflected in how the various pass- rules do not interact with labels – we assume that labelling occurs after all of the words are saturated. We have still not introduced rules for labelling nouns: we delay their consideration until we have settled coreferential structure. For now they are labelled informally with greeks.



Example 1.2.5 (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. It is an invariant of the rules by construction that left-to-right traversal of the surface is always meaningful and yields the desired linear ordering of text in finished derivations.

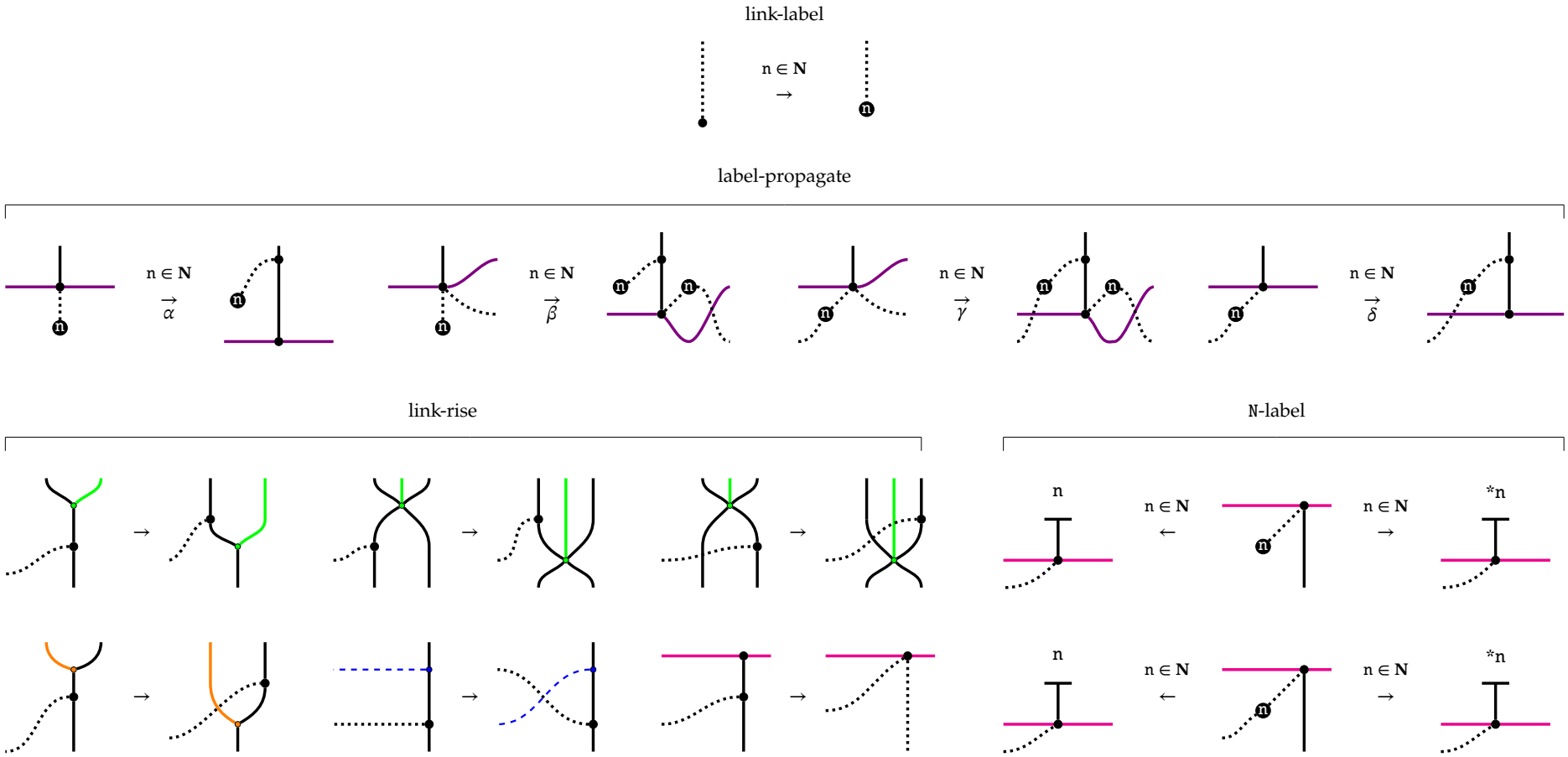


Rules 1.2.6 (Coreferential structure and noun labels)**.**



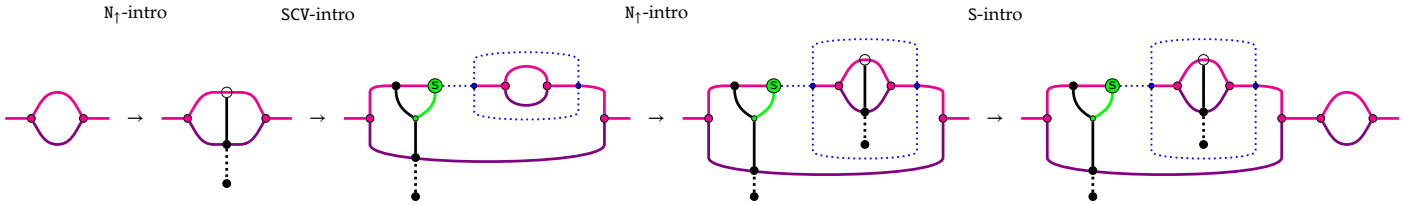
The linked-N-intro rules introduce a new unsaturated noun in the next sentence that coreferences the noun in the previous sentence that generated it, with variants for each pair of sentences involved. We depict three: simple to simple, between CNJ-related sentences, and from either a CNJ or SCV to a simple sentence.
N-shift rules allow any unsaturated noun to move into the next sentence, again with variants for different pairs of sentences. 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.

Rules 1.2.7 (Labelling nouns)**.** 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:

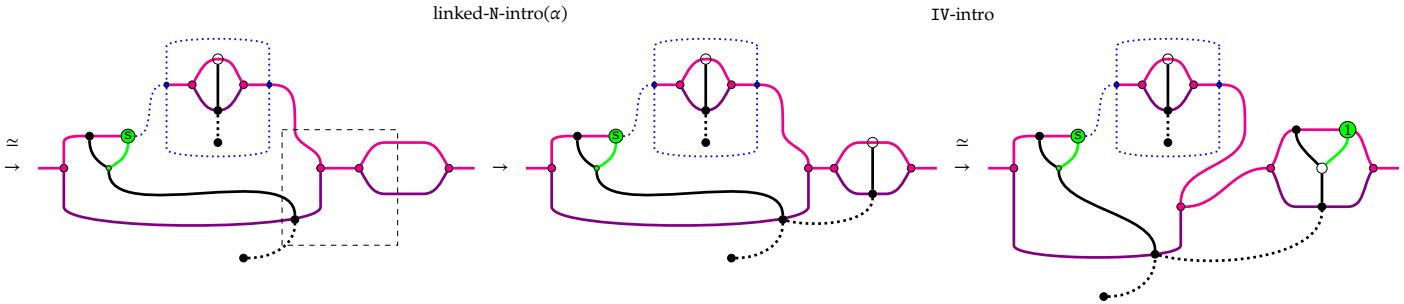


The $n \in \mathbf{N}$ notation indicates a family of rewrites (and generators) for each noun in the lexicon. Link-label assigns a noun to a diagrammatically linked collection of coreferent nouns, and link-propagation is a case analysis that copies a link label and distributes is across coreferent nouns. 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

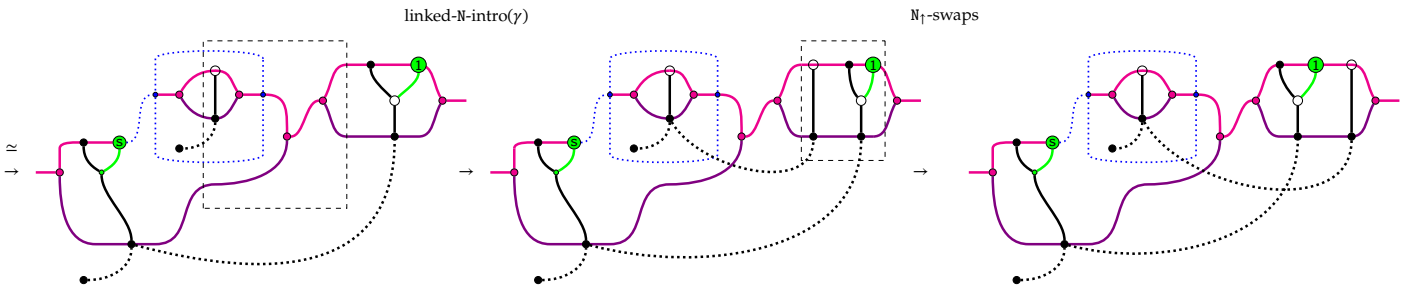
Example 1.2.8 (`sober Alice sees Bob clumsily dance. She laughs at him.`). We start the derivation by setting up the sentence structure using `S`- and `SCV`-intro rules, and two instances of `N`-intro, one for Alice, and one for Bob. Observe how the `N`-intro for Bob occurs within the subsentence scoped over by the `SCV`-rule.



