GRAPHICAL CALCULI FOR NATURAL LANGUAGE

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.)

Novel contributions:

- Section REF is a pedestrian introduction to weak n-category theory (via `homotopy.io`, underpinned by the theory of associative n-categories) from the perspective of generalising familiar string-rewrite systems to higher dimensions. The chief development of this section is a demonstration that context-free grammars and tree-adjoining grammars may be formalised in the n-categorical setting.
- Section REF spells out a generative grammar for text using an n-categorical signature as a rewrite system, which additionally provides a unified framework from which the Text Circuit Theorem first proved in CITE is recovered.
- Section REF introduces the category **ContRel** of continuous relations. I detail the relationships (or lack thereof) of **ContRel** to its cousins **Top** and **Rel**. Though **ContRel** is constructed naïvely, its definition and an exposition of its expressivity from the monoidal perspective appears to be novel.
- Section REF string-diagrammatically characterises set-indexed collections of disjoint open subsets of spaces in **ContRel** as *sticky spiders* – special frobenius algebras that satisfy certain interaction relations with an idempotent. The diagrammatic outcome is that reasoning with such set-indexed collections remains as graphically intuitive as with spiders.
- Section REF string-diagrammatically characterises a vocabulary of linguistic topological relations in **ContRel** such as simple connectedness, touching, insideness, and rigid motion.
- Section REF argues for the centrality of explaining communication as a criterion for formal approaches to syntax, and explores the relationship between productive and parsing grammars as organised by a monoidal cofunctor. A diagrammatic treatment of monoidal cofunctor boxes is introduced for this purpose.
- REF is a standalone introduction to the mathematical setup of Montague's *Universal Grammar*, aimed at modern algebraists. An outcome of this section is the placement of text circuits as a natural mathematical development in the broadly conceived programme of Montague Semantics.
- Appendix REF constructs a subcategory of sticky spiders in **ContRel** on the unit square that behaves as a model of **FinRel** equipped with a *Turing object* – a single object in a category with respect to which all other objects and morphisms between them may be encoded. This is of particular relevance to the linguistic phenomenon of *entification* – that essentially all grammatical categories of words and even phrases have noun-equivalents, e.g. `to run` $\simeq$ `running`, `Bob drinks` $\simeq$ `the fact that Bob drinks`. The presence of a Turing object in a symmetric monoidal category additionally provides a semantic model for higher-order processes of generic input type, and for *lasso diagrams*.
- In Section REF , by parsing text as circuits (Section REF) and using merge-boxes (Section REF) to interpret those circuits in **ContRel** as iconic representations equipped with a vocabulary of lingustic-topological concepts (Section REF), I sketch the computation of metaphors by hand.

0
Context and synopsis

There are potentially practical and theoretical benefits to a fresh mathematical take on basic linguistics. String diagrams are formal, intuitive, expressive, fun, and pretty. I review the relevant research context.

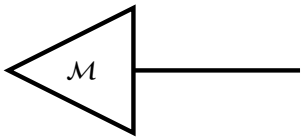


Figure 1: Let's say that the meaning of text is how it updates a model. So we start with some model of the way things are, modelled as data on a wire.

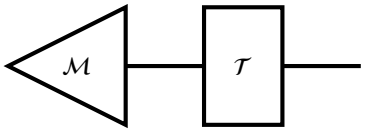


Figure 2: Text updates that model; like a gate updates the data on a wire.

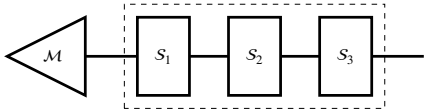


Figure 3: Text is made of sentences; like a circuit is made of gates and wires.

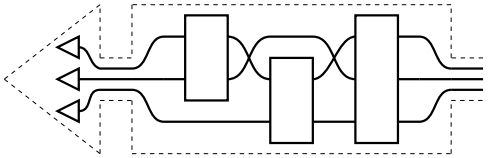


Figure 4: Let's say that ***The meaning of a sentence is how it updates the meanings of its parts.*** As a first approximation, let's say that the *parts* of a sentence are the nouns it contains or refers to. Noun data is carried by wires. Collections of nouns are related by gates, which play the roles of verbs and adjectives.

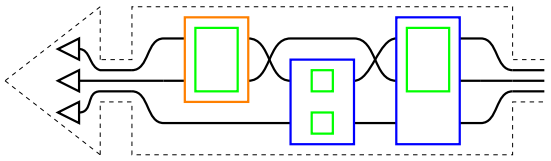


Figure 5: Gates can be related by higher order gates, which play the roles of adverbs, adpositions, and conjunctions; anything that modifies the data of first order gates like verbs.

0.1 What this thesis is about

This thesis is about studying language using string diagrams.

I am interested in using contemporary mathematical tools as a fresh approach to modelling some features of natural language considered as a formal object. Specifically, I am concerned with the compositional aspect of language, which I seek to model with the compositionality of string diagrams. Insofar as compositionality is the centrepiece of "knowledge of language", I share a common interest with linguists, but I will not hold myself hostage to their methods, literature, nor their concern with empirical capture. I will make all the usual simplifying assumptions that are available to theoreticians, such that an oracular machine will decide on lexical disambiguation and the appropriate parse using whatever resources it wants, so that I am left to work with lexically disambiguated words decorating some formal grammatical structure. It is with this remaining disambiguated mathematical structure that I seek to state a general framework for *meaningful compositional representations of text*, in the same way we humans construct rich and interactable representations of things-going-on in our minds when we read a storybook. So if you are interested in understanding language, this thesis is an invitation to a conception of formal linguistics that's maybe worth a damn in a world where large language models exist.

Objection: Isn't that reinventing the wheel?

Yes, to an extent. I am not interested in the human language faculty *per se*, so my aims differ. There are several potential practical and theoretical benefits that a fresh mathematical perspective on language enables. First, the mathematics of applied category theory allows us to unify different views of syntax, and conservatively generalise formal semantics to aspects of language that may have seemed beyond the reach of rigour, such as metaphor. Practically, the same mathematics allows us to construct interfaces between syntax/structure and semantics/implementation in such a way that we can control the former and delegate the latter by providing specifications without explicit implementation, which (for historical reasons I will explain shortly) is possibly the least-bad idea for getting at natural language understanding in computers from the bottom-up. Second, there are probably benefits to expressing linguistics in the same mathematical and diagrammatic lingua franca that can be used to represent and reason – often soundly and completely – about linear and affine algebra [Sob15, BSZ17, BPSZ19], first order logic [HS20], causal networks [LT23, JKZ19], signal flow graphs [BSZ14], electrical circuits [BS22], game theory [Hed15], petri nets [BM20], probability theory [FGP21], machine learning [CGG+22], and quantum theory [CD11, CK17, PWS+23], to name a few applications. At the moment, the practical achievements of language algorithms de-emphasise the structure of language, and there is no chance of reintroducing the study of structure with dated mathematics.

POINT OF INFORMATION: WHAT DO YOU MEAN BY NATURAL LANGUAGE?

Natural language is a human superpower, and the foundation of our collective achievements and mistakes as a species. By *natural language* I mean a non-artificial human language that some child has grown up speaking. English is a natural language, while Esperanto and Python are constructed languages. If you are still reading then you probably know a thing or two already about natural language. Insofar as there are rules for natural languages, it is probable that like most natural language users, you obey the rules of language intuitively without knowing what they are formally. For example, while you may not know what adpositions are, you know where to place words like `to`, `for`, `of` in a sentence and how to understand those sentences. At a more complex level, you understand idioms, how to read between the lines, how to flatter, insult, teach, promise, wager, and so on. There is a dismissive half-joke that "engineering is just applied physics", which we might analogise to absurdity as "law is just applied linguistics"; in its broadest possible conception, linguistics is the foundational study of everything that can possibly be expressed.

POINT OF INFORMATION: WHAT ARE STRING DIAGRAMS?

String diagrams are a heuristically natural yet mathematically formal pictorial syntax for representing complex, composite systems. I say *mathematically formal* to emphasise that string diagrams are not merely heuristic tools backed by a handbook of standards decided by committee: they are unambiguous mathematical objects that you can bet your life on [JS91**?** , Mac63, Lan10, Sel10].

String diagrams are also compositional blueprints that we can give semantics to – i.e. instantiate – in just about any system with a notion of sequential and parallel composition of processes. In particular, this means string diagrams may be interpreted as program specifications on classical or quantum computers, or as neural net architectures. Moreover, we can devise equations between string diagrams to govern the behaviour of interacting processes without having to spell out a bottom-up implementation.

Many fields of study have developed string diagrams as informal calculational aids, unaware of their common usage across disciplines and the rather new mathematics that justifies their use; everybody knows, but it isn't common knowledge. Why is that so? Because just as crustaceans independently converge to crab-like shapes within their own ecological niches by what is called *carcinisation*, formal notation for formal theories of "real world" problem domains undergo "string-diagrammatisation" in similar isolation. Why is that so? Because our best formal theories of the real world treat complexity as the outcome of composing simple interacting parts; perhaps nature really works that way, or we cannot help but conceptualise in compositional terms. When one has many different processes sending information to each other via channels, it becomes tricky to keep track of all the connections using one-dimensional syntax; if there are N processes, there may be on the order of $\mathcal{O}(N^2)$ connections, which quickly becomes unmanageable to write down in a line, prompting the development of indices in notation to match inputs and outputs. In time, probably by doodling a helpful line during calculation to match indices, link-ed indices become link-ing wires, and string-diagrammatisation is complete.

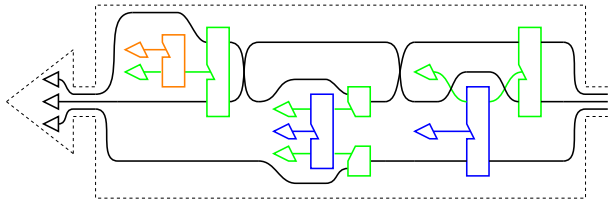


Figure 6: In practice, higher order gates may be implemented as gates that modify parameters of other gates. Grammar, and *function words* – words that operate on meanings – are in principle absorbed by the geometry of the diagram. These diagrams are natural vehicles for *dynamic semantics* CITE , broadly construed, where states are prior contexts and sentences-as-processes update prior contexts.

Definition 0.1.1 (Text Circuits)**.** *Text circuits* are made up of three ingredients:

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

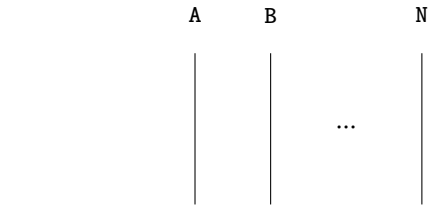


Figure 7: Nouns are represented by wires, each 'distinct' noun having its own wire.

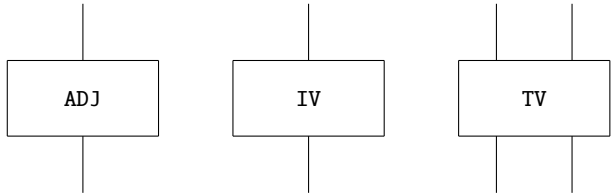


Figure 8: We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires. Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

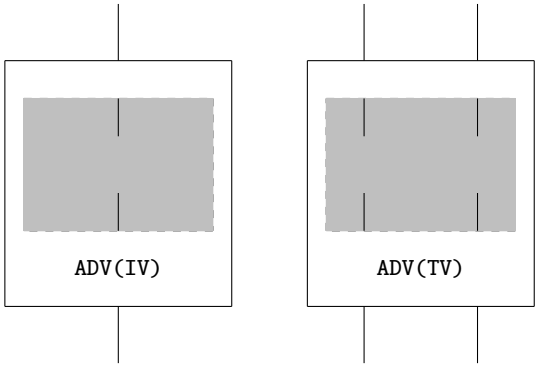


Figure 9: Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicating the shape of gate expected, and these should match the input- and output-wires of the box with the whole.

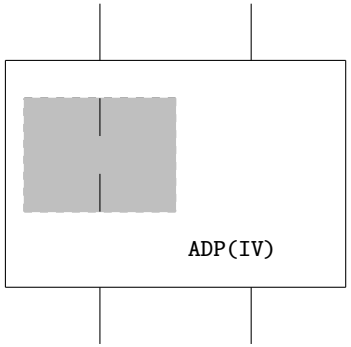


Figure 10: Similarly, adpositions also modify verbs, by moreover adding another noun-wire to the right.