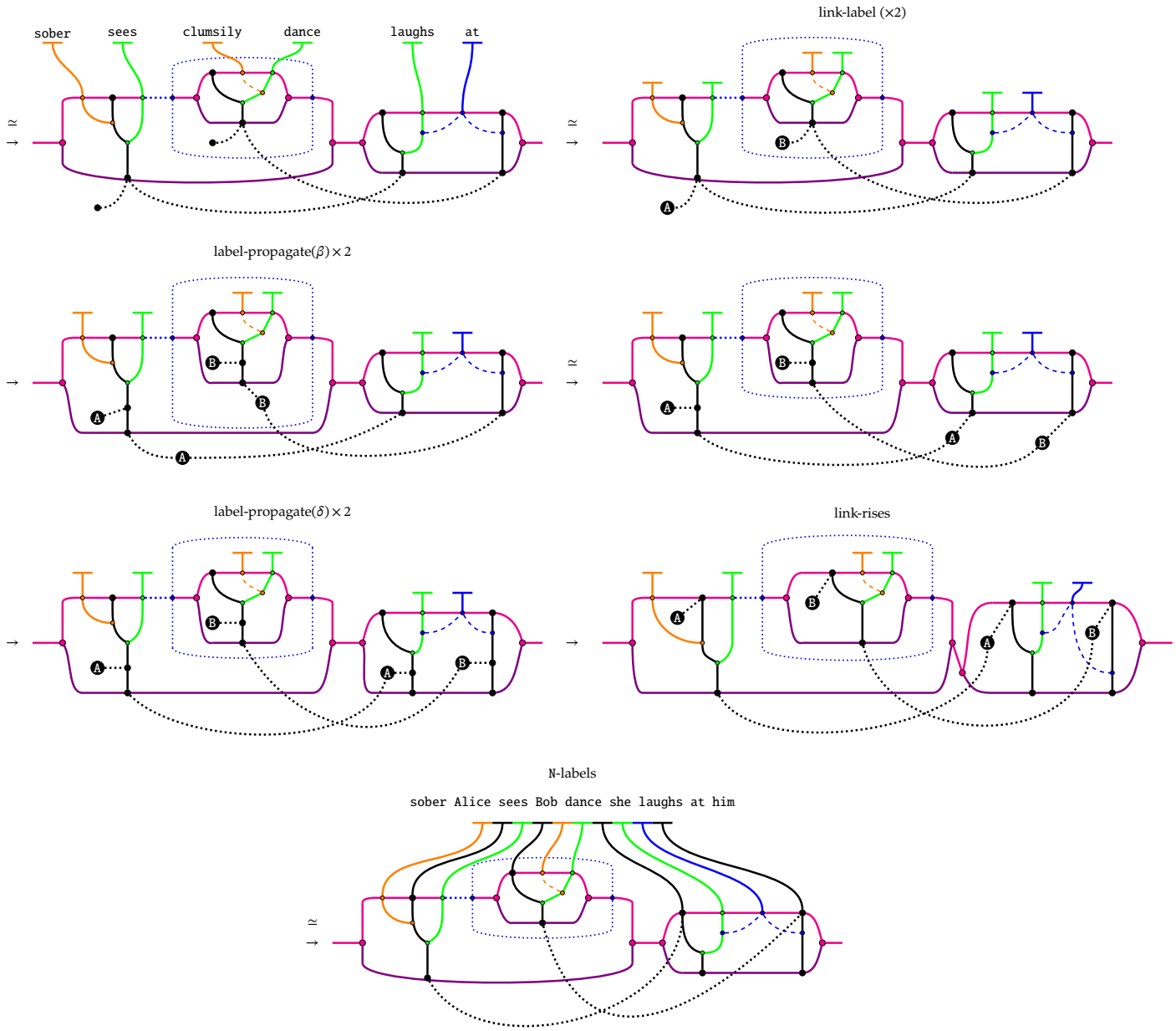
By homotopy, we can rearrange the previous diagram to obtain the source of the linked-`N`-intro rewrite in the dashed-box visual aid. Observe how we drag in the root of what is to be Alice's wire. Then we use the `IV`-intro in the second sentence, which sets up the surface structure `she laughs`, and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



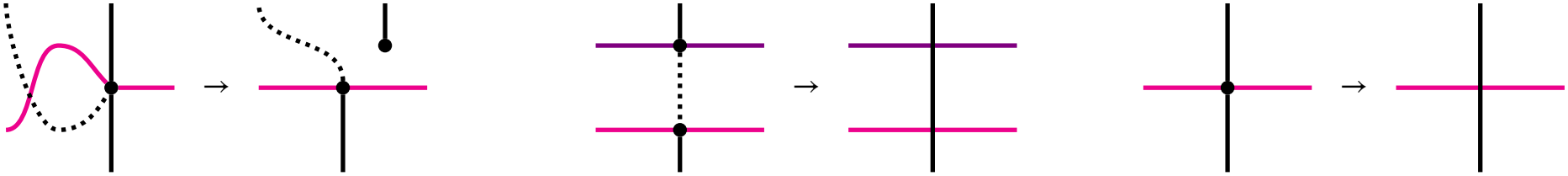
By homotopy again, we can do the same for Bob, this time setting up for the γ variant of linked-`N`-intro which handles the case when the spawning noun is within the scope of an SCV. Then by applying a series of `N`$_\uparrow$-swaps, the unsaturated noun is placed to the right of the intransitive verb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



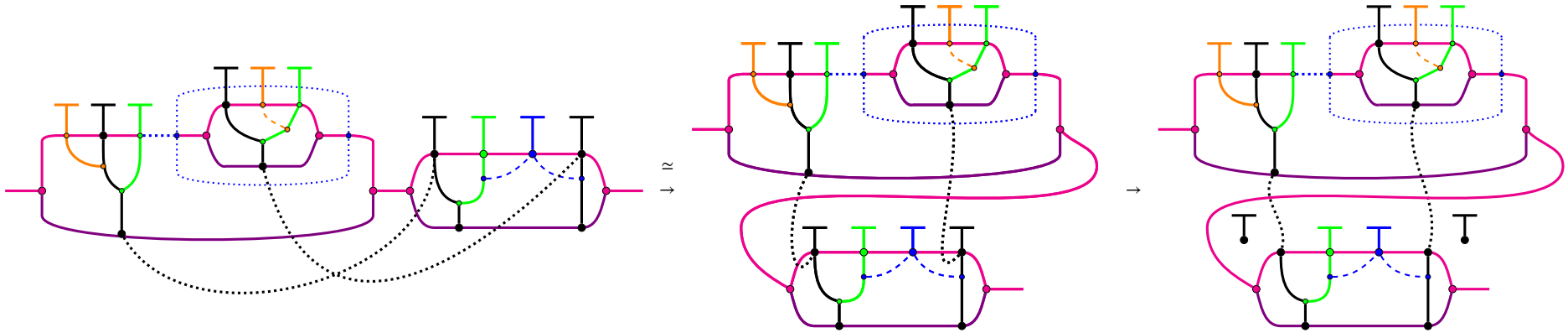
We've already done the surface derivation for the two sentences separately in Examples 1.2.4 and 1.2.5; since neither of those derivations touch the roots of noun-wires, we can emulate those derivations and skip ahead.


sober
sees
clumsily
dance
laughs
at
link-label (×2)
label-propagate(β) × 2
label-propagate(δ) × 2
link-rises
N-labels
sober Alice sees Bob dance she laughs at him

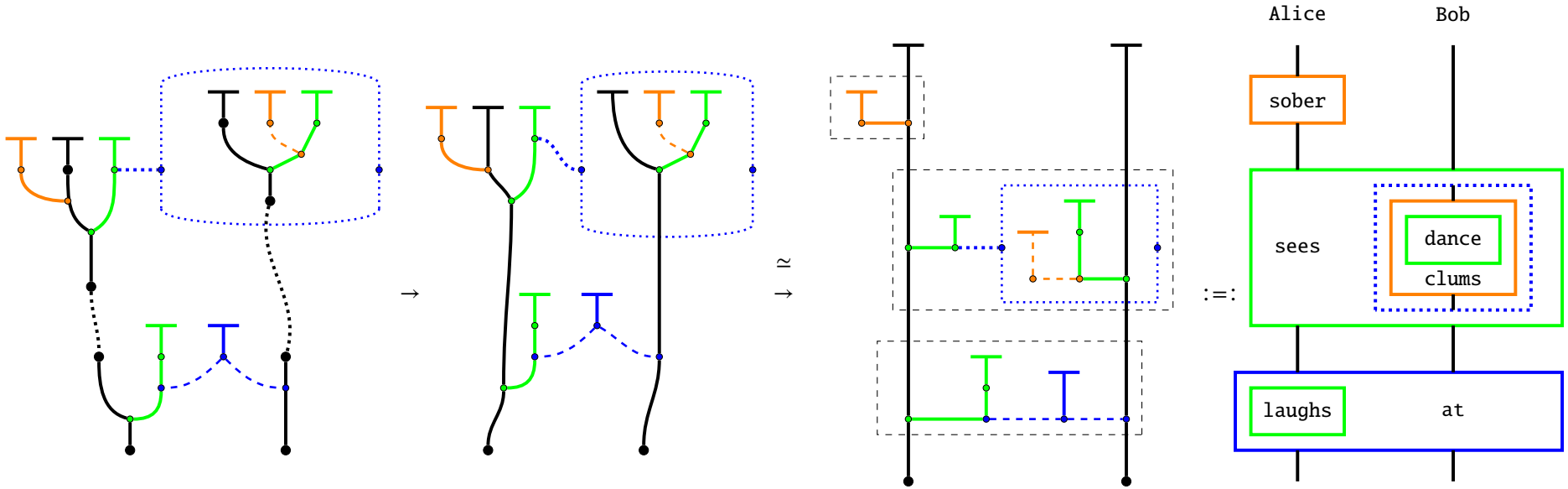
Rules 1.2.9 (Text to circuit)**.** We turn finished text diagrams into text circuits by operating *in situ*, with extra rules outside the grammatical system that handle connecting noun wires.



Example 1.2.10 (Text to circuit in action)**.** In the first step below, by Lemmas 1.2.15 and 1.2.16, 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 1.2.9 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 to. In the fourth step, we apply the second and third rewrites of Construction 1.2.9 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*.



1.2.2 Text circuit theorem

NOW WE WOULD LIKE TO FORGET ABOUT THE MESSY DETAILS OF THE CIRCUIT-GROWING GRAMMAR and treat the text circuits themselves as a generative grammar for text, where the primitive operations are string-diagrammatic composition and nesting within boxes: we aim to prove that all text circuits that one might draw correspond to grammatically acceptable text. Moreover, this correspondence has to hold in such a way that connectively equivalent ways to present text circuits correspond to texts that "mean the same thing", e.g. up to rearrangement or grouping of sentences respecting constraints on pronominal reference.

OUR STRATEGY HAS TWO PHASES. Since we have already demonstrated that the circuit-growing grammar yields text circuits it will suffice to demonstrate grammatical acceptability for the circuit-growing grammar, and separately exhibit a well-behaved translation from arbitrary text circuits to circuit-growing grammars. Altogether this achieves our desired correspondence between text circuits and finished circuit-growing derivations, and text circuits will inherit the grammatical acceptability properties we demonstrate of circuit-growing grammars.

WE PROCEED IN STEPS. First, we relate the circuit-growing rules for simple and complex sentences to pedestrian CSGs, which establishes grammatical sensibility at the sentential level. Second, we analyse the pronoun connection rules of the circuit-growing grammar to establish that the text produced is sensible. Third, we expand the rules for our circuit-growing grammar so that all of the diagrammatic idiosyncrasies of text circuits have something to correspond to in the circuit-growing grammar. Finally, we demonstrate how to convert text circuits into finished circuit-growing derivations.

Construction 1.2.11 (CSG for simple sentences)**.** We may characterise simple sentences (containing only one verb) with the context-sensitive grammar in Figures 1.26 and 1.27.