0.2 ***Question:*** *What is the practical value of studying language when Large Language Models exist?*

This is the devastating question. Although this thesis is pure theory, I wish to address the question of practical value early because I imagine practical people are impatient. I will summarise the stakes: LLMs raise questions of existential concern for the field of linguistics. More narrowly, they demand justification as to why I am writing a thesis about theoretical approaches to basic linguistics as a computer scientist in current year. I will note in passing that I have an ugly duckling problem, in that I am not strictly aligned with machine learning, nor linguists broadly construed, nor mathematical linguists. I feel enough affinity to have defensive instincts for each camp, but I am distanced enough from each that I fear attacks from all sides. Perhaps a more constructive metaphor than war is that I am writing in a cooperative spirit between domains, or that I am an arbitrageur of ideas between them. With that in mind, I am for the moment advocating on behalf of pen-and-paper-and-principles linguists in formulating a two-part reply to the devastating question, and I will switch sides later for balance. First a response that answers with practical values in mind, and then a response that asserts and rests upon the distinct values of linguists.

Point of information: What are Large language models? Assume that everything about LLMs is prefaced with "at the time of writing", because the field is developing so quickly. Large Language Models are programs trained using a lot of data and a lot of compute time to predict the next word in text, a task for which computational techniques have evolved from Markov n-grams to transformers [VSP+17]. This sounds unimpressive, but in tandem with methods such as fine-tuning from human feedback in the case of chatGPT [Ope22] it is enough to tell and explain jokes [Bas22], pass the SAT [ted22] and score within human ranges on IQ tests [Tho22]. There is an aspect of genuine scientific and historical surprise that text-prediction can do this kind of magic. On the account of [MN21], computational linguistics began in a time when compute was too scarce to properly attempt rationalist, knowledge-based and theoretically-principled approaches to modelling language. Text-prediction as a task arose from a deliberate pursuit of "low-hanging fruit" as a productive and knowledge-lean alternative to doing nothing in an increasingly data-rich environment. Some observers [Chu11] expressed concern that the fruit would be quickly picked bare but those concerns are now evidently unfounded.

I'm sure there will be many further notable developments, and to be safe I won't make any claims about what machines can't do if we keep making them bigger and feed them more data or have them interact with one another in clever ways. Nonetheless there remain limitations that seem persistent for the foreseeable future, not in terms of *capabilities*, but in terms of *interpretability, explainability and safety*. These models have a tendency to hallucinate facts and are (ironically, for a computer) bad at arithmetic [HBK+21]. I imagine that the cycle of discovering limitations and overcoming them will continue. Despite whatever limitations exist in the state-of-the-art, it is evident to all sane observers that this is an important technology, for several reasons.

1. LLMs are a civilisational milestone technology. A force-multiplication tool for natural language – the universal interface – built from abundant data and compute in the information age may have comparably broad, deep, and lasting impact to the conversion of abundant chemical fuel to physical energy by steam engines in the industrial revolution.

2. LLMs represent a paradigm shift for humanity because they threaten our collective self-esteem, in a more pointed manner than losing at chess or Go to a computer; modifying a line of thinking from [Flo14], LLMs demonstrate that language and (the appearance of) complex thought that language facilitates is not a species-property for humans, and this stings on par with Darwin telling us we are ordinary animals like the rest, or Galileo telling us our place in the universe is unremarkable.

3. LLMs embody the latest and greatest case study of the bitter lesson [Sut19]. The tragedy goes like this: there's a group of people who investigate language – from syntax and semantics to pragmatics and analogies and storytelling and slang – who treat their subject with formal rigour and have been at it for many centuries. Their role in the story of LLMs is remarkable because it doesn't exist. They were the only qualified contestants in a "let's build a general-purpose language machine" competition, and they were a no-show. Now the farce: despite the fact that all of their accumulated understanding and theories of language were left out of the process, the machine is not only built but also far exceeds anything we know how to build in a principled way out of all their hard-earned insight. That is the bitter lesson: dumb methods that use a lot of data and compute outperform clever design and principled understanding.

0.3 ***First Reply:*** *Interpretability, maybe.*

Expressing grammar as composition of processes might yield practical benefits. Moreover, we want economy, generality, and safety for language models, and we can potentially do that with few tradeoffs if we use the right framework. Simplified, half of the problem of learning language is learning the meaning of words. The meanings change over time and are context-dependent, and the words are always increasing in number. Encoding these meanings by hand is a sisyphean task. Data-driven learning methods are a good fit: the patterns to be learnt are complex and nebulous, and there is a lot of data. However, data-driven methods may be weaker at the second half of the problem: learning and executing the composition of meanings according to syntax. We can see just how much weaker when we consider the figures involved in 'the poverty of the stimulus'.

Point of information: What is the poverty of the stimulus? In short, this famous problem is the observation that humans learn language despite having very little training data, in comparison to the complexity of the learned structure. It is on the basis of this observation – alongside many others surrounding language acquisition and use – that Chomsky posits [Cho00] that language is an innate human faculty, the

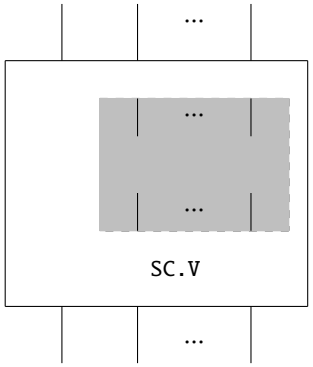


Figure 11: For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit.

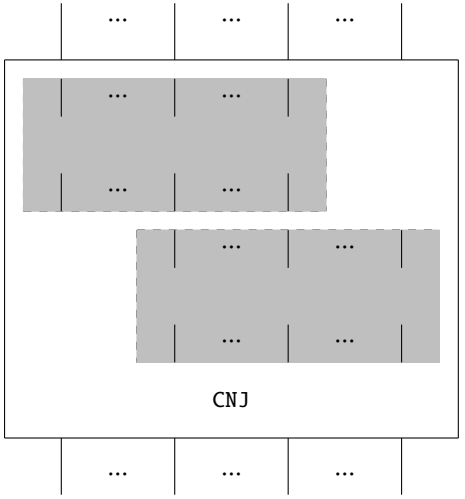


Figure 12: Conjunctions are boxes that take two circuits which might share labels on some wires.

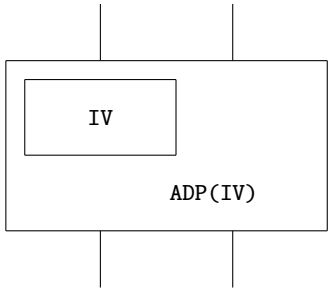


Figure 13: Of course filled up boxes are just gates.

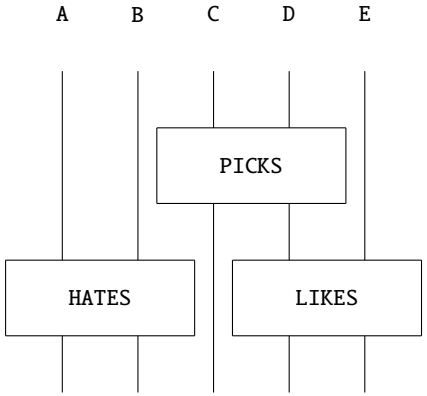


Figure 14: Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits.

development of which is less like effortfully going to the gym and more like effortlessly growing arms you were meant to have. The explanation goes like this: we can explain how a complex structure like grammar gets learnt from a small amount of data if everyone shares an innate Universal Grammar with a small number of free parameters to be learned. Whether or not the intermediate mechanism is a species-property of humans, the point is that we humans get a very small amount of input data, that data interacts with the mechanism in some way, and then we know a language. So, now that there are language-entities that are human-comparable in competence, we can make a back-of-the-envelope approximation of how much work the intermediate mechanism is doing or saving by comparing the difference in how much data and compute is required for both the human and for the machine to achieve language-competence. Humans get about 1.5 megabytes of data [MP19], 90 billion neurons [HH12], and an adult human consumes around 500 calories per day for thinking, for let's say 20 years of language learning. Rounding all values *up* to the closest order of magnitude, this comes to a cost metric of 10^{29} bits $\times$ joules $\times$ neurons. PaLM – an old model which is by its creators' account the first language model to be able to reason and joke purely on the basis of linguistic ability and without special training [CND+22, NC22] – required 780 billion training tokens of natural language (let's discount the 198 gigabytes of source code training data), which we generously evaluate at a rate of 4 characters per token [Kha23] and 5 bits per character. The architecture has 540 billion neurons, and required 3.2 million kilowatt hours of energy for training [Tom22]. Rounding values for the three units down *down* to the nearest order of magnitude comes to a cost metric of 10^{41} bit-joule-neurons. Whatever the human mechanism is, it is responsible for an order of magnitude in efficiency *give or take an order of magnitude of orders of magnitude*. It's possible that over time we can explain this difference away by various factors such as the efficiency of meat over minerals, separating knowledge of the world from knowledge of language, more efficient model architectures, or the development of efficient techniques to train new language models using old ones [TGZ+23]. One thing is clear: if it is worth hunting a fraction of a percent of improvement on a benchmark, forget your hares, a 10^{10} factor is a stag worth cooperating for.

Point of information: What progress have linguists made on this problem? The linguistic strategy for hunting the stag starts with what we know about how the mechanism between our ears works with language. The good news is that the chief methodology of armchair introspection is egalitarian and democratic. The bad news is that it is also anarchistic and hard-by-proximity; we are like fish in water, and it is hard for fish to characterise the nature of water. So the happy observations are difficult to produce and easily verified, and that means there are just a few that we know of that are are unobjectionably worth taking into account. One, or *the* such observation is *systematicity*. The intuition is best summarised by a quote. "Just as you don't find linguistic capacities that consist of the ability to understand sixty-seven unrelated sentences, so too you don't find cognitive capacities that consist of the ability to think seventy-four unrelated thoughts." (Fodor and Pylyshyn [FP88]) CITE .

POINT OF INFORMATION: SYSTEMATICITY? Systematicity refers to when a system can (generate/process) infinitely many (inputs/outputs/expressions) using finite (means/rules/pieces) in a "consistent" (or "systematic") manner. In short, how systems (like our capacity for language) achieve infinite ends by finite means. Like pornography, examples are easier than definitions. For example(s); we observe that anyone capable of understanding `Alice likes Bob` seems also to be capable of understanding `Bob likes Alice`; we know finitely many words but we can produce and understand potentially infinitely many texts; we can make infinitely many lego sculptures out of finitely many types of pieces; we can describe infinite groups and other mathematical structures using finitely many generators and relations; in the practical domain of computers, systematicity is synonymous with programmability and expressibility.

POINT OF INFORMATION: DO WE HAVE MATHS FOR SYSTEMATICITY? Yes, and I will consider it to be whatever it is that applied category theorists study. The concepts of systematicity and compositionality are deeply linked, because the only way we know how to achieve systematicity in practice is by a compositional systems, which can achieve infinite ends by finite means. Frege's initial conception of compositionality [Fre84] was borne of meditations on language, and states that a whole is the sum of its parts. Later conceptions of compositionality, the most notable deviation arising from meditations on quantum theory, generalises Frege's set-function conception of compositionality by varying the formal definitions of parts and the method of summation, and weakening the identification of the wholes with its parts to methods of keeping track of the relationships between wholes and parts [Coe21].

RETURNING TO THE STAG: So our starting point is that language is systematic and systematicity is the empirical surface of compositionality as far as we know, so compositionality is probably part of the solution to the poverty of the stimulus, if not most of it. The reasoning above should clarify why some folks don't think LLMs have anything to do with language as we humans do it. Their issue with purely data-driven architectures is either that we know immediately that they cannot be operating upon their inputs in a compositional way, or perhaps they appear to but their innards are too large and their workings too opaque to tell with confidence. Insofar as the task of learning language splits between learning meanings and learning the compositional rules of syntax that give rise to systematicity, the framework I present in this thesis is a proposal to split the cake sensibly between the two halves of the problem: meanings for the machines, and we'll supply the compositional rules. Syntax is still difficult and vast, but the rules are finite and relatively static. We can crack the black-box by treating syntax as directions for composition of smaller black-boxes that handle semantics. We all stand to benefit: we may give machines an easier time – now they only have to learn the meanings of words well – and we might gain confidence that the internal representations of the machine – their "mind's eye" – contains something we can probe and understand.