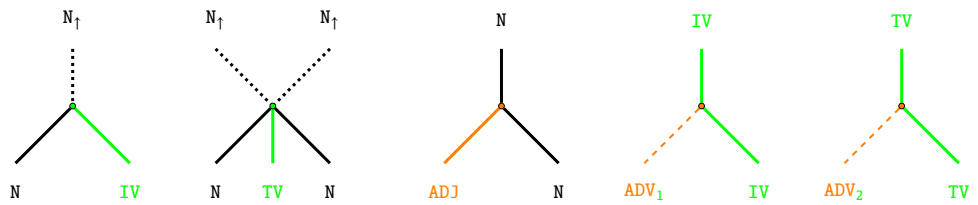


Figure 1.26: Reading each diagram from top-to-bottom, from left-to-right we have generators for intransitive verbs, transitive verbs, adjectives, and adverbs. Generators for verbs require a number of $\mathtt{N}_\uparrow$ matching their arity as input, hence a CSG.

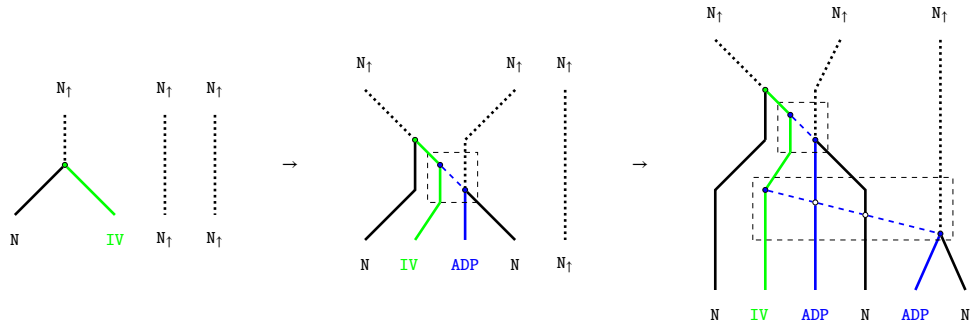


Figure 1.27: Adpositions require several helper-generators, which are the components within dashed boxes in the depicted example demonstrating the process of appending adpositions to an intransitive verb.

Proposition 1.2.12. Up to labels, the simple-sentence rules of the circuit-growing grammar are strongly equivalent to the CSG; in particular, they yield the same sentences.

Proof. By graphical correspondence between CSG rules and how the generators of the circuit-growing grammar changes surface nodes (Figure 1.28). □

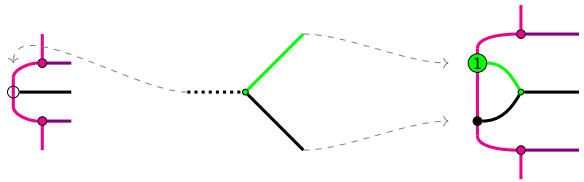


Figure 1.28: Viewing nodes on the pink surface of circuit-growing grammar as 1-cells, each rewrite rule yields a 2-cell; e.g. the dashed-blue helper lines for adpositions correspond to the ADP-pass rules in circuit-growing grammar. The correspondence between the IV-intro rules of both grammars is depicted.

Proposition 1.2.13. Up to labels, Rules 1.2.2 and 1.2.3 for simple and complex sentences yield the same sentences as the combined CSG of Construction 1.2.11 with the additional rules depicted in Figure 1.29.

Proof. Same correspondence as Proposition 1.2.12, ignoring the dotted-blue guards. □

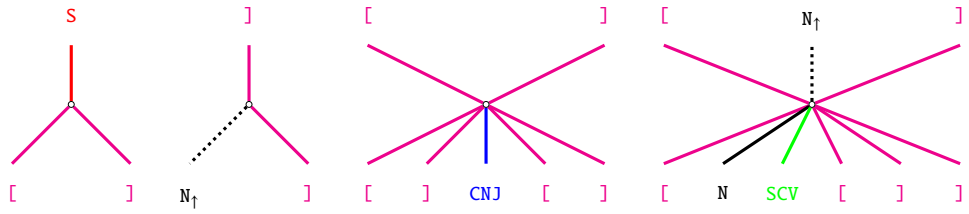


Figure 1.29: The first rule instantiates the left and right boundaries of a sentence, corresponding the starting bubble in circuit-growing grammar. The second corresponds to $\mathtt{N}_\uparrow$-intro, the third CNJ-intro, and the fourth SCV-intro.

Now we address complex sentences and text by connecting nouns (Figure 1.30). This presents no issue across distinct simple sentences, but a complication arises when connecting nouns within the same simple sentence with reflexive pronouns e.g. `Alice likes herself`. 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 we several options exist, explored in Figure 1.31.

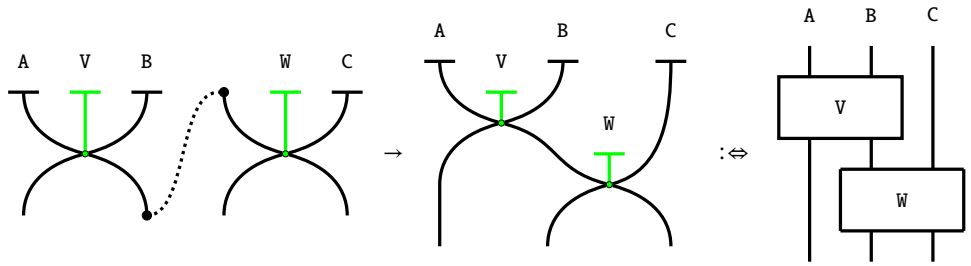
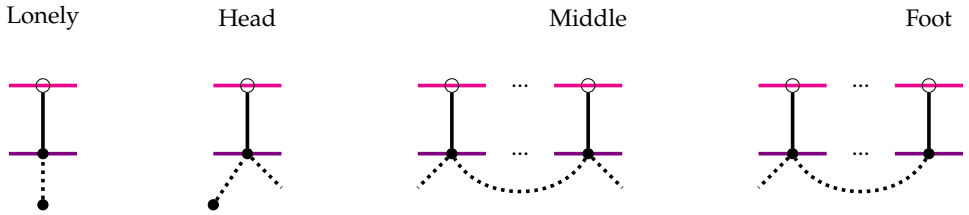


Figure 1.30: We choose the convention of connecting from left-to-right and from bottom-to-top, so that we might read circuits as we would English text: the components corresponding to words will be arranged in the reverse order, left-to-right and top-to-bottom.

Terminology 1.2.14 (Kinds of nouns with respect to coreference)**.**



We classify kinds of nouns 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.

Lemma 1.2.15. The unsaturated nouns in Terminology 1.2.14 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 1.2.16. 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. □

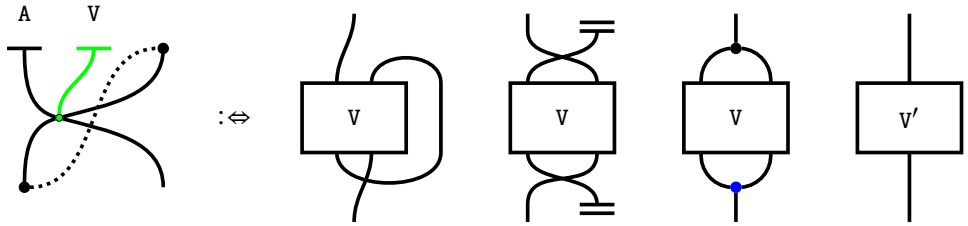


Figure 1.31: From left to right in roughly decreasing stringency, compact closed categories are the most direct solution for reflexive pronouns. Traced symmetric monoidal categories also suffice. So long as the noun wire possesses a monoid and comonoid, a convolution works. We also can just specify a new gate. We provide a purely syntactic treatment in (**?**); for now we treat them as if they were just verbs of lower arity.

Definition 1.2.17 (Finished text diagram)**.** The circuit-growing grammar produces *text diagrams*. We call a text diagram *finished* if all surface nodes are labelled.

Proposition 1.2.18. 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 1.2.12 and 1.2.13. When multiple sentences occur within the scope of a `SCV` we might prefer the use of contentless complementisers and conjunctions, e.g. `Alice sees that Bob draws and Charlie drinks , and Dennis dances .` is grammatically preferable but meaningfully equivalent to `Alice sees (Bob draws Charlie drinks) 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. □

Proposition 1.2.19 (Finished text diagrams yield unique text circuits (up to processive isotopies))**.** *Proof.* Every sentence corresponds to a gate up to notation, and we have a handle on sentences via Propositions 1.2.12 and 1.2.13. Lemmas 1.2.15 and 1.2.16 guarantee processivity. Uniqueness-up-to-processive-isotopy is inherited: text diagrams themselves are already specified up to connectivity, which entails equivalence up to processive isotopy. Therefore, for any circuit C obtained from a text diagram T by Construction 1.2.9, 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' may be obtained in this way. □

The converse of Proposition 1.2.19 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.

Convention 1.2.21 (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 1.32).

Convention 1.2.22 (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 1.2.23 (Sliding)**.** Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel (Figure 1.33).

Remark 1.2.20. There are some oddities about our conventions that will make sense later when we consider semantics. For example, Convention 1.2.21 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 **??**. Similarly, Convention 1.2.26 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 and the distinction in Convention 1.2.22 between typed and "untyped" higher-order processes will be given a suitable semantic interpretation in Section **??**.

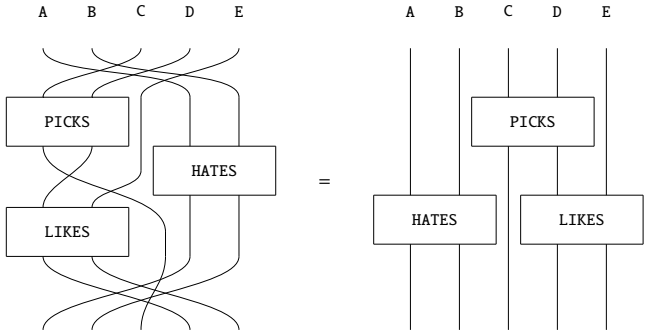


Figure 1.32: Only connectivity matters in text circuits, which we may use to freely rearrange and simplify presentations.

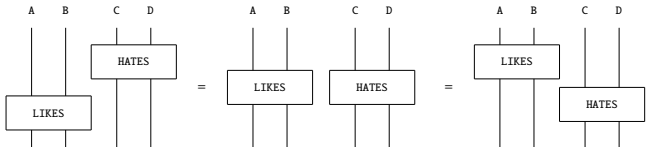


Figure 1.33: Postscript: While sequential composition in process theories often has implicit temporality, this is not necessarily the case for text circuits, which may just (for instance) represent relational constraints. Temporality may be achieved in text circuits by interpreting them in premonoidal settings [CITE], at the cost of the interchange rule depicted here.

Convention 1.2.24 (Reading text circuits)**.** Pronominal connection conventions are to be chosen so that text circuits may be read in the same directional reading conventions of the language they aim to represent.

Convention 1.2.25 (Contentless conjunctions)**.** Conventions 1.2.23 and 1.2.24 require something else to allow them to work at the same time: something to distinguish when the gates are parallel. In terms of text diagrams, we want rewrites that introduces such contentless conjunctions and witnesses their associativity, as in Figure 1.34:

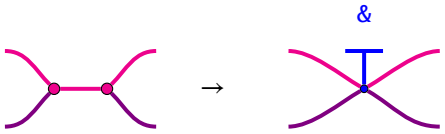


Figure 1.34: Parallel gates represent compound sentences with contentless conjunctions. In English, some examples might be a punctuation mark such as a comma, or phrases such as `and also`.

Convention 1.2.26 (Lonely wires)**.** There's only a single kind of text circuit we can draw that does not obviously correspond to a text diagram, and that's one where gates are missing (left). In process theories, wires are identity processes that do nothing to their inputs. So to deal with lonely wires in terms of text diagrams, we want a rewrite that introduces such contentless verbs (Figure 1.35).

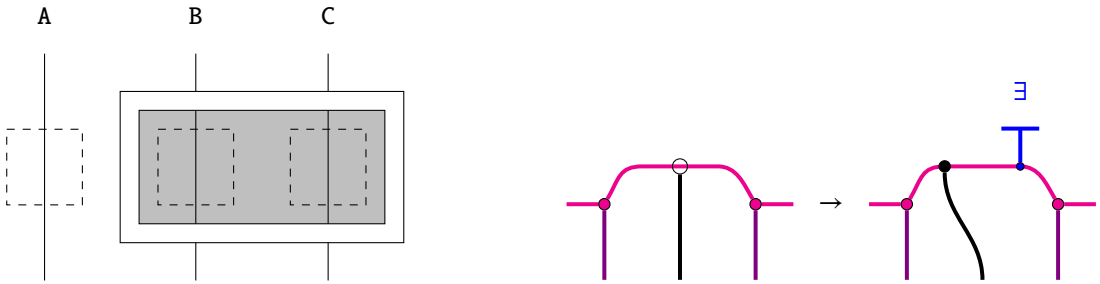
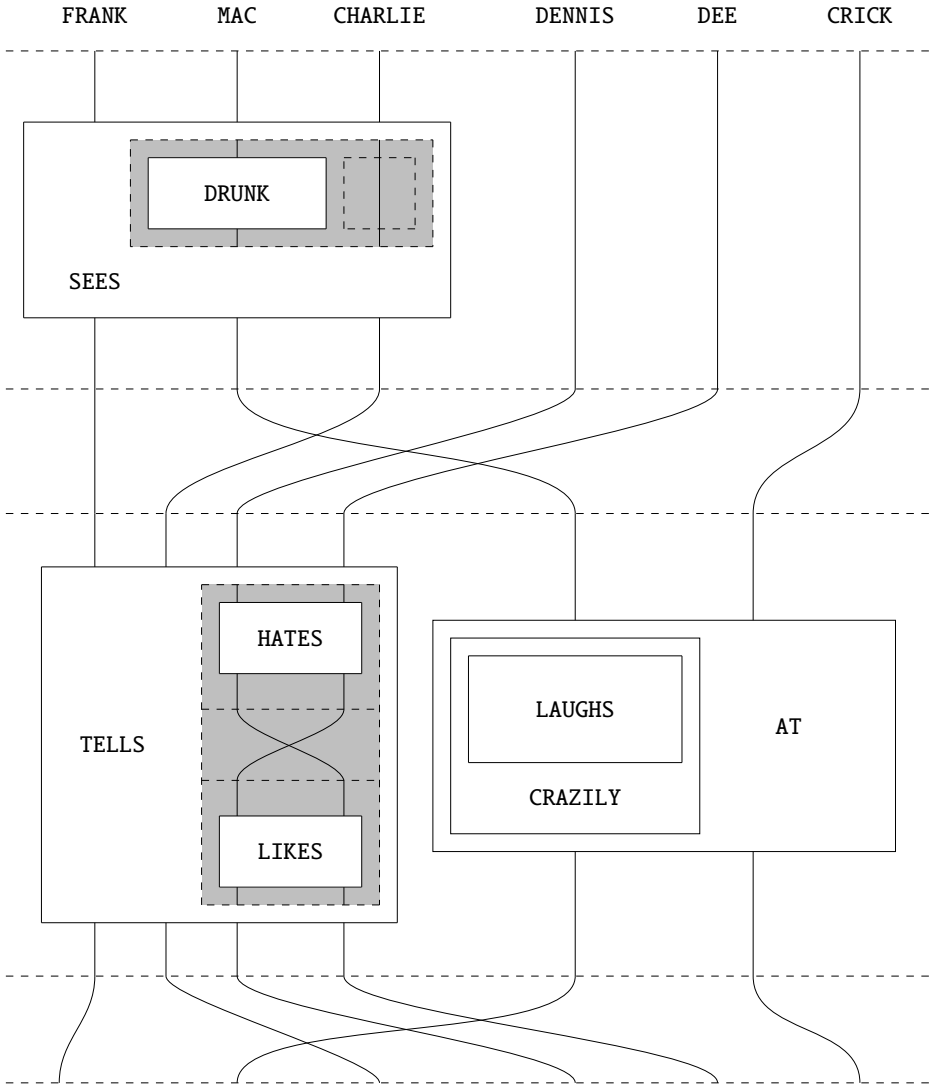
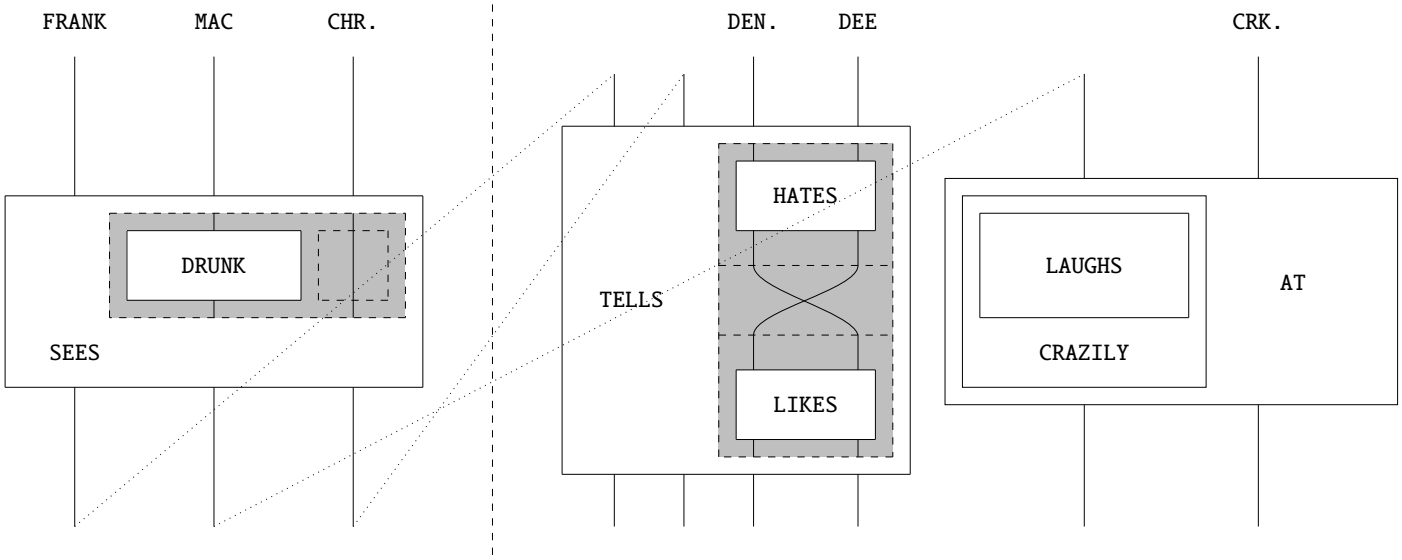


Figure 1.35: Lonely wires in text circuits are identity processes. We require a text diagram analogue, and an intransitive "null-verb" in English that seems to work is `is`, in the sense of `exists`.

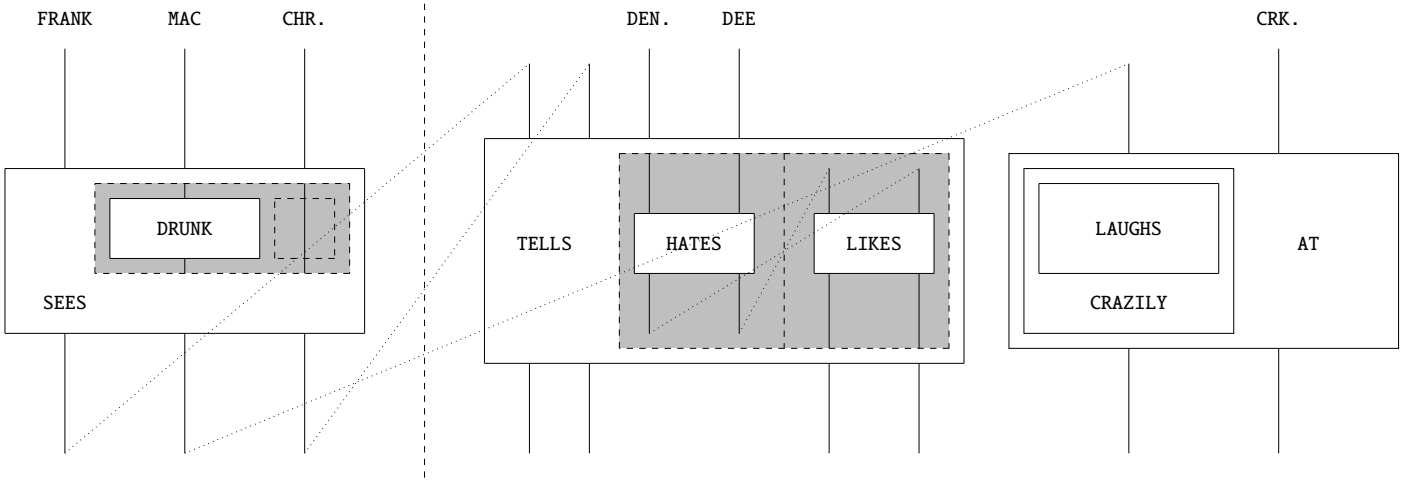
Construction 1.2.27 (Circuit to text)**.** In the presence of additional rewrites from Conventions 1.2.25 and 1.2.26, every text circuit is obtainable from some text diagram, up to Conventions 1.2.21 and 1.2.23. Starting with a circuit, we may use Convention 1.2.21 to arrange the circuit into alternating slices of twisting wires and (possibly tensored) circuits, and this arrangement recurses within boxes. Slices with multiple tensored gates will be treated using Convention 1.2.25. By convention 1.2.26, 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 1.2.22.