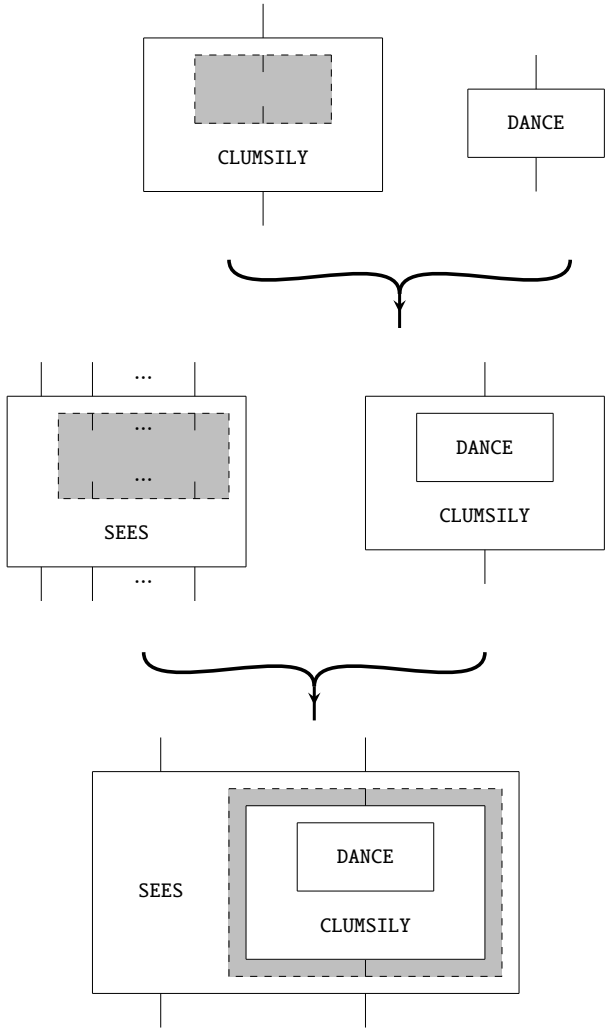


Figure 15: To summarise: composition by nesting corresponds to grammatical structure within sentences. Sentences correspond to filled gates, boxes with fixed arity correspond to first-order modifiers such as adverbs and adpositions, and boxes with variable arity correspond to sentential-level modifiers such as conjunctions and verbs with sentential complements.

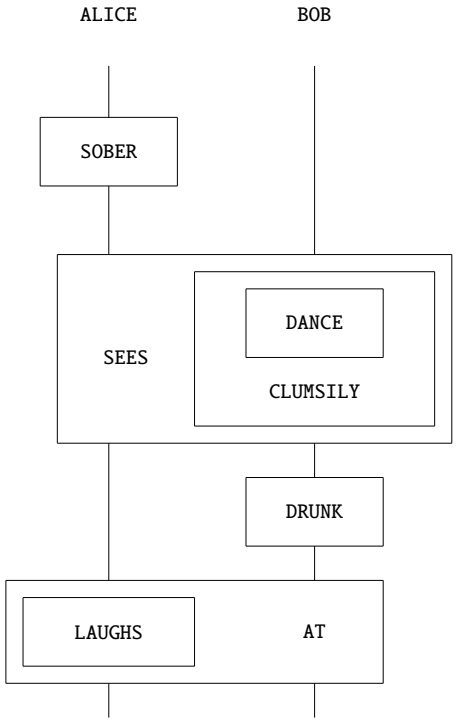


Figure 16: Composition by connecting wires corresponds to identifying coreferences in discourse. We obtain the same circuit for multiple text presentations of the same content, e.g. `Sober Alice who sees drunk Bob clumsily dance laughs at him.` yields the same circuit as the text `Alice is sober. She sees Bob clumsily dance. Bob is drunk. She laughs at him.` More details in Section REF .

0.3.1 ***Objection:*** *You're forgetting the bitter lesson.*

The bitter lesson is so harsh and often-enough repeated that this viewpoint is worth addressing proactively. The caveat that saves us is that the curse of expertise applies only to the object-language of the problem to be solved, not model architectures. We agree that qualitative improvements in problem-solving ability rarely if ever arise from encoding expert knowledge of the problem domain. Instead, these improvements come from *architectural* innovations, which means altering the parts and internal interactions of a model: changing *how* it thinks rather than *what* it thinks, to paraphrase Sutton's original prescription. We have good historical evidence that this prescription works, which we see by tracing the evolutionary path for data-driven language models from markov chains to deep learning [LBH15], RNNs [RM87], LSTMs [HS97], and now transformers [VSP+17]. Such structural changes are motivated by understandings (at varying degrees of formality) of the "geometry of the problem" [BBCV21]. The value proposition here is that with an appropriate mathematical lingua franca for structure, composition, and interaction, we can mindfully design rather than stumble upon the "meta-methods" Sutton calls for, allowing experts to encode *how* machines think and discover rather than *what*. Importing compositional and structural understanding from linguistics to machine learning via string diagrams might allow us to cheat the bitter lesson in spirit while adhering to the letter, and there is some preliminary empirical evidence for this, which I report on in Section REF .

0.3.2 ***Objection:*** *GOFAI? GO-F-yourself!*

Hostility (or at least indifference) to symbolic approaches is a stance espoused by virtually all of modern machine learning, and for good reasons. This stance is worth elaborating and steelmanning for pen-and-paper-people in the context of engineering language applications.

First, many linguistic phenomena are nebulous [Cha10]: the boundary of a simile is like that of a cloud, not sharp like the boundary of a billiard ball. Second, linguistic phenomena are complex, dynamic, and multifactorial: there are so many interacting mechanisms and forces in the production and comprehension of language that it is plausibly "computationally irreducible" [Wol02], or a "type 2" problem [Mar77], both terms referring to a kind of computational difficulty where the only explanation of a system amounts to a total computational simulation of it. Third, nebulousity and irreducibility together weakly characterise the kinds of problem domains where machine learning shines, so add to all this that we can achieve better results by caring less, c.f. Jelinek on speech-recognition: "Every time I fire a linguist, the performance of the speech recognizer goes up". So for the practical person, these are very good reasons to not bother with trying to understand or "break down" the phenomenon in a principled way as part of the process of engineering an application.

So what good are pen-and-paper theories as far as practical applications are concerned? To borrow terms from concurrency, there is already plenty of liveness, what is needed is more safety; liveness is when the program does something good, and safety is a guarantee it won't do something bad. For example, there is

ongoing work in integrating LLMs with stuctured databases for uses where facts and figures and ontologies matter; there is still a need for safeguards to prevent harmful outputs and adversarial attacks like prompt injection; while LLMs give a very convincing impression of reasoned thought, we would like to be sure if ever we decide to use such a machine for anything more than entertainment, such as assisting a caregiver in the course of healthcare decisions.

The good news is that symbolic-compositional theories are the right shape for safety concerns, because they can be picked apart and reasoned about. It is clear however that symbolic-compositional approaches by themselves are nowhere near achieving the kind of liveness LLMs have. Therefore, the direction of progress is synthesis.

0.3.3 ***Objection:*** *How does any of this improve capabilities?*

It's not meant to. The core value proposition for synthesis is interpretable AI, which operates in a manner we can analyse, and if appropriate, constrain. When lives are on the line (or more gravely, when capital is at risk), we would like to be certain that outputs are backed by guarantees. For this purpose, merely knowing *what* a deep-learning model is thinking is not enough[1]: i.e. solving something like symbol-grounding alone is a necessary but insufficient component. For instance, merely knowing what the weights of subnetworks of an image classification model represent does not meet our requirement of an understanding of the computations that manipulate those representations. It would be nice to simply tell the AI *how to behave* in such-and-such a way according to common sense, but having it do as you mean and not as you say is such a difficult problem that it has a name: *alignment*, and it's worth noting that acategory theory underpins some of the most promising approaches to this problem [dav]. This isn't to say that techniques such as reinforcement learning from human feedback cannot in principle succeed at doing precisely what we want for alignment, it's just that a constructive methodology of verifying or guaranteeing success to the bulletproof epistemic standards of mathematics remains wanting. Our best bet is some kind of symbolic-compositional structure for us to begin reasoning about the innards of the machines.

The investigation of the common ground between symbolic-composition and connectionism takes on, I suggest, essentially two, dual forms. The first kind uses connectionist methods to simulate symbolic-composition, which we can see the beginnings of in LLMs by examples such as chain-of-thought reasoning [WWS+23] and by probing their behaviour with respect to understood symbolic models [KWM23]. The second kind is the inverse, where connectionist architectures are organised and reasoned with by symbolic-compositional means. Some examples of the first kind include implementing data structures as operations on high-dimensional vectors, taking advantage of the idiosyncrasies of linear algebra in very high dimension [Kan19], or work that explores how the structure of word-embeddings in latent space encode semantic relationships between tokens. Some examples of the second kind include reasoning about the capability of graph neural networks by identifying or isolating their underlying compositional structure [LGT23], or architectures

[1] I recount the following from [Sø23], which argues that symbol-grounding is solvable from data alone, and in the process surveys the front of the symbol-grounding problem in AI: the issue of whether LLMs encode what words refer to and mean. On the account of [BK20], the performance of current LLMs is a form of Chinese Room [Sea80] phenomenon, so no amount of linguistic competence can be evidence that LLMs solve the symbol-grounding problem. However, the available evidence appears to suggest otherwise. For example, large models converge on word embeddings for geographical place names that are isomorphic to their physical locations [LAS21]. Since we know that brain activity patterns encode abstract conceptual space with the same mechanisms as they do physical spaces [KS16], extrapolating the ability of LLMs to encode spatially-analogical representations would in the limit suggest that LLMs encode meanings in a way isomorphic to how we do, modulo the token-word distinction and so long as we take seriously some version of Gärdenfors' [Gä14] thesis that meaning is encoded geometrically.

whose behaviour arises from compositional structure using neural nets as constituent parts, such as GANs [GPAM+14] and gradient boosted decision trees [CG16]. The work in this thesis builds upon a research programme – DisCoCat [CSC10], elaborated in Section REF – which lies somewhere in the middle of a duality of approaches to merging connectionism and symbolic-composition. It is, to the best of my knowledge, the only approach that explicitly incorporates mathematically rigourous compositional structures from the top-down alongside data-driven learning methods from the bottom-up. Fortifying this bridge across the aisle requires a little give from both sides; I ask only that reader entertain some pretty string diagrams.

0.4 ***Second Reply:*** *LLMs don't help us understand language; how might string diagrams help?*

Another way to deal with the devastating question of LLMs is to reject it, on the basis that using or understanding LLMs is completely different from understanding language, and language is worth understanding in its own right. To illustrate this point by a thought experiment, what would linguistics look like if it began today? LLMs would appear to us as oracles; wise, superhumanly capable at language, but inscrutable. Similarly, most people effortlessly use language without a formal understanding which they can express. So the fundamental mystery would remain unchanged. Understanding how an LLM works at the algorithmic level cannot help[2]. Borrowing and bastardising a thought from Marr, suppose you knew the insides of a mechanical calculator by heart. Does that mean you understand arithmetic? At best, obliquely, and maybe not at all: the calculator is full of inessentialities and tricks to fit platonic arithmetic against the constraints of physics, and you would not know where the tricks begin and the essence ends. Similarly, suppose you knew every line of code within and every piece of data used to train an LLM; does that mean you understand how language works? How does one delineate what is essential to language, and what is accidental? So let's forget about LLMs. The value proposition to establish now is how string diagrams and some category theory comes into the picture for the formal linguist who is concerned with understanding how language works, and that's the whole rest of the thesis. I sense one more objection from the practical reader, and one from the theoretical reader, so I'll address them in that order before moving on.

0.4.1 ***Objection:*** *Isn't the better theory the one with better predictions?*

LLMs are a theory of language in the same way a particular human brain is a theory of cognition; at best, debatablely. There are various criteria – not all independent – that are arguably necessary for something to qualify as an explanatory theory, and while LLMs satisfice (or even excel) at some, they fail at others. Empirical adequacy – the ability of theory to account for the available empirical data and make good predictions about future observations – is one such criterion, and here LLMs excel. In constast to the idealised and partial nature of formal theories, the nature of LLMs is that they are trained on empirical data about language that captures the friction of the real world. So, in terms of raw predictive power, we should naturally expect

[2] But there is a worthwhile observation we can make from an understanding of the computational aims of LLMs. Insofar as the computational aim of a finished LLM is purely to predict the most plausible next token (modulo RLHF and with respect to a massive corpus), it is now an empirical fact that the artefact of language as it exists outside of human users carries sufficient structure to reconstruct the appearance of novel complex thought processes. I cannot understand why linguists are not all deeply excited at the possibilities. If it is the case that we learn such complex thought processes in the first place from language, we might elevate our consideration of language from a technology or tool to an equal and symbiotic partnership with its users as a living repository of disembodied cognition; linguists stand to be promoted from archaeologists to keybearers of *thinking*. The existence of competent non-human language users tantalises the exploration of language as a phenomenon in its own right, outside of the cognitive turn and the human perspective – consider that if aliens were discovered tomorrow, xenobiologists would simply be called biologists; why should the study of language remain parochial when the aliens landed yesterday? Plus our aliens don't mind vivisection! However, such radical reconceptions of language have not yet been articulated, so it remains that LLMs do not help linguists do linguistics in its current conception.

the LLMs to have an advantage over principled theories. They are so good at empirical capture that to some degree they automatically satisfy the related criteria of coherence – consistency with other established linguistic theories – and scope – the ability to capture a wide range of phenomena. But while empirical capture is necessary for explanatory theories, it is insufficient.