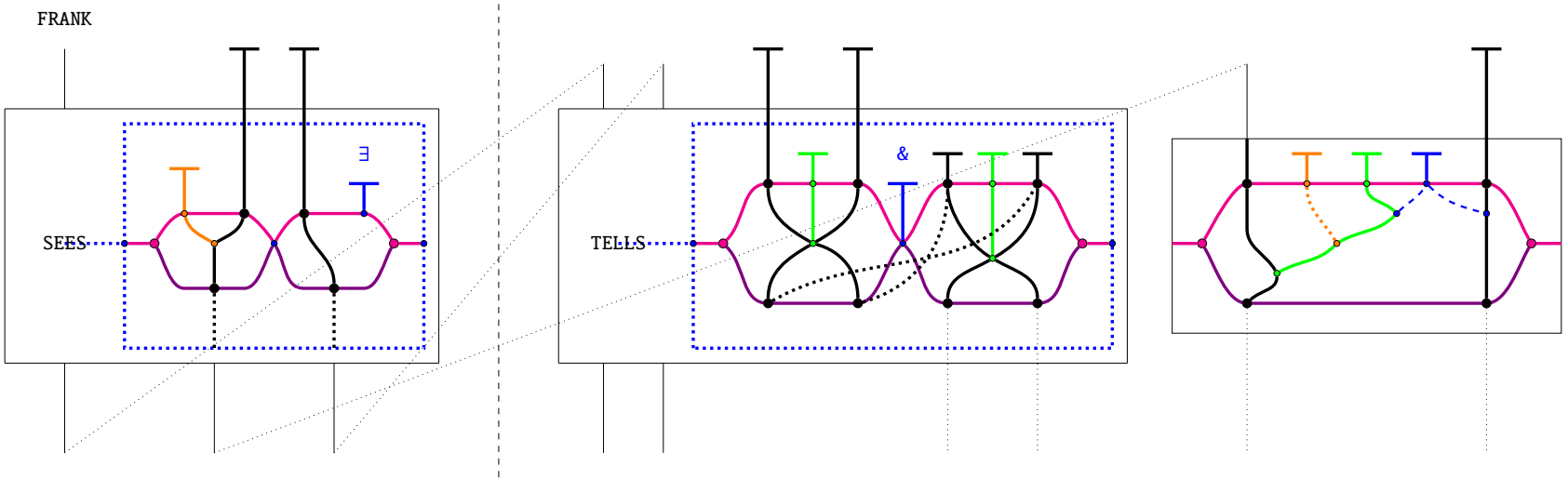
We then linearise the slices, representing top-to-bottom composition as left-to-right. Twist layers are eliminated, replaced instead by dotted connections indicating processive connectivity. The dashed vertical line distinguishes slices. This step of the procedure always behaves well, guaranteed by Proposition 1.2.15. Noun wires that do not participate in earlier slices can be shifted right until the slice they are introduced.



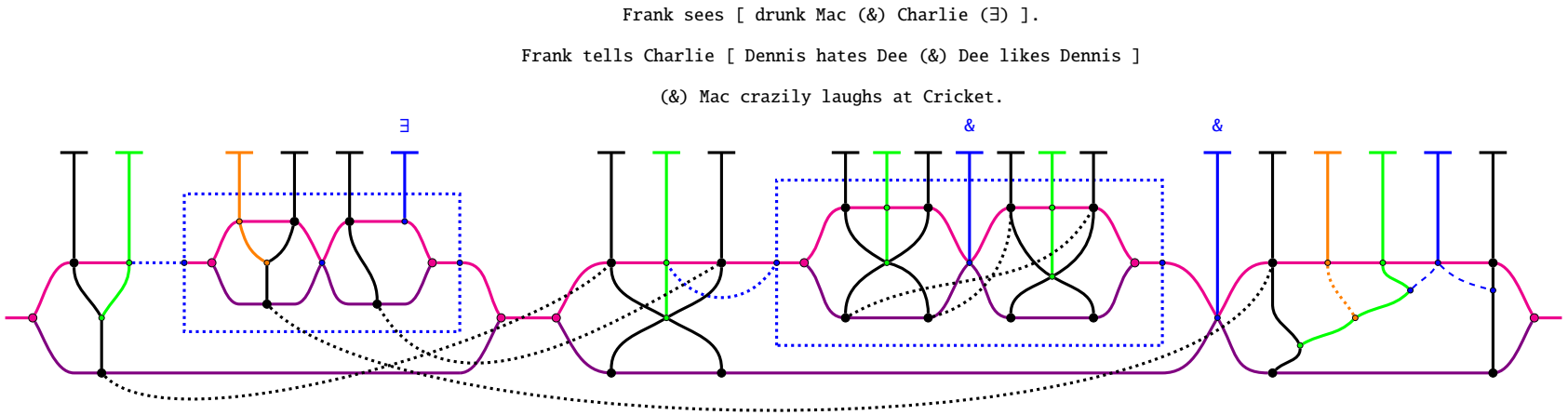
We recurse the linearisation procedure within boxes until there are no more sequentially composed gates. The linearisation procedure evidently terminates for finite text circuits. At this point, we have abstracted away connectivity data, and we are left with individual gates.



By Proposition 1.2.13, gates are equivalent to sentences up to notation, so we swap notations *in situ*. Conventions 1.2.25 and 1.2.26 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.



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 1.2.25 and 1.2.26: applying the reverse of those rewrites and the reverse of text-diagram rewrites yields a valid text-diagram derivation, by Propositions 1.2.13 and 1.2.15. 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.



Theorem 1.2.28 (Text Circuit Theorem)**.** Text generated by the circuit-growing grammar is sensible and surjects onto text circuits. Hence:

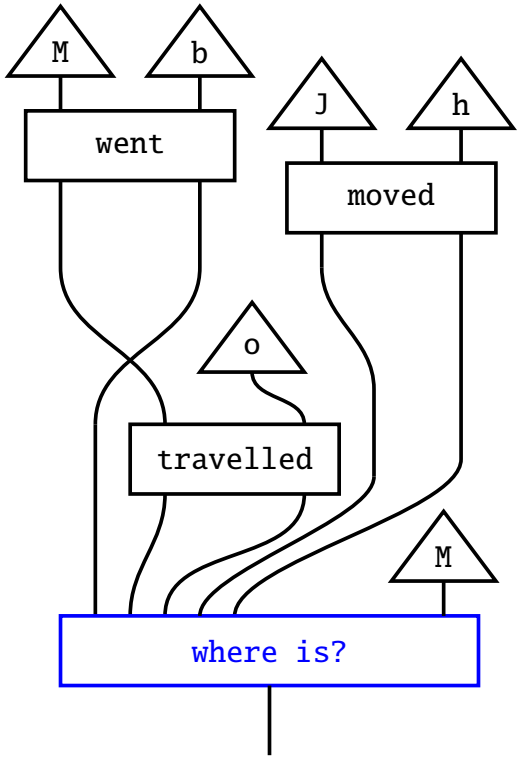
Text circuits are a generative grammar for text

Proof. Sensibility at the sentential level is established by Propositions 1.2.12 and 1.2.13. Proposition 1.2.19 establishes a map from text to circuits, and Construction 1.2.27 witnesses its surjectivity. The conventions required for the construction are accompanied by justifications of sensibility. □

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 in the form of a process-theoretic equation to be satisfied: the depicted composite process ought to be equal to the `office` state:



1.3 *Text circuits: details and development*

This section covers some practical developments, references for technical details of text circuits, and sketches of how to play with them. The most striking demonstration to date is that circuits are defined over a large enough fragment of language to *leverage* several bAbi tasks (**?**), 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, detailed in the margin. 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 (**?**). 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. Second, one implements the text circuits on a computer. On quantum computers, boxes are modelled as quantum combs. 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; an example of this interpretation of families of processes is the use of an aggregation monoid in graph neural network (**?**). 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 (**?**) 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 computing with text, where facets are open to interpretation and modification.

1.4 How to play

Text diagrams of the sort we have been growing with the circuit-growing grammar have rich structure for modelling syntactic structure. In tandem with an implementation in neural nets, there are many things that can be achieved. Here I want to share some possibilities and general principles.

FUNCTIONAL WORDS. No separate "information structure" is required; all is transparent and manipulable. For instance, in the case of relative pronouns in Example 1.4.2, ansatz wires may be freely introduced and manipulated to achieve the desired correspondence between deep and surface structure.

GRAMMAR EQUATIONS. A principle for the 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, for instance, that the use of passive voice or copulas amounts to swaps in the linear surface order, as in Examples 1.4.3 and 1.4.4. Syntactic rules may even introduce semantic components, as in for instance the treatment of the possessive pronoun in Example 1.4.5. In sum, intuitions about the systematicity of language are directly translatable into rewrite rules that manipulate deep structure as one sees fit.

GRAMMATICAL TYPING IS 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, as demonstrated in Examples 1.4.6 and 1.4.7.

SYNCATEGOREMATICITY. 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. See Examples 1.4.8 and 1.4.9 for a demonstration.

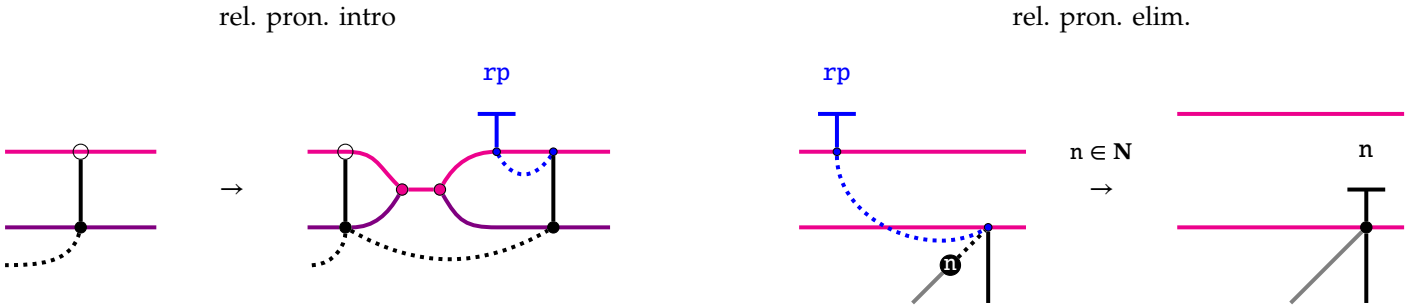
Syntax in context. Extending the reach of text circuits to determiners, quantifiers, and conditionals appears to require a contextual process theory in which to evaluate and enforce constraints upon the purely syntactic content of text circuits. The broad strategy sketched here rests upon three tactics. 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, the latter of which may be generic tests for adjectives as static predicates or verbs as dynamic predicates. The pairing of gates with measurements follows the philosophy of update structures (**?**). The 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. 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 may be analysed in particular finitary contexts as conditionals and constraints on truth-conditional measurements. Existential quantifiers evaluated in the finitary case yield alternative circuits. See Examples 1.4.11, 1.4.12, 1.4.13, 1.4.15, and 1.4.15 for demonstrations.

Text is composition. Apart from nesting, the other form of composition available for text circuits is the connectivity of wires. We have presented a simplified theory of discourse where the only discourse referents are nouns, but this is not an inherent limitation of text diagrams, where grammatical data of all kinds are freely presentable and composable. In this presentation, I have stuck to string-diagrams in a compact-closed setting and their rewrites, and I have avoided the affordance of weak n-categories to specify *manifold* diagrams and their rewrites, where in addition to 1-D strings connecting 0-D boxes, there may be 2-D planes connecting 1-D strings, 3-D volumes connecting planes, string diagrams restricted to surfaces or volumes... All this is to say that if you can draw it, language can have whatever geometry you want. It just so happens that symmetric monoidal categories are the royal road for pedagogy and practicality (who knows how to interpret a manifold diagram as a computational process?), so it is best to stick to strings.

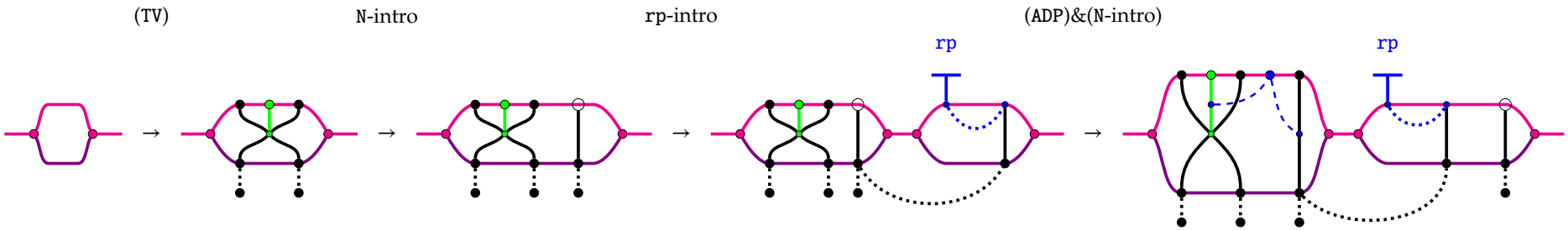
Objection!: Hold on, isn't this just transformational grammar? Haven't we moved on from that? In spirit, yes. The two major mathematical distinctions here are well-typing and many-input-many-output instead of treelike. The practical distinction is that this theory works with neural nets. Both approaches have the same theoretical problems: over- and undergeneration, no evidentiary basis for psychological realism, too rigid for functionalists, and so on. But recall that we differ in aims: our formalist approach is ultimately in service of approximating human language structure in machines for interpretability. How so? Solving language tasks such as bAbi via text circuits 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. What is meant by "conceptually-compliant" is a stronger variant of Firth's maxim: "the meaning of a word is how it interacts with other words". How do we justify that claim? The initial conception of

bAbi was that the ability to answer questions about – for instance, the verb `to go` – in many different contexts amounted to having a consistent internal "world-model". But question-answering performance by itself is evidently insufficient for the degree of interpretability implied by conceptual-compliance, because the internal model is not forthcoming in transformer solutions. On the other hand, we *do* obtain the building blocks of compositional world-models by learning word gates in text circuits: each learnt word gate may be considered a well-grounded semantic primitive in the construction of novel text circuits, and the resulting circuits are modifiable world-models that are queriable using the (also learnt) measurement-gates. Why is that so? Because just as in Section **??** we do not need to know how an update is implemented if it satisfies characteristic operational constraints imposed by process-theoretic equations, we don't need to know what's going on inside the gate `to go` so long as it satisfies the process-theoretic equations that `to go` ought to satisfy. What are these equations? Firth says that it is how `to go` behaves with respect to all other words in all contexts, which we approximate by translating individual bAbi tasks involving the word `to go`, via text circuits, into a representative sample of the process-theoretic equations that `to go` ought to satisfy. So the philosophical strength of the claim that `to go` and synonyms have been learnt-from-data in a way that coheres with human conceptions rests on three points: performance, Firth (or if you like, the Yoneda Lemma), and the breadth and variety exhibited in the bAbi dataset. The real test is practical demonstration, for which time will tell.

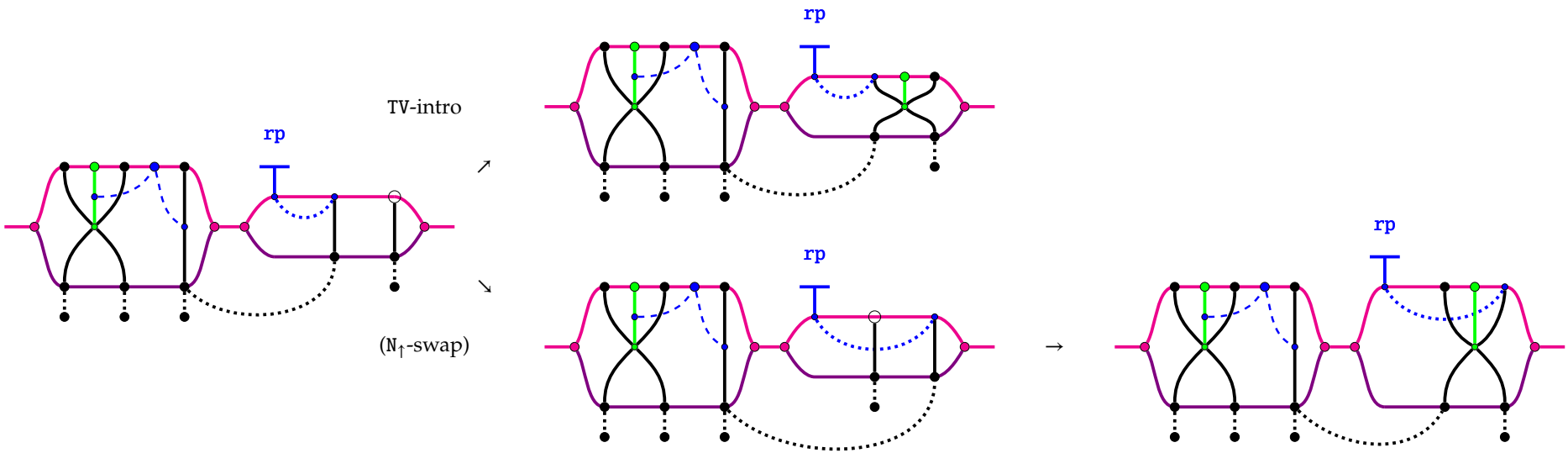
Rules 1.4.1. A relative pronoun glues two sentences across a single noun. For example, what would be a text with two sentences `Alice teaches at school.` `School bores Bob.` can be composed by identifying `school`: `Alice teaches at school that bores Bob.` In this case, `that` is called a subject-relative 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 may be dealt with in a similar fashion as we have with relative pronouns, and various semantic interpretation options have been covered already in Figure 1.31. 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.



Example 1.4.2 (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.



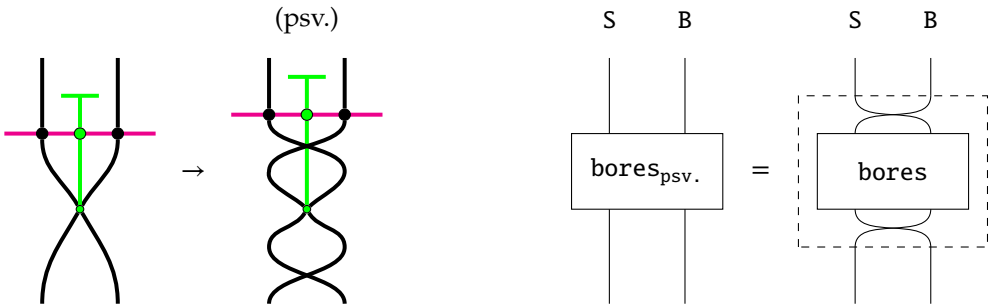
Now have a branching derivation. We may either directly generate a transitive verb treating the relative pronoun as a subject, or we may first perform an $\mathtt{N}_\uparrow$-swap first and then generate a transitive verb, treating the relative pronoun as an object. Now the ends of either branch can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



Example 1.4.3 (Passive voice).

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

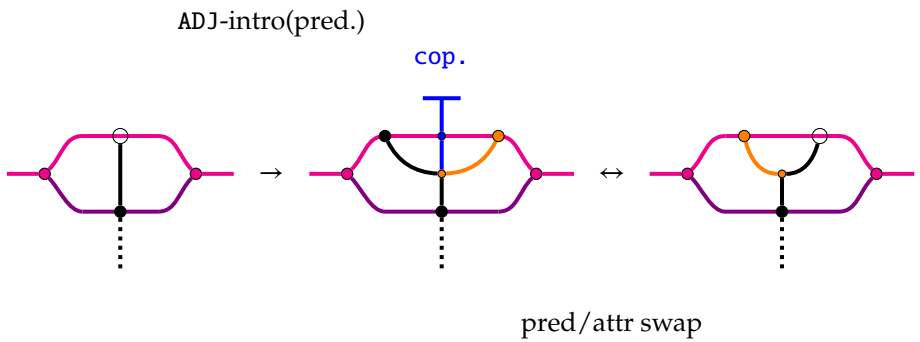
Twists in wires can be used to model passive voice constructions, which amount to swapping the argument order of verbs. In the original (**?**), a more detailed analysis including the flanking words `is bored by` involves introducing a new diagrammatic region, which is modelled by having more than a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



Example 1.4.4 (Copulas).