There are several criteria where the adequacy of LLMs is unclear or debatable. Fruitfulness is a sociological criterion for goodness of explanatory theories, in that they should generate new predictions and lead to further discoveries and research. While they are certainly a potent catalyst for research in many fields even beyond machine learning, it is unclear for now how they relate to the subject matter of linguistics. Whether they satisfy Popper's criterion of falsifiability is as of yet not determined, because it is not settled how to go about falsifying the linguistic predictions of LLMs, or even express what the content of a theory embodied by an LLM is. The closest examples to falsifiability that come to mind are tests of LLM fallibility for reasoning and compositional phenomena [DLS+23], or their weakness to adversarial prompt-injections [noa22], but these weaknesses do not shed light on their linguistic competence and "understanding" directly.

Now the disappointments. As far as we can tell; LLMs are far from simple, and simplicity (Occam's Razor) is an ancient criterion for the goodness of explanation; while they exhibit, they do not explain the structure, use, and acquisition of language; they do not unify or subsume our prior understanding of linguistics. The first two points are basically unobjectionable, so I will briefly elaborate on the criterion of unification and subsumption of prior understandings, borrowing a framework from cognitive neuroscience. A common methodology for investigating cognitive systems is Marr's 3 levels CITE (poorly named, since they are not hierarchical, but more like interacting domains.) Level 1 is the computational theory, an extensional perspective that concerns tasks and functions: at this level on asks what the contents and aims of a system are, to evaluate what the system is computing and why, respectively. Level 2 is representation and algorithm, an intensional perspective that concerns the representational format of the contents within the system, and the procedures by which they are manipulated to arrive at outcomes and outputs. Level 3 is hardware, which concerns the mechanical execution of the system, as gears in a mechanical calculutor or as values, reads, and writes in computer memory. In the case of LLMs, we understand well the nature of computational theory level, at least in their current incarnation as next-token-predictors, which is a narrow and clear task. Furthermore, we understand the hardware level well, from the silicon going up through the ladder of abstraction to software libraries and the componentwise activity of neural nets. Yet somehow, we know everything and nothing at once about the representation and algorithm level; we can explain how transformer models work in terms of attention mechanisms and lookback, and how it is that these models are trained using data to produce the outputs they do. In spite of understandings which should jointly cover all of level 2, we cannot relate their operations on language to our own.

To illustrate the insufficiency of empirical capture to make a theory, consider the historical case study of models of what we now call the solar system. The Ptolemaic geocentric model of the solar system was more empirically precise than the heliocentric Copernican, even though the latter was empirically "more correct". This should not be surprising, because Ptolemaic epicycles can overfit to approximate any observed trajectory of planets. It took until Einstein's relativity to explain the precession of perihelion of mercury, which at last aligned theoretical understanding with empirical observation. But Newton's theory of gravity was undeniably worthwhile science, even if it was empirically outperformed by its contemporaries. Consider just how divorced from reality Newton was: Aristotelian physics is actually correct on earth, where objects don't continue moving unless force is continually supplied, because friction exists. It took a radical departure from empirical concerns to the frictionless environment of space in order obtain the simplified and idealised model of gravity that is the foundation of our understanding of the solar system and beyond. The lessons as I see them are as follows. First, aimed towards some advocates of theory-free approaches, we should belay the order to evacuate linguistics departments because performance is to some degree orthogonal to understanding. In fact, the scientific route of understanding involves simplified and idealised models that ignore friction, and will necessarily suffer in performance while maturing, so one must be patient. Second, aimed towards some theoreticians, haphazard gluing together of different theories and decorating them with bells-and-whistles for the sake of fitting empirical observation is no different than adding epicycles; one must either declare a foundational or philosophical justification apart from empirical capture (which machines are better at anyway), or state outright that it's just a fun and meaningful hobby, like painting. Third, interpretability done well requires a suitable representation and level of abstraction; imagine an epicyclist explaining the precession of mercury's perihelion by pointing at a collection of epicycles and calling it a "distributed representation", and compare to prodding subnetworks.

Set theory sucks to build theories with, for a couple of reasons.

1. It hurts to read, it hurts to write, it hurts to think with. Pain selects for sadomasochists. If only there were an easier and prettier way to communicate ideas formally, but alas.

2. Suppose you're writing your system with blood sweat and tears (but you enjoy it). You are probably among perhaps a dozen people that are intimate enough with the intricacies of the system to modify and extend it, so your theory is one conference-bus accident away from goodbye. If only there were a broader community of people that shared a common mathematical competence for your kind of theorybuilding work who could carry on, but alas.

3. You're extending the system and you want to incorporate a new module from elsewhere, or relate your system to someone elses'. It turns out that the formation of connective tissue between theories is impeded by the mind-numbing busywork of translating foundations of formalisms and their encoding choices (but you enjoy it). If only there were some kind of mathematics where one could just work at the level of abstraction that suited them, but alas.

4. You're passing the torch to the next generation. Students of your field don't know enough set theory and first order logic – why can't they be more like mathematicians? – so it's a hard time learning and a hard time teaching (but you enjoy it). If only there were some way to step away from implementation details and get them to *see* the essence of the ideas, but alas.

0.4.2 ***Objection:*** *What's wrong with λ-calculus and sequent calculi and graphs and sets?*

Our capacity for language is one of the oldest and sophisticated pieces of compositional technology, maybe even the foundation of compositional thought. So, linguists are veteran students of compositionality and modularity. How does syntax compose meaning? How do the constraints and affordances of language interact? Concern number one for the formal study of language is having a metalanguage in which to build models and theories, and here we find our λs and sequents and whatever else.

Linguistics embodies a encyclopaedic record of how compositionality works "in the field", just as botanists record flowers, early astronomers the planetary motions, or stamp-collectors stamps. But a disparate collection of observations encoded in different formats does not a theory make; we will inevitably wish to bring it all together. Concern number two for the formal study of language is having a metametalanguage with which to relate the various metalanguages. Obviously, the metametalanguage is set theory, which is the gold standard that backs everything else.

The set theoretic standard was forced by a historical lack of alternatives, and as a result, serious formal linguists are applied set theorists. However, set theory is not well-suited for complex and interacting moving parts, because it demands bottom-up specifications. So for instance if one wishes to specify a function, one has to spell out how it behaves on the domain and codomain, which means spelling out what the innards of the domain and codomain are; to specify a set theoretic model necessitates providing complete detail of how every part looks on the inside. This is an innate feature of set theory. Consider the case of the cartesian product of sets, one of the basic constructions. $A \times B$ is the "set of ordered pairs" (a, b) of elements from the respective sets, but there are many ways of encoding ordered pairs that are equivalent in spirit but not in syntax; a sign that the syntax is a hindrance, or obfuscating something important. Here is a small sampling of different ways to encode an ordered pair. Kuratowski's definition is

$$A \times B := \left\{ \{\{a\}, \{a, b\}\} \mid a \in A \ , \ b \in B \right\}$$

Which could have just as easily been:

$$A \times B := \left\{ \{\{a, b\}, b\} \mid a \in A \ , \ b \in B \right\}$$

And here is Wiener's definition:

$$A \times B := \left\{ \{\{a, \varnothing\}, b\} \mid a \in A \ , \ b \in B \right\}$$

But we don't care what the precise implementation is so long as the property that $(a, b) = (c, d)$ just when $a = c$ and $b = d$ holds. The same kind of problem keeps occurring at all levels of complexity: suppose you

have a set-indexed set of things $\{T_i \mid i \in I\}$, which you can choose to implement as a function $I \to \mathbf{T}$. Then somebody else wants to make the indexing set dynamically updatable with novel elements, so they have to rephrase the indexing mechanism as a set of tuples $\{(T_{i^1}, i^1), (T_{i^2}, i^2), \cdots\}$ so that they can add or remove elements, and then someone else comes along and decides that the indexes have structure that disallow certain things to be indexed... All this means that if one wants to use set theory to relate different theories at a "structural" level, one must first analyse both in terms of their constituent sets and functions in order to construct more functions between sets and functions. As you may already know if you're in the business of articulating formal systems, representation-dependency makes this process a bureaucratic nightmare.

But it gets the job done, so why fix what ain't broke? Well, there is nothing wrong with punchcard machines as far as computability is concerned, but nobody uses them anymore because there are better options. String diagrams are the better option as a metalanguage for formal linguistics, and applied category theory is a better option as a metametalanguage.

Having a diversity of metalanguages is good fun science, but an *unrelated* diversity leads to sociological problems. It is a recipe for siloization and fracture when talking to your neighbour is difficult unless you have the time to invest in *their* mathematics. The value proposition of string diagrams here is that they are a lingua franca with basically no downsides for adoption. Just as high-level code is compiled to lower-level languages, string diagrams may be (in the brute-force case) used directly as an abstraction layer over sets and functions with the cartesian and direct products, so there is upside of marshalling the power of a visual cortex refined over aeons *at no cost* in terms of fundamental expressive capacity.

But what of the epistemic costs? Sets and functions are comprehensible, but who the hell knows about the category theory that underpins string diagrams? Just as being a good python programmer doesn't necessitate knowing what is happening at the level of hardware, using string diagrams to model and calculate doesn't necessitate knowledge of category theory. How can that be? Because while *installing* an abstraction layer requires a one-time epistemic payment, thereafter *using* the abstraction layer is epistemically free. For example, when a python programmer calls a matrix multiplication method from a code library in python, they don't have to know about matrix multiplication algorithms or how python compiles down to bytecode or how the von Neumann architecture works, because computer engineeers and hardware hackers and mathy codewriters have already done the translation work between layers of abstraction. In the case of string diagrams, the epistemic guarantees have been purchased [JS91**?** , Mac63, Lan10, Sel10], and there is a large and growing ecosystem of code libraries [Sob15, BSZ17, BPSZ19, HS20, LT23, JKZ19, BSZ14, BS22, Hed15, BM20, FGP21, CGG+22, CD11, CK17, PWS+23]. As for applied category theory, I can't make the case better than [FS19]. But as someone who has spent too much time with set-builder notation, the case against sets is in the margins.

5. Some goddamn upstarts from machine learning have eaten your lunch! The thought that they didn't even consult your expertise boils you (but you enjoy it). If those know-nothings can do all of that with data and a computer, surely *you* could computerise *your* principled and peer-reviewed system and make it work with "data-driven machine learning" and beat them at their own game! It's *obviously* doable, because neural nets are *merely* functions between sets of vectors, just like your λs and your graphs and your logic are also *really just* sets and functions. So all you have to do is translate your formalism into... what, exactly? If only there were some way to relate and reason about the common structure between distinct mathematical domains, but alas.

If you think that these are inevitable problems that come with the territory of being a serious linguist doing serious work with serious sets, stop reading. String diagrams are an aesthetic, intuitive, flexible, and rigourous metalanguage syntax that gives agency to the modeller by operating at a level of abstraction of their choice. Applied category theory as a meta^{2}language allows formal systems expressed in terms of string diagrams to be easily modified and related to one another. To achieve this, a particular formal system may be expressed as a finitely presented symmetric monoidal category, and relationships between theories are expressed as various kinds of symmetric monoidal functors, which are generalised structure-preserving maps.