`Red car` = `Car is red`

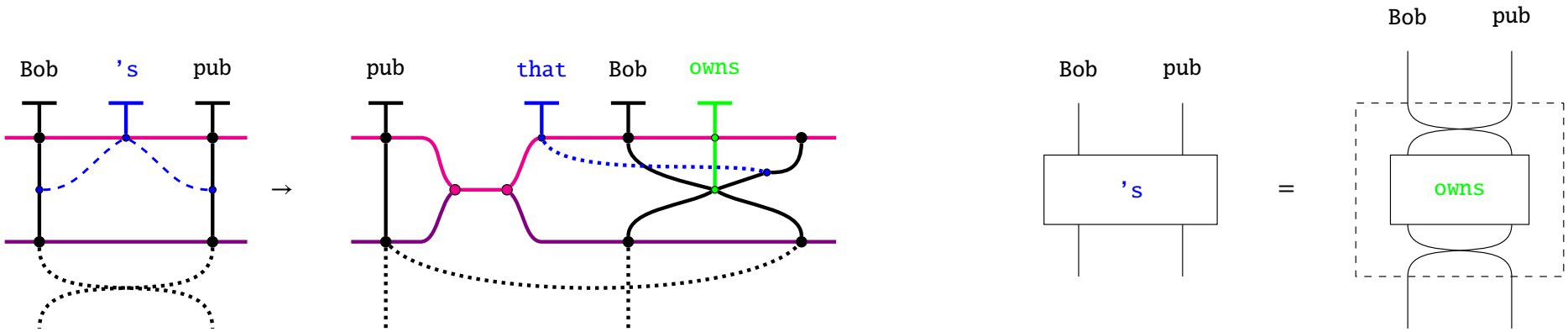
Modifiers such as adjectives and adverbs when they occur before their respective noun or verb are called *attributive*. When modifiers occur after their respective target, they are called *predicative*. In English, without the aid of `and`, only a single predicative modifier is permissible, e.g. `big red car` and `big car is red` are both acceptable, but `car 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.



Example 1.4.5 (Possessive pronouns).

$$\texttt{Bob's pub} = \texttt{Pub that Bob owns}$$

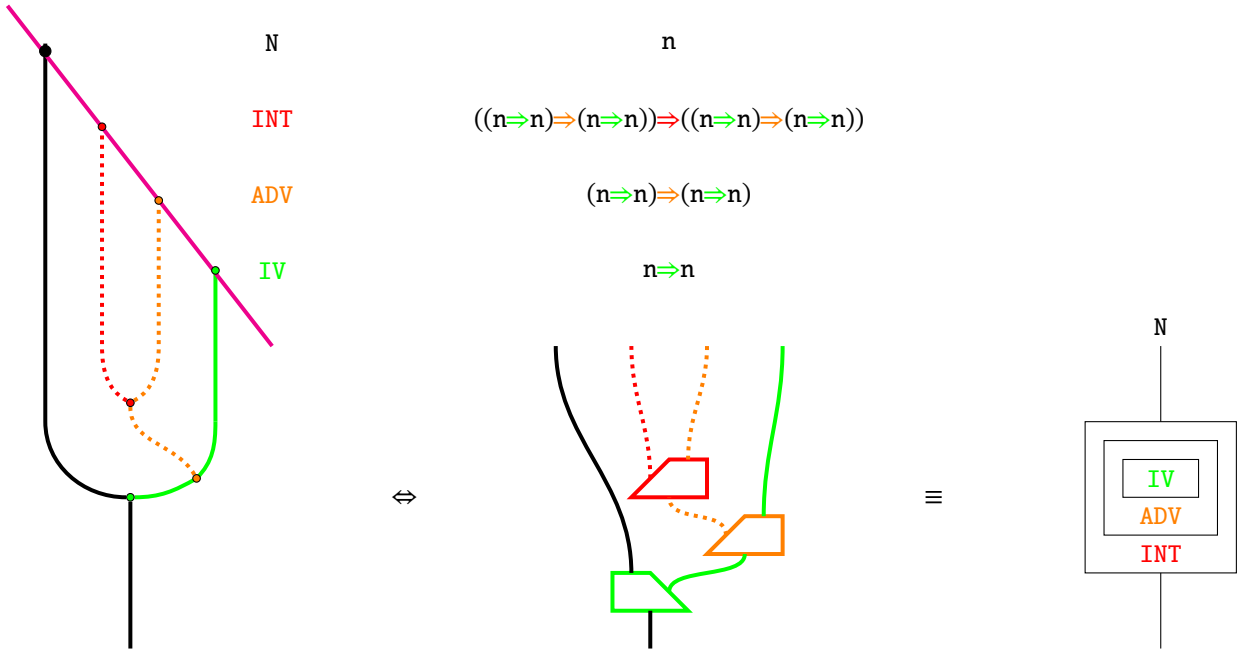
This example, along with other grammar equations, was first introduced in the pregroups and internal wirings context in (**?**). Possessive pronouns are placed contiguously in between noun-phrases, for which the diagrammatic technology we developed for placing adpositions can be repurposed. Possessive 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.



Example 1.4.6 (Intensifers).

```
Alice very quickly runs
```

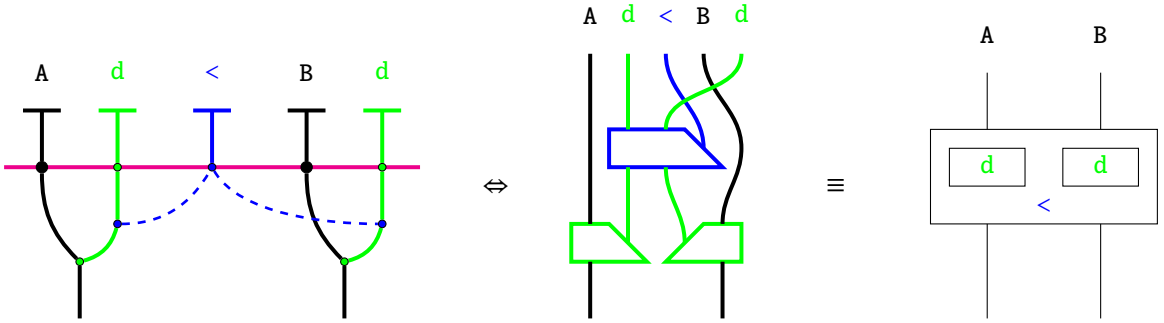
The deep nodes of a text diagram may be equivalently viewed as evaluators in a symmetric monoidal closed setting, and the surface nodes as states for the evaluators. By Curry-Howard-Lambek, this view recovers typelogical grammar settings where composition is some variant of modus ponens. So long as the 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.



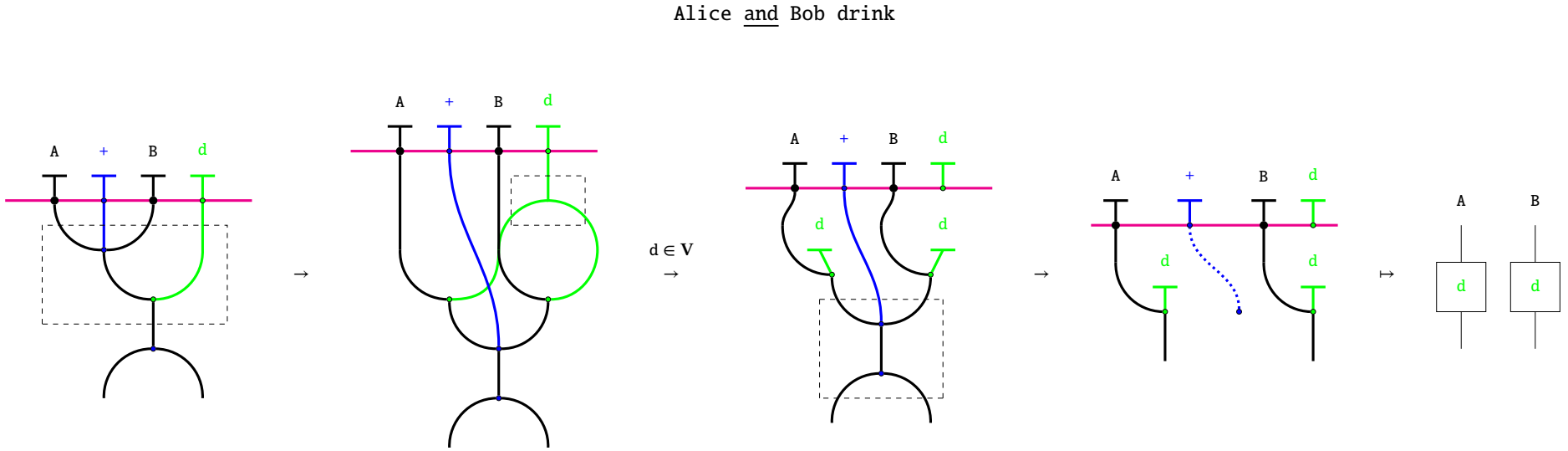
Example 1.4.7 (Comparatives).

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.



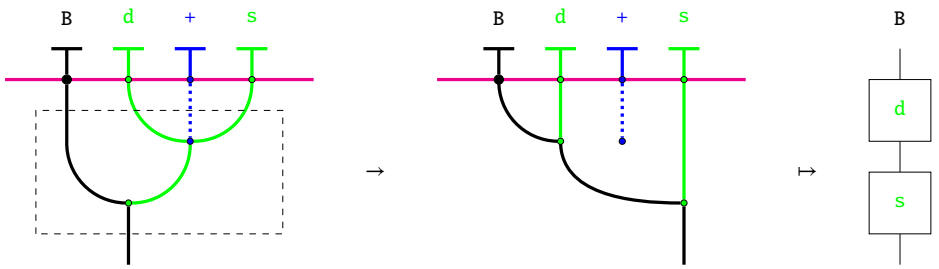
Example 1.4.8 (Syncategorematicity I). *Syncategorematic* words are roughly those that have contextually-dependent semantics. Their dependency is usually predicated on the grammatical type of their arguments. In our terms, since we consider the semantics of text circuits to be underpinned by monoidal functors that reify the circuits in a target category, syncategorematic words such as and may be treated as distributive laws. Here 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 $a \times (b + c) \quad = \quad (a \times b) + (a \times c)$. A new copy-node for verb labels that has rewrites for all verbs facilitates distribution, and the deep and nodes come in a tensor-dentensor pair analogous to those for nonstrict string diagrams. Sources of rewrites are outlined in dashed boxes.



Example 1.4.9 (Syncategorematicity II).

Bob drinks and smokes

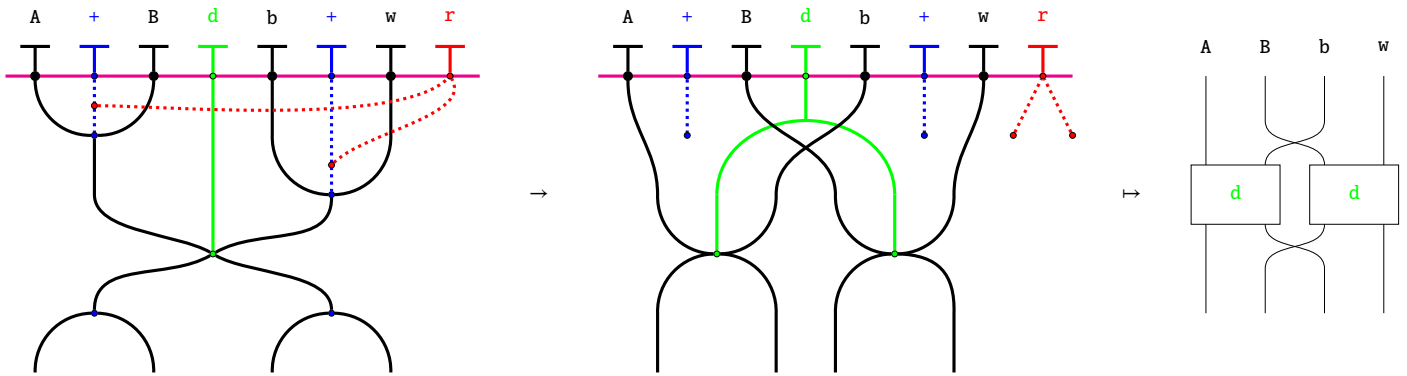
In this example, the same word and is a conjunction of verbs. In this case we choose to interpret the conjunction of verbs as sequential composition, so there is no need for a corresponding detensor for the and of verbs.



Example 1.4.10 (Coordination).

Alice and Bob drink beer and wine respectively

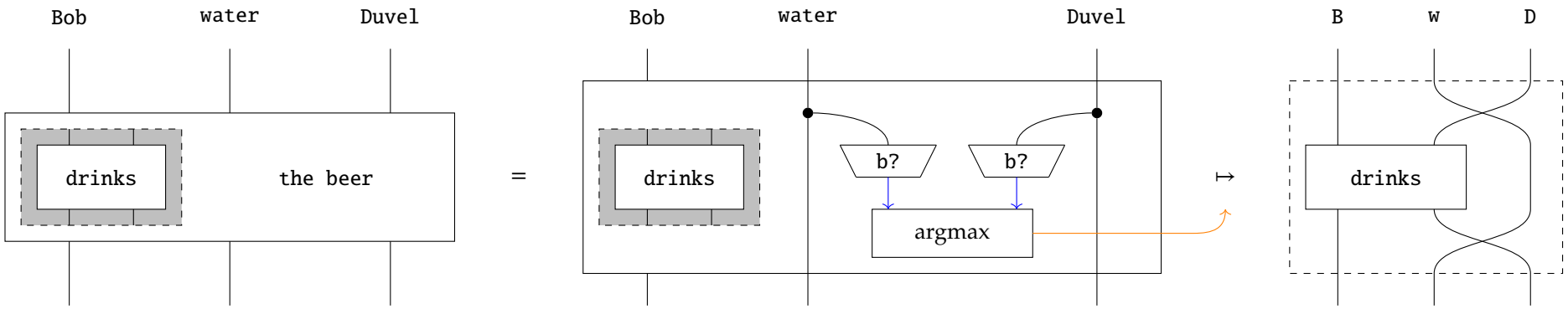
We stand to win in terms of conceptual economy for modelling; more complex phenomena of text structure such as coordination appear to be resolvable in the same framework of distributivity-law rewrites.



Example 1.4.11 (Determiners I).

`Bob drinks the beer` (among drinks)

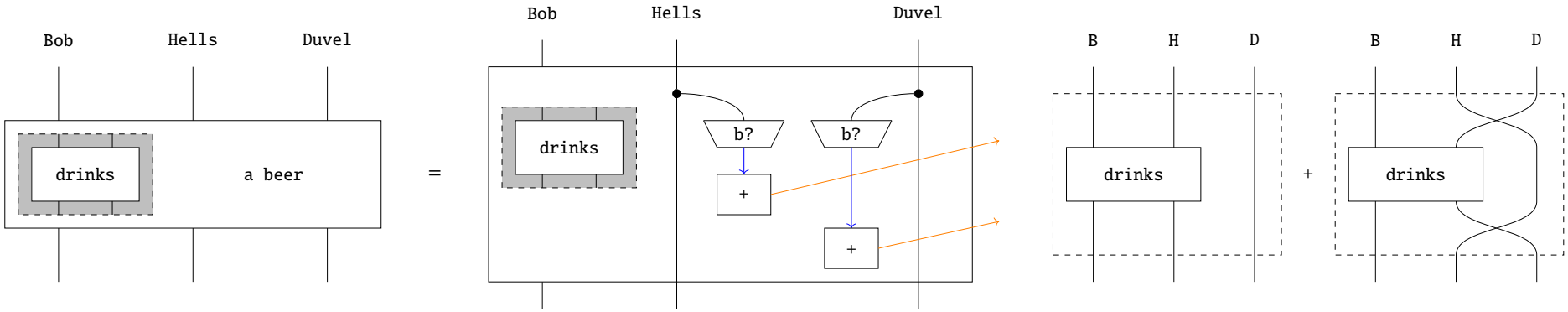
Here, `drinks` is considered transitive and `the beer` a nesting box for `drinks` that reaches over to contextual wires representing a selection of beverages. In this case (relying on the implicit uniqueness of `the`), a series of `beer?` tests may be computed, and the best match chosen as the resulting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



Example 1.4.12 (Determiners II).

`Bob drinks a beer` (among drinks)

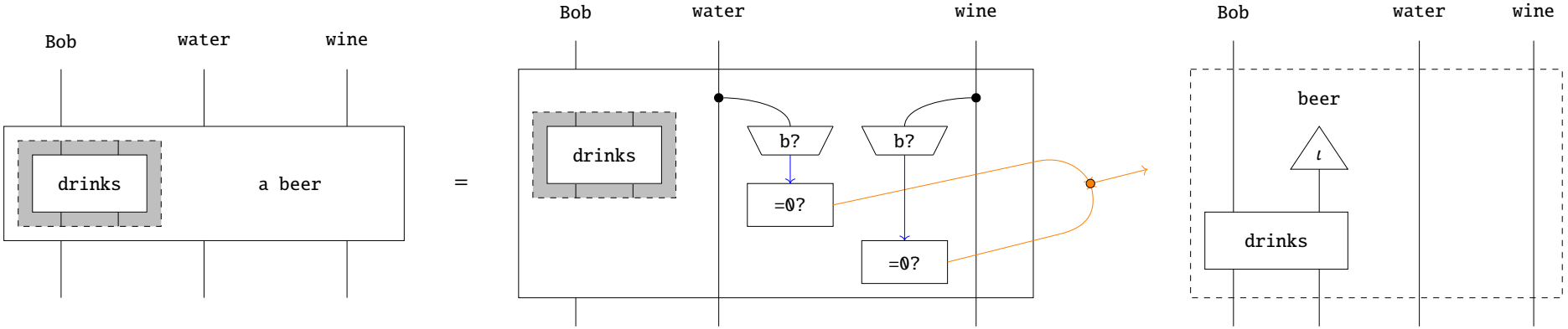
We take the logical (and pragmatic) reading of `a` as $\exists!x \;:\; \texttt{beer?}(x) \wedge \texttt{drinks?}(\texttt{Bob}, x)$. Subject to having a method to hold onto alternatives – in essence an inquisitive semantics approach – we may create alternative circuits for each successful `beer?` test.



Example 1.4.13 (Determiners III).

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

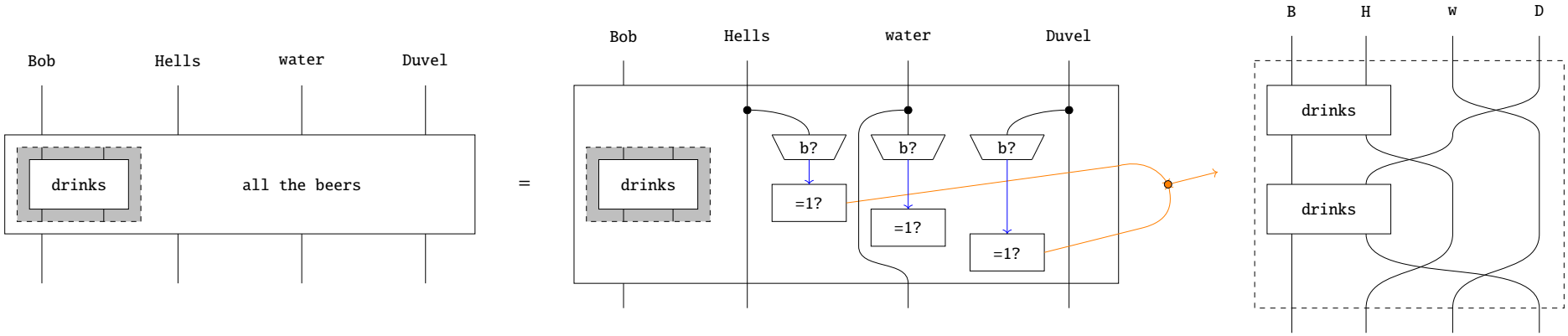
When there are no beers in context, the same statement takes on a dynamic reading: it constitutes the introduction of a beer into discourse. In terms of text circuits, this amounts to introducing a novel beer-state and beer-wire. Determining an appropriate setting to accommodate "arbitrary" vs. "concrete" beers (c.f. Fine's arbitrary objects (**?**)) requires further research and experimentation, but preliminarily it is known that density matrices are capable of modelling semantic entailment (**?**), at the computational cost of adopting the kronecker product. This diagram doesn't typecheck, but note that it doesn't have to, because our strategy for evaluation of determiners treats circuits as syntactic objects to be manipulated.



Example 1.4.14 (Quantifiers I).

`Bob drinks all the beers` (in context)

In a finitary context, drinking `all the beers` amounts to applying the distributivity of **and** iteratively in that context. In this case, `all the beers` is treated as a reference-in-context to `Hells` and `Duvel`. In the same manner, existential quantifiers in finite contexts can be treated as finitary disjunctions, which is handled by creating alternative circuits, as in Example **??**



Example 1.4.15 (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 in at least two ways. The first essentially truth-conditional approach is to treat the generic as a process-theoretic condition governing measurements: whenever it is the case that something is a beer, it is the case that Bob drinks it. The second "inferential" appraoch is to treat the generic as a rewrite of text circuits conditioned on a beer test: whenever something is a beer we may add on a gate witnessing that Bob drinks that beverage.