The deeper objection here is that diagrams do not look like *serious* mathematics. The reasons behind this rather common prejudice are worth elaborating. This is the wound Bourbaki has inflicted. Nicolas Bourbaki is a pseudonym for a group of French mathematicians, who wrote a highly influential series of textbooks. It is difficult to overstate their influence. The group was founded in the aftermath of the First World War, around the task of writing a comprehensive and rigourous foundations of mathematics from the ground up. The immediate *raison-d'être* for this project was that extant texts at the time were outdated, because the oral tradition and living history of mathematics in institutions of learning in France were decimated by the deaths of mathematicians at war. In a broader historical context, Bourbaki was a reactionary response to the crisis in the foundations of mathematics at the beginning of the century, elicited by Russell's paradox. Accordingly, their aims were rationalist, totalitarian, and high-modernist, in line with their contemporary artistic and musical fashions; they wanted to write timelessly, to settle the issues once and for all. Consequently, Bourbaki's Definition-Proposition-Theorem style of mathematical exposition is a historical aberration: a bastardisation of Euclid that eschews intuition via illustration and specific examples in favour of abstraction and generality, requiring years of initiation to effectively read and write, and remaining *de rigeur* for rigour today in dry mathematics textbooks. The deeper objection arises from the supposition that serious mathematics ought to be arcane and difficult, as most mathematics exposition after Bourbaki is. The reply is that it need not be so, and that it was not always so! The Bourbaki format places emphasis and prestige upon the deductive activity that goes into proving a theorem, displacing other aspects of mathematical activity such as constructions, algorithms, and taxonomisation. These latter aspects are better suited for the nebulous subject matter of natural language, which doesn't lend itself well to theorems, but is a happy muse for mathematical play.

0.4.3 ***Objection:*** *Aren't string diagrams just graphs?*

Yes and no! This point is best communicated by a mathematical koan. Consider the following game between two players, you and me. There are 9 cards labelled 1 through 9 face up on the table. We take turns taking one of the cards. The winner is whoever first has three cards in hand that sum to 15, and the game is a draw if we have taken all the cards on the table and neither of us have three cards in hand that sum to 15. I will let you go first. Can you guarantee that you won't lose? Can you spell out a winning strategy? If you have never heard this story, give it an honest minute's thought before reading on.

The usual response is that you don't know a winning strategy. I claim that you probably do. I claim that even a child knows how to play adeptly. I'll even wager that you have played this game before. The game is Tic-Tac-Toe, also known as Naughts-and-Crosses: it is possible to arrange the numbers 1 to 9 in a 3-by-3 magic square, such that every column, row, and diagonal sums to 15.

The lesson here is that choice of representations matter. In the mathematical context, representations matter because they generalise differently. On the surface, here is an example of two representations of the same platonic mathematical object. However, Tic-Tac-Toe is in the same family as Connect-4 or 5-in-a-row on an unbounded grid, while the game with numbered cards generalises to different variants of Nim. That they coincide in one instance is a fork in the path. In the same way, viewing string diagrams as "just graphs" is taking the wrong path, just as it would be true but unhelpful to consider graphs "just sets of vertices and edges". String diagrams are indeed "just" a special family of graphs, just as much as prime numbers are special integers and analytic functions are special functions.

In a broader context, representations matter for the sake of improved human-machine relations. These two representations are the same as far as a computer or a formal symbol-pusher is concerned, but they make world of difference to a human native of meatspace. We ought to swing the pendulum towards incorporating human-friendly representations in language models, so that we may audit those representations for explainability concerns. As it stands, there is something fundamentally inhuman and behavioural about treating the production of language as a string of words drawn from a probability distribution. I don't know about you, but I tend to use language to express pre-existing thoughts in my head that aren't by nature linguistic. Even if we grant that the latent space of a data-driven architecture is an analog for the space of internal mental states of a human user of language, how can we know whether the spaces are structurally analogous to the extent that human-machine partnership through the interface of natural language is safe? So here again is a possible solution: by composing architectures in the shape of language from the start, we may begin to attempt guarantees that the latent-space representations of the machine are built up in the same way we build up a mental representation when we read a book or watch a film.

0.5 Synopsis of the thesis

I'm going to compute the semantics of some metaphors, via syntax, using string diagrams. It doesn't interest me whether it's been done before by other formal means, I only care to demonstrate the breadth and reach of string diagrams. All of the rest of the thesis until then is preparation for that exercise, and the remainder of this chapter after this section will deal with mathematical and scientific background.

First it will be necessary to justify some kind of systematic relationship between text circuits and something resembling text in natural language, which will be the purpose of Chapter REF . Here I introduce weak n-categorical signatures as generalisations of string rewrite systems to higher dimensions. I demonstrate that context-free, context-sensitive, and tree-adjoining grammars are all formalisable in this one setting, in which I then construct a generative grammar that simultaneously produces grammatical text as strings of words, and the requisite structure to obtain text circuits. This relationship between text circuits and text will be encapsulated in the Text Circuit Theorem. I close this section with some discussion of ongoing practical and theoretical developments in text circuits, and point out some avenues of generalisation.

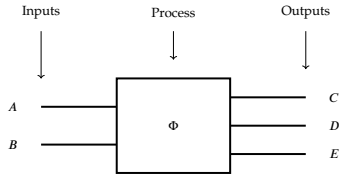
Once we have text circuits, we will need some monoidal category in which to interpret and calculate with them. In particular, it would be nice to calculate formally with the kinds of iconic cartoon representations that are typically used typically as informal schematic illustrations of metaphors. For this purpose, in Chapter REF I introduce **ContRel**, a symmetric monoidal category of continuous relations. I diagrammatically characterise set-indexed collections of disjoint open subsets of a space – i.e. shapes with labels – as idempotents that interact with a special frobenius algebra. I will then develop a vocabulary of linguistic topological concepts so that shapes can be connected or touching or inside one another, and I will make them move and dance as I please. All of that gets done using just equations between string diagrams, and the diagrams underpinning these iconic semantics are a natural basis upon which one can perform truth-conditional analyses.

Once we have text circuits and a formal setting to reason with and about cartoons, we require something that leverages the systematicity of a metaphor to form a structured correspondence between a text and its representation as a cartoon. In Chapter CITE , I introduce monoidal cofunctors for this purpose diagrammatically, in the process also settling some questions about how productive and parsing grammars ought to relate to each other in light of the fact that communication is possible. Then I will put everything together to compute a metaphor, by turning words into diagrams and diagrams into cartoons.

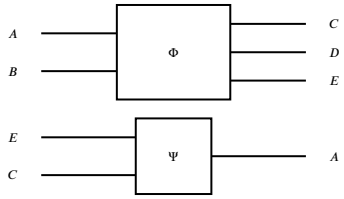
0.6 Process Theories

This section seeks to introduce process theories via string diagrams. The margin material will provide the formal mathematics of string diagrams from the bottum-up. The main body develops process theories via string diagrams by example, through which we develop towards a model of linguistic spatial relations – words like "to the left of" and "between" – which are a common ground of competence we all possess. Here we only focus on geometric relations between points in two dimensional Euclidean space equipped with the usual notions of metric and distance, providing adequate foundations to follow []talkspace, in which I demonstrate how text circuits can be obtained from sentences and how such text circuits interpreted in the category of sets and relations **Rel** provides a semantics for such sentences. This motivates the question of how to express the (arguably more primitive []piaget) linguistic topological concepts – such as "touching" and "inside" – the mathematics of which will be in Chapter **??**; the reader may skip straight to that chapter after this section if they are uninterested in syntax. We close this section with a brief note on how process theories relate to mathematical foundations and computer science.

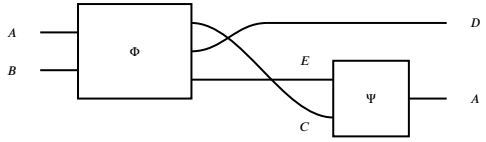
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. Unless otherwise specified, we read processes from left to right.



Processes may compose in parallel, depicted as placing boxes next to each other.



Processes may compose sequentially, depicted as connecting wires of the same type.



There are a lot of definitions to get started, but as with programming languages, preloading the work makes it easier to scale. This is the mathematical source code of string diagrams, which is only necessary if we need to show that something new is a symmetric monoidal category or if we are tinkering deeply, so it can be skipped for now. The important takeaway is that string diagrams are syntax, with morphisms in symmetric monoidal categories as semantics.

Definition 0.6.1 (Category)**.** A *category* C consists of the following data

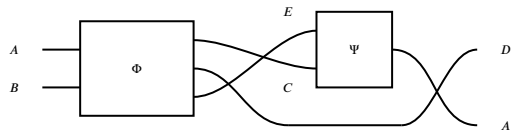
- A collection $\mathrm{Ob}(C)$ of *objects*
- For every pair of objects $A, B \in \mathrm{Ob}(C)$, a set $C(A, B)$ of *morphisms* from a to b.
- Every object $a \in \mathrm{Ob}(C)$ has a specified morphism 1_a in $C(a, a)$ called *the identity morphism* on a.
- Every triple of objects $A, B, C \in \mathrm{Ob}(C)$, and every pair of morphisms $f \in C(A, B)$ and $g \in C(b, c)$ has a specified morphism $(f; g) \in C(a, c)$ called *the composite* of f and g.

This data has to satisfy two coherence conditions, which are:

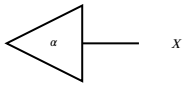
Unitality: For any morphism $f : a \to b$, $1_a; f = f = f; 1_b$

Associativity: For any four objects A, B, C, D and three morphisms $f : A \to B, g : B \to C, h : C \to D$, $(f; g); h = f; (g; h)$

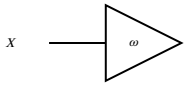
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



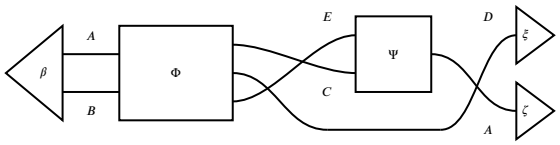
Some processes have no inputs; we call these *states*.



Some processes have no outputs; we call these *tests*.



A process with no inputs and no outputs is a *number*; the number tells us the outcome of applying tests to a composite of states modified by processes.



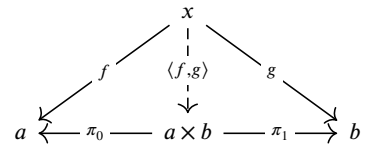
A process theory is given by the following data:

- A collection of systems
- A collection of processes along with their input and output systems
- A methodology to compose systems and processes sequentially and in parallel, and a specification of the unit of parallel composition.
- A collection of equations between composite processes

Example 0.6.12 (Linear maps with direct sum)**.** Systems are finite-dimensional vector spaces over $\mathbb{R}$. Processes are linear maps, expressed as matrices with entries in $\mathbb{R}$.

Sequential composition is matrix multiplication. Parallel composition of systems is the direct sum of vector

Definition 0.6.2 (Categorical Product)**.** In a category C, given two objects $a, b \in \mathrm{Ob}(C)$, the *product* $A \times B$, if it exists, is an object with projection morphisms $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$ such that for any object $x \in \mathrm{Ob}(C)$ and any pair of morphisms $f : X \to A$ and $g : x \to b$, there exists a unique morphism $f \times g : X \to A \times B$ such that $f = (f \times g); \pi_0$ and $g = (f \times g); \pi_1$. This is a mouthful which is easier expressed as a commuting diagram as below. The dashed arrow indicates uniqueness. $A \times B$ is a product when every path through the diagram following the arrows between two objects is an equality.



The idea behind the definition of product is simple: instead of explicitly constructing the cartesian product of sets from within, let's say *a product is as a product does*. For objects, the cartesian product of sets $A \times B$ is a set of pairs, and we may destruct those pairs by extracting or projecting out the first and second elements, hence the projection maps π_0, π_1. Another thing we would like to do with pairs is construct them; whenever we have some A-data and B-data, we can pair them in such a way that construction followed by destruction is lossless and doesn't add anything. In category-theoretic terms, we select 'arbitrary' A- and B-data by arrows $f : X \to A$ and $g : X \to B$, and we declare that $f \times g : X \to A \times B$ is the unique way to select corresponding tuples in $A \times B$. This design-pattern of "for all such-and-such there exists a unique such-and-such" is an instance of a so-called *universal property*, the purpose of which is to establish isomorphism between operationally equivalent implementations.

To understand what this style of definition gives us, let's revisit Kuratowski's and Wiener's definitions of cartesian product, which are, respectively:

$$A \overset{K}{\times} B := \left\{ \{\{a\}, \{a,b\}\} \mid a \in A\ ,\ b \in B \right\}$$

$$A \overset{W}{\times} B := \left\{ \{\{a,\varnothing\}, b\} \mid a \in A\ ,\ b \in B \right\}$$

Keeping overset-labels and using maplet notation, the associated projections are:

$$\overset{K}{\pi_0} := \{\{a\}, \{a,b\}\} \mapsto a$$

$$\overset{K}{\pi_1} := \{\{a\}, \{a,b\}\} \mapsto b$$

$$\overset{W}{\pi_0} := \{\{a,\varnothing\}, b\} \mapsto a$$

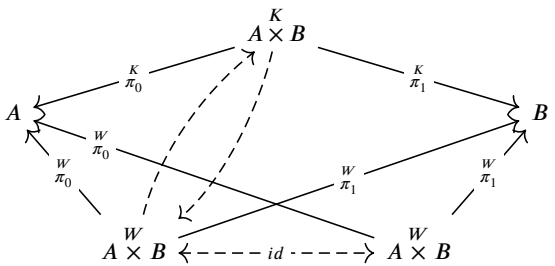
$$\overset{W}{\pi_1} := \{\{a,\varnothing\}, b\} \mapsto b$$

And maps f, g into A and B are tupled by the following:

$$f \overset{K}{\times} g := x \mapsto \{\{f(x)\}, \{f(x), g(x)\}\}$$

$$f \overset{W}{\times} g := x \mapsto \{\{f(x), \varnothing\}, g(x)\}$$

Both satisfy the commutative diagram defining the product. Something neat happens when we pick $A \overset{K}{\times} B$ to be the arbitrary X for the product definition of $A \overset{W}{\times} B$ and vice versa. We get to mash the commuting diagrams together:



spaces $\oplus$. The parallel composition of matrices $\mathbf{A}, \mathbf{B}$ is the block-diagonal matrix

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}$$

The unit of parallel composition is the singleton 0-dimensional vector space. States are row vectors. Tests are column vectors. The numbers are $\mathbb{R}$. Usually the monoidal product is written with the symbol $\otimes$, which clashes with notation for the hadamard product for linear maps, while the process theory we have just described takes the direct sum $\oplus$ to be the monoidal product.

Example 0.6.13 (Sets and functions with cartesian product)**.** Systems are sets A, B. Processes are functions between sets $f : A \to B$. Sequential composition is function composition. Parallel composition of systems is the cartesian product of sets: the set of ordered pairs of two sets.

$$A \otimes B = A \times B := \{(a,b) \mid a \in A, b \in B\}$$

The parallel composition $f \otimes g : A \times C \to B \times D$ of functions $f : A \to B$ and $g : C \to D$ is defined:

$$f \otimes g := (a,c) \mapsto (f(a), g(c))$$

The unit of parallel composition is the singleton set $\{\star\}$. There are many singletons, but this presents no problem for the later formal definition because they are all equivalent up to unique isomorphism. States of a set A correspond to elements $a \in A$ – we forgo the usual categorical definition of points from the terminal object in favour of generalised points from the monoidal perspective. Every system A has only one test $a \mapsto \star$; this is since the singleton is terminal in **Set**. So there is only one number.

Example 0.6.14 (Sets and relations with cartesian product)**.** Systems are sets A, B. Processes are relations between sets $\Phi \subseteq A \times B$, which we may write in either direction $\Phi^* : A \nrightarrow B$ or $\Phi_* : B \nrightarrow A$. Relations between sets are equivalently matrices with entries from the boolean semiring. Relation composition is matrix multiplication with the boolean semiring. Φ^*, Φ_* are the transposes of one another. Sequential composition is relation composition:

$$A \overset{\Phi}{\nrightarrow} B \overset{\Psi}{\nrightarrow} C := \{(a,c) \mid a \in A,\ c \in C,\ \exists b_{\in B} : (a,b) \in \Phi \wedge (b,c) \in \Psi\}$$

Parallel composition of systems is the cartesian product of sets. The parallel composition of relations $A \otimes C \overset{\Phi \otimes \Psi}{\nrightarrow} B \otimes D$ of relations $A \overset{\Phi}{\nrightarrow} B$ and $C \overset{\Psi}{\nrightarrow} D$ is defined:

$$\Phi \otimes \Psi := \{\big((a,c),(b,d)\big) \mid (a,b) \in \Phi \wedge (c,d) \in \Psi\}$$

The unit of parallel composition is the singleton. States of a set A are subsets of A. Tests of a set A are also

subsets of A.

0.6.1 What does it mean to copy and delete?

Now we discuss how we might define the properties and behaviour of processes by positing equations between diagrams. Let's begin simply with two intuitive processes *copy* and *delete*:

Example 0.6.15 (Linear maps)**.** Consider a vector space $\mathbf{V}$, which we assume includes a choice of basis. The copy map is the rectangular matrix made of two identity matrices:

$$\Delta_{\mathbf{V}} : \mathbf{V} \to \mathbf{V} \oplus \mathbf{V} := \begin{bmatrix} \mathbf{1}_{\mathbf{V}} & \mathbf{1}_{\mathbf{V}} \end{bmatrix}$$

The delete map is the column vector of 1s:

$$\epsilon_{\mathbf{V}} : \mathbf{V} \to \mathbf{0} := \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Example 0.6.16 (Sets and functions)**.** Consider a set A. The copy function is defined:

$$\Delta_A : A \to A \times A := a \mapsto (a, a)$$

The delete function is defined:

$$\epsilon_A : A \to \{\star\} := a \mapsto \star$$

Example 0.6.17 (Sets and relations)**.** Consider a set A. The copy relation is defined:

$$\Delta_A : A \nrightarrow A \times A := \{\big(a, (a, a)\big) \mid a \in A\}$$

The delete relation is defined:

$$\epsilon_A : A \nrightarrow \{\star\} := \{(a, \star) \mid a \in A\}$$

We may verify that, no matter the concrete interpretation of the diagram in terms of linear maps, functions

The two unique arrows between $\overset{K}{\times}$ and $\overset{W}{\times}$ are format-conversions, and we know by definition that the unique arrow that performs format conversion from $\overset{W}{\times}$ to itself in the bottom face is the identity. In maplet notation, the conversion from $A \overset{K}{\times} B \to A \overset{W}{\times} B$ is $\{\{a\}, \{a, b\}\} \mapsto \{\{a, \varnothing\}, b\}$, and similarly for the other direction. Because these conversions are uniquely determined arrows, their composite is also uniquely determined, and we know their composite is equal to the identity. So, the nontrivial conversions witness an *isomorphism* between $A \overset{K}{\times} B$ and $A \overset{W}{\times} W$; a pair of maps $X \to Y$ and $Y \to X$ such that their loop-composites equal identities. This in a nutshell is the category-theoretic approach to overcoming the bureaucracy of syntax: use universal properties (or whatever else) encode your intents and purposes, establish isomorphisms, and then treat isomorphic things as "the same for all intents and purposes". The idea of treating isomorphic objects as the same is ingrained in category theory, so isomorphism notation $\simeq$ is often just written as equality $=$; going forward we will use equality notation unless there are good reasons to remember that we only have isomorphisms.

Definition 0.6.3 (Functor)**.** A functor $F : \mathcal{C} \to \mathcal{D}$ (read: with domain a category $\mathcal{C}$ and codomain a category $\mathcal{D}$) consists of two suitably related functions. An object function $F_0 : \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{D})$ and a morphism function (equivalently viewed as a family of functions indexed by pairs of objects of $\mathcal{C}$) $F_1(X, Y) : \mathcal{C}(X, Y) \to \mathcal{D}(F_0X, F_0Y)$. F_1 must map identities to identities – i.e., be such that for all $X \in \mathcal{C}$, $F_1(1_X) = 1_{F_0X}$ – and F_1 must map composites to composites – i.e., for all $X, Y, Z \in \mathrm{Ob}(\mathcal{C})$ and all $f : X \to Y$ and $g : Y \to Z$, $F_1(f; g) = F_1 f; F_1 g$.

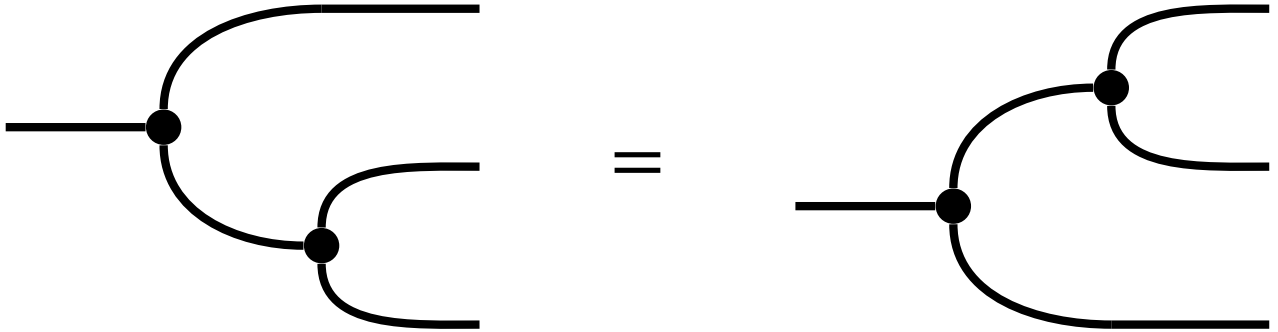
Functors in short map categories to categories, preserving the structure of identities and composition. They are the essence of "structure preserving transformation". Insofar as semantics is the science of finding structure-preserving transformations that tell us when syntactic things are equal, functors are just that. They are incredibly useful and mysterious and worth internalising in a way I am not adept enough to impress by example in this margin. For us, for now, they are just stepping stones to define transformations *between functors*.

Definition 0.6.4 (Natural Transformation)**.** A natural transformation $\theta : F \to G$ for (co)domain-aligned functors $F, G : \mathcal{C} \to \mathcal{D}$ is a family of morphisms in $\mathcal{D}$ indexed by objects $X \in \mathcal{C}$ such that for all $f : X \to Y$ in $\mathcal{C}$, the following commuting diagram holds in $\mathcal{D}$:

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ \downarrow{\scriptstyle \theta_X} & & \downarrow{\scriptstyle \theta_Y} \\ GX & \xrightarrow{Gf} & GY \end{array}$$

or relations, the following equations characterise a cocommutative comonoid internal to a monoidal category.

coassociativity



cocommutativity

counitality

It is worth pausing here to think about how one might characterise the process of copying in words; it is challenging to do so for such an intuitive process. The diagrammatic equations, when translated into prose, provide an answer.

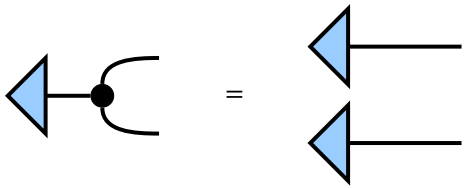
Coassociativity: says there is no difference between copying copies.

Cocommutativity: says there is no difference between the outputs of a copy process.

Counitality: says that if a copy is made and one of the copies is deleted, the remaining copy is the same as the original.

Insofar as we think this is an acceptable characterisation of copying, rather than specify concretely what a copy and delete does for each system X we encounter, we can instead posit that so long as we have processes $\Delta_X : X \otimes X \to X$ and $\epsilon_X : X \to I$ that obey all the equational constraints above, Δ_X and ϵ_X are as good as a copy and delete.

Example 0.6.18 (Not all states are copyable)**.** Call a state *copyable* when it satisfies the following diagrammatic equation:

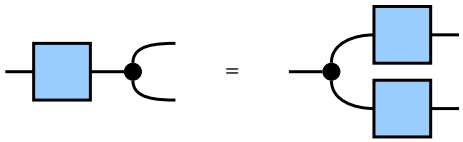


In the process theory of sets and functions, all states are copyable. Not all states are copyable in the process theories of sets and relations and of linear maps. For example, consider the two element set $\mathbb{B} := \{0, 1\}$, and let $\top : \{\star\} \nrightarrow \mathbb{B} := \{(\star, 0), (\star, 1)\} \simeq \{0, 1\}$. Consider the composite of $\top$ with the copy relation:

$$\top; \Delta_{\mathbb{B}} := \{\big(\star, (0, 0)\big), \big(\star, (1, 1)\big)\} \simeq \{(0, 0), (1, 1)\}$$

This is a perfectly correlated bipartite state, and it is not equal to $\{0, 1\} \times \{0, 1\}$, so $\top$ is not copyable.

Remark 0.6.19. The copyability of states is a special case of a more general form of interaction with the copy relation:

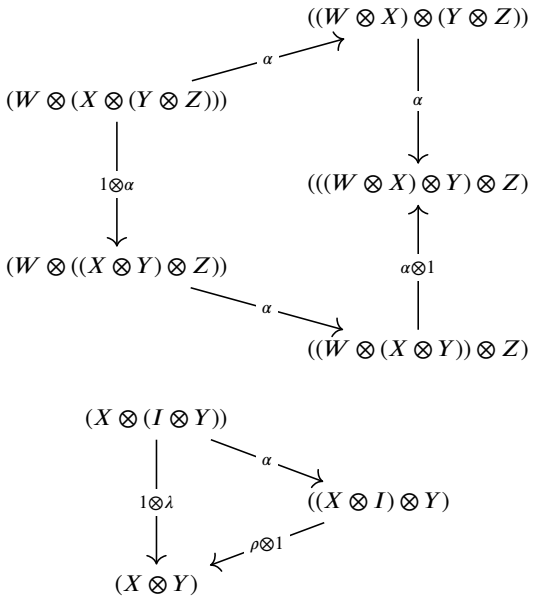


A cyan map that satisfies this equation is said to be a homomorphism with respect to the commutative comonoid. In the process theory of relations, those relations that satisfy this equation are precisely the functions; in other words, this diagrammatic equation expresses *determinism*.

Definition 0.6.5 (**Cat**)**.** **Cat** is a (2-)category where the objects are (1-)categories as defined above, the morphisms are functors, and the (2-)morphisms are natural transformations. (2-)morphisms are morphisms between morphisms that we discuss in more detail in Section **??**. There's no "set of all sets" paradox here by construction; **Cat** is slightly more than a category as we have seen so far because of the (2-)morphisms. We're introducing this just to state that the definition of product also works here so that we can consider product categories $\mathcal{C} \times \mathcal{D}$, whose objects are pairs of objects and morphisms pairs of morphisms.

Definition 0.6.6 (Monoidal Category)**.** A monoidal category consists of a category $\mathcal{C}$, a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, a monoidal unit object $I \in \mathrm{Ob}(\mathcal{C})$, and the following natural isomorphisms – i.e. natural transformations with inverses, where multiple bar notation indicates variable object argument positions: an associator $\alpha : ((- \otimes =) \otimes \equiv) \mapsto (- \otimes (= \otimes \equiv))$, a right unitor $\rho - \otimes I \mapsto -$, and a left unitor $\lambda I \otimes - \mapsto -\big)$. These natural isomorphisms must in addition satisfy certain *coherence* diagrams, to be displayed shortly.

Theorem 0.6.7 (Coherence for monoidal categories)**.** The following pentagon and triangle diagrams are conditions in the definition of a monoidal category. When they hold, all composites of associators and unitors (and their inverses) are isomorphisms . 1 denotes identities.



Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 0.6.18 and Remark 0.6.19, we know that this demand is incompatible with certain process theories. In particular, this demand would constrain a process theory of sets and relations to a subtheory of sets and functions. The moral here is that process theories are flexible enough to meet ontological needs. A classical computer scientist who works with perfectly copyable data and processes might demand universal copying along with Equations 1, whereas a quantum physicist who wishes to distinguish between copyable classical data and non-copyable quantum data might taxonomise copy and delete as a special case of a more generic quasi-copy and quasi-delete that only satisfies equations 1. In fact, quantum physicists *do* do this; see Dodo: [].

0.6.2 What is an update?

In the previous section we have seen how we can start with concrete examples of copying in distinct process theories, and obtain a generic characterisation of copying by finding diagrammatic equations copying satisfies in each concrete case. In this section, we show how to go in the opposite direction: we start by positing diagrammatic equations that characterise the operational behaviour of a particular process – such as *updating* – and it will turn out that any concrete process that satisfies the equational constraints we set out will *by our own definition* be an update.

Perhaps the most familiar setting for an update is a database. In a database, an entry often takes the form of pairs of fields and values. For example, where a database contains information about employees, a typical entry might look like:

```
< NAME:Jono Doe, AGE:69, JOB:CONTENT CREATOR, SALARY:$420, ...  >
```

There are all kinds of reasons one might wish to update the value of a field: Jono might legally change their name, a year might pass and Jono's age must be incremented, Jono might be promoted or demoted or get a raise and so on. It was the concern of database theorists to formalise and axiomatise the notion of updating the value of a field – *independently of the specific programming language implementation of a database* – so that they had reasoning tools to ensure program correctness []. The problem is reducible to axiomatising a *rewrite*: we can think of updating a value as first calculating the new value, then *putting* the new value in place of the old. Since often the new value depends in some way on the old value, we also need a procedure to *get* the current value. That was a flash-prehistory of *bidirectional transformations* []. Following the monoidal generalisation of lenses in [], a rewrite as we have described above is specified by system diagrammatic equations in the margin, each of which we introduce in prose.

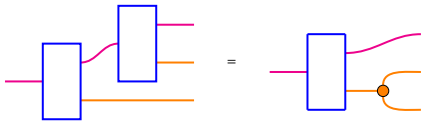
PutPut: Putting in one value and then a second is the same as deleting the first value and just putting in the

second.

GetPut: Getting a value from a field and putting it back in is the same as not doing anything.

PutGet: Putting in a value and getting a value from the field is the same as first copying the value, putting in one copy and keeping the second.

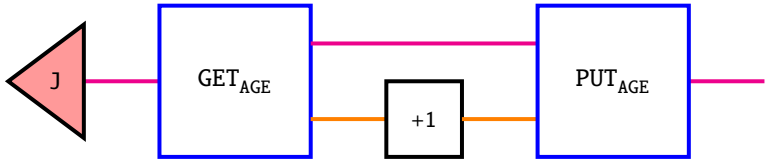
GetGet: Getting a value from a field twice is the same as getting the value once and copying it.



These diagrammatic equations do two things. First, they completely specify what it means to get and put values in a field in an implementation independent manner; it doesn't matter whether database entries are encoded as bitstrings, qubits, slips or paper or anything else, what matters is the interaction of get and put. Second, the diagrammatic equations give us the right to call our processes *get* and *put* in the first place: we define what it means to get and put by outlining the mutual interactions of get, put, copy, and delete. These two points are worth condensing and rephrasing:

A *kind of process* is determined by patterns of interaction with other kinds of processes.

Now we can diagrammatically depict the process of updating Jono's age, by getting Jono's age value from their entry, incrementing it by 1, and putting it back in.



0.6.3 *Pregroup diagrams and correlations*

Let's revisit the copy and delete maps for a moment. Suppose our process theory is such that for every wire X, there is a unique copy map δ_X such that every state on X is copyable and deletable. A consequence of this assumption is that every state is $\otimes$-separable (read *tensor separable*):

placeholder

But there are certainly process theories in which we don't want this. For example, if states are random variables and parallel composition is the product of independent random variables (as is the case in Markov categories for probability theory []) then copying a random variable gives a perfectly correlated pair of variables, which cannot be expressed as the product of a pair of independent random variables.

0.6.4 *Equational Constraints and Frobenius Algebras*

0.6.5 *Processes, Sets, and Computers*

Objection: But what are the *things* that the processes operate on?

This is a common objection from philosophers who want their ontologies tidy. The claim roughly goes that you can't really reason about processes without knowing the underlying objects that participate on those processes, and since set theory is the only way we know how to spell out objects intensionally in this way, we should stick to sets. In simpler terms, if we're drawing only functions as (black)-boxes in our diagrams, how will we know what they do to the elements of the underlying sets?

The short answer is that – perhaps surprisingly – reasoning process-theoretically is mathematically equivalent to reasoning about sets and elements for all practical purposes; it is as if whatever is going on *out there* is indifferent to whether we describe using a language of only nouns or only verbs.

In the case of set theory, let's suppose that instead of encoding functions as sets, we treat functions as primitive, so that we have a process theory where wires are labelled with sets, and functions are process boxes that we draw. The problem we face now is that it is not immediately clear how we would access the elements of any set using only the diagrammatic language. The solution is the observation that the elements $\{x \mid X\}$ of a set X are in bijective correspondence with the functions from a singleton into X: $\{f(\star) \mapsto$

$x \mid \{\star\} \xrightarrow{f} X$. In prose, for any element x in a set X, we can find a function that behaves as a pointer to that element $\{\star\} \to X$. So the states we have been drawing, when interpreted in the category of sets and function, are precisely elements of the sets that label their output wires.

The full and formal answer will require the reader to see Section **??** which spells out the category theory underpinning process theories. The caveat here is that process theories work for all *practical* purposes, so I make no promises about how diagrams work for the kind of set theories that deals with hierarchies of infinities that set theorists do. For other issues concerning for instance the set of all functions between two sets, that requires symmetric monoidal closure, for which there exist string-diagrammatic formalisms [].

OBJECTION: BUT IF THEY'RE EXPRESSIVELY THE SAME, WHAT'S THE POINT?

The following rebuttal draws on Harold Abelson's introductory lecture to computer science [] (in which string diagrams appear to introduce programs without being explicitly named as such).

There is a distinction between declarative and imperative knowledge. Declarative knowledge is *knowing-that*, for example, 6 is the square root of 36, which we might write $6 = \sqrt{36}$. Imperative knowledge is *knowing-how*, for example, to obtain the square root of a positive number, for instance, by Heron's iterative method: to obtain the square root of Y, make a guess X, and take the average of X and $\frac{Y}{X}$ until your guess is good enough.

Computer science concerns imperative knowledge. An obstacle to the study of imperative knowledge is complexity, which computer scientists manage by black-box abstraction – suppressing irrelevant details, so that for instance once a square root procedure is defined, the reasoner outside the system does not need to know whether the procedure inside is an iterative method by Heron or Newton, only that it works and has certain properties. These black-boxes can be then composed into larger processes and procedures within human cognitive load.

Abstraction also yields generality. For example, in the case of addition, it is not only numbers we may care to add, but perhaps vectors, or the waveforms of signals. So there is an abstract notion of addition which we concretely instantiate for different domains that share a common interface; we may decide for example that all binary operations that are commutative monoids are valid candidates for what it means to be an addition operation.

In this light, string diagrams are a natural metalanguage for the study of imperative knowledge; string diagrams in fact independently evolved within computer science from flowcharts describing processes. Process theories, which are equations or logical sentences about processes, allow us to reason declaratively about imperative knowledge. Moreover, string diagrams as syntactic objects can be interpreted in various concrete settings, so that the same diagram serves as the common interface for a process like addition, with compliant implementation details for each particular domain spelled out separately.

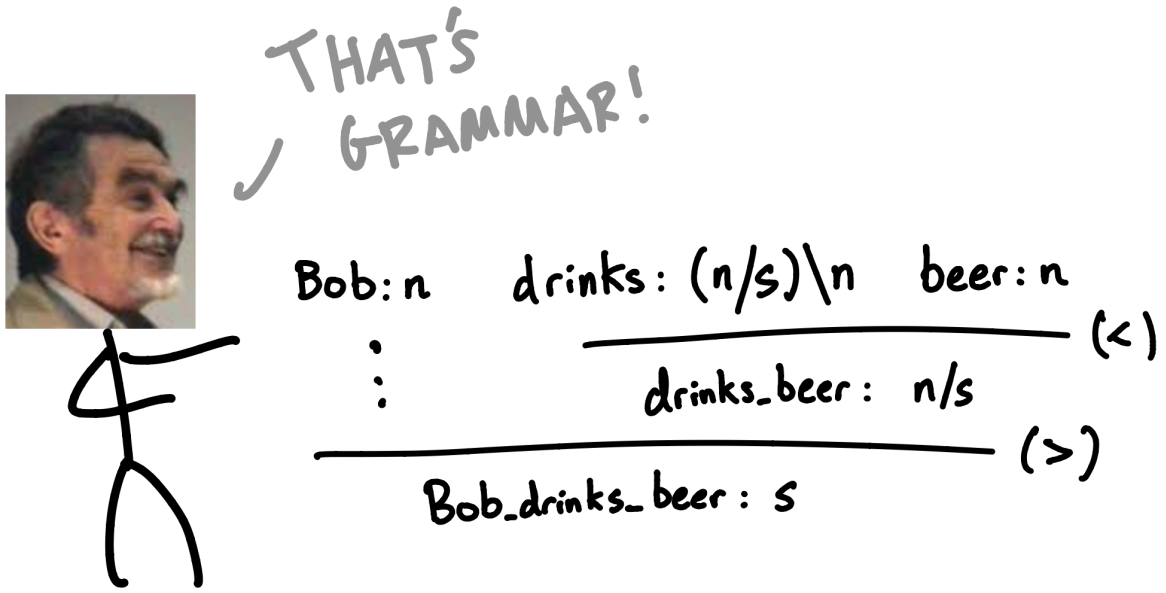
0.7 *Previously, on DisCoCat*

DisCoCat is a research programme in applied mathematical linguistics that is **Dis**tributional, **Co**mpositional and **Cat**egorical. In this section I will recount a selective development of DisCoCat as relevant for this thesis.

0.7.1 *Lambek's Linguistics*

It's hard for me to do justice to Jim Lambek's life. I feel as if have been in intimate conversation with Jim throughout my research, despite our separation by time. Anyone can look up the Curry-Howard-Lambek correspondence and follow the rabbit hole to see Jim's broad reach and lasting impact on category theory. I know that he was a jovial man who always carried a good sense of humour and a wad of twenties. I also can't do better than Moortgat's history and exposition of typelogical grammar in CITE , so I will borrow Moortgat's phrasing and summarise Lambek's role in the story. Typelogical grammar originated in two seminal papers by Lambek in 1958 and 1961 CITE , where Lambek sought "to obtain an effective rule (or algorithm) for distinguishing sentences from non-sentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages [. . .]". The method is to assign grammatical categories – parts of speech such as nouns and verbs – logical formulae. Whether a sentence is grammatical or not is obtained from deduction using these logical formulae in a Gentzen-style sequent proof.

Figure 17: In English, we may consider a noun to have type n, and a transitive verb $(n/s) \setminus n$, to yield a well-formedness proof of `Bob drinks beer`. The type formation rules for such a grammar are intuitive. Apart from a stock of basic types $\mathbb{B}$ that contains special final types to indicate sentences, we have two type formation operators $(-/=)$ and $(-\setminus=)$, which along with their elimination rules establish a requirement that grammatical categories require other grammatical categories to their left or right. This is the essence of Lambek's calculi **NL** and **L**. CCGs keep the same minimal type-formations, but include extra sequent rules such as type-raising and cross-composition.



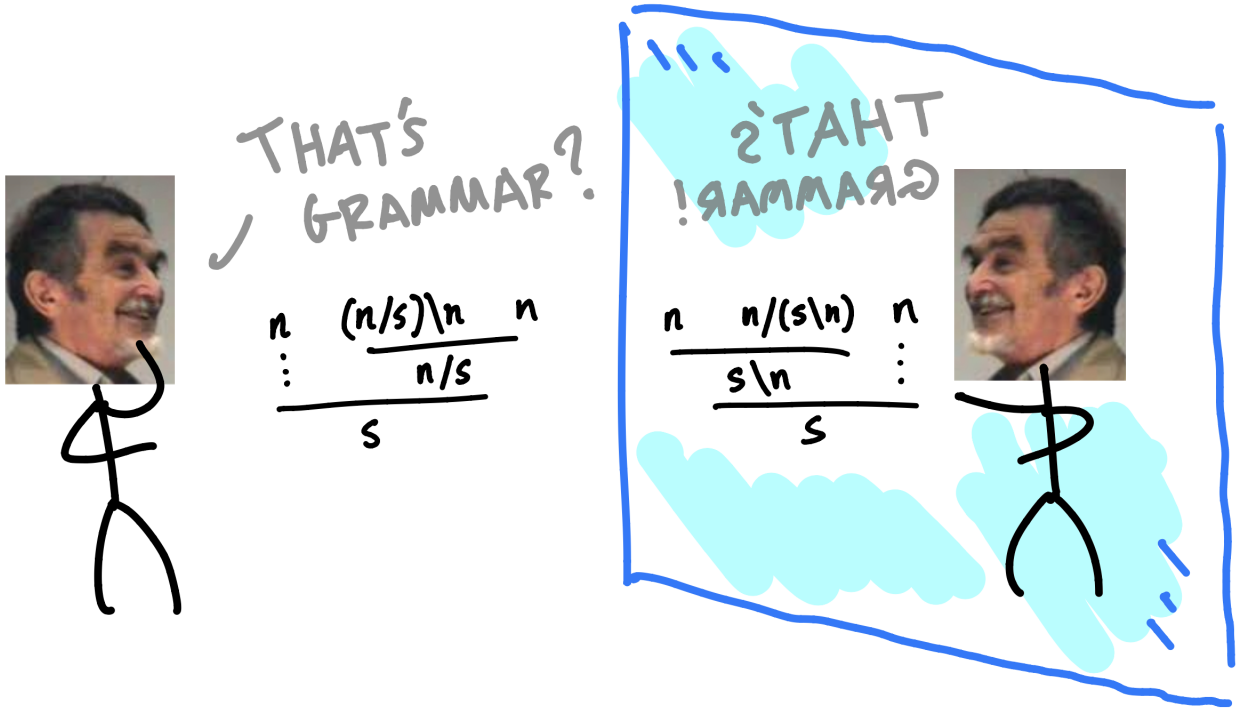
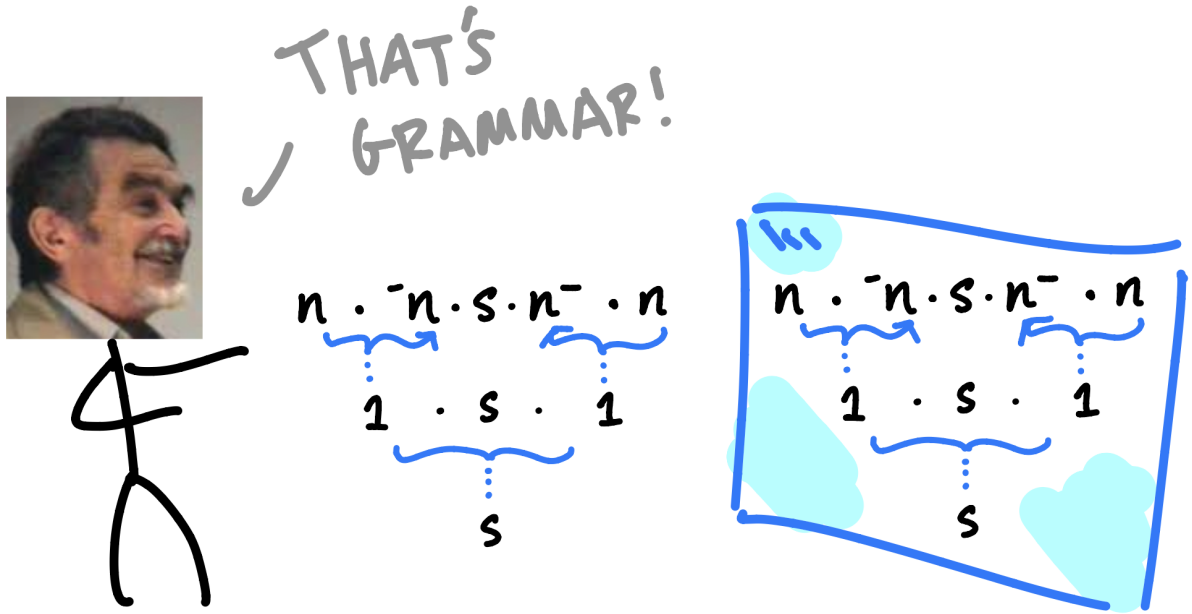


Figure 18: We can notice an asymmetry in the above formulation when we examine the transitive verb type $(n/s) \setminus n$ again; it asks first for a noun to the right, and then a noun to the left. We could just as well have asked for the nouns in the other order with the typing $(n/s) \setminus n$ and obtained all of the same proofs.

Figure 19: To eliminate this asymmetry, Lambek devised pregroup grammars. Whereas a group is a monoid with inverses up to left- and right-multiplication, a pregroup weakens the requirement for inverses so that all elements have distinct left- and right- inverses, denoted x^{-1} and ${}^{-1}x$ respectively. Eliminating or introducing inverses is a non-identity relation on elements of the pregroup, so we have axioms of the form e.g. $x \cdot^{-1} x \to 1 \to^{1} x \cdot x$. In this formulation, denoting the multiplication with a dot, both $(n/s) \setminus n$ and $(n/s) \setminus n$ become ${}^{-1}n \cdot s \cdot n^{-1}$, which just wants a noun to the left and a noun to the right in whatever order to eliminate the flanking inverses to reveal the embedded sentence type. Now we can obtain the same proof of correctness as a series of algebraic reductions.

$$
\begin{aligned}
& n \cdot ({}^{-1}n \cdot s \cdot n^{-1}) \cdot n \\
\to \quad & (n \cdot^{-1} n) \cdot s \cdot (n^{-1} \cdot n) \\
\to \quad & 1 \cdot s \cdot 1 \\
\to \quad & s
\end{aligned}
$$



0.7.2 *Coecke's Composition*

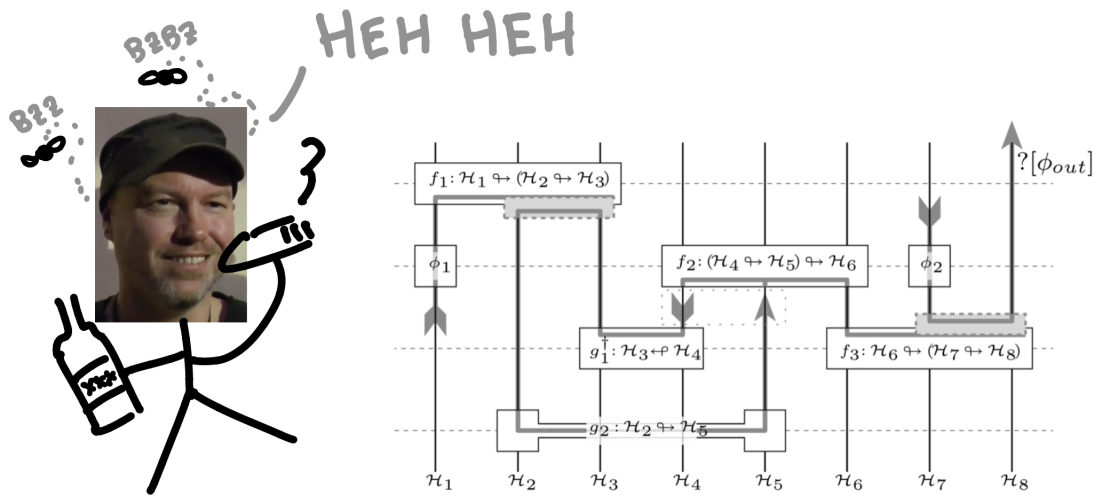


Figure 20: Meanwhile, an underground grunge vagabond moonlighting as a quantum physicist moonlighting as a computer scientist was causing a shortage of cigars and whiskey in a small English town. He noticed a funny thing about the composition of multiple non-destructive measurements of a quantum system, which was that information could be carried, or flow, between them. So he wrote a paper CITE , which contained informal diagrams that looked like this.

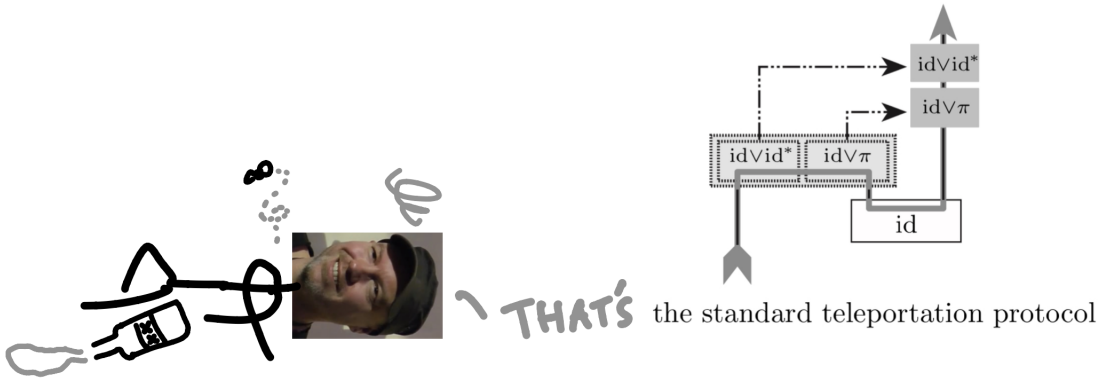


Figure 21: There were two impressive things about these diagrams. First, the effects such as transparencies for text boxes and curved serifs for angled arrows give a modern feel, but they were done manually in macdraw, the diagrammatic equivalent of sticks and stones. Second, though the diagrams were informal, they provided a way to visualise and reason about entanglement that was impossible by staring at the equivalent matrix formulation of the same composite operator. The most important diagram for our story was this one, which captures the information flow of quantum teleportation.

0.7.3 *Categorical quantum mechanics*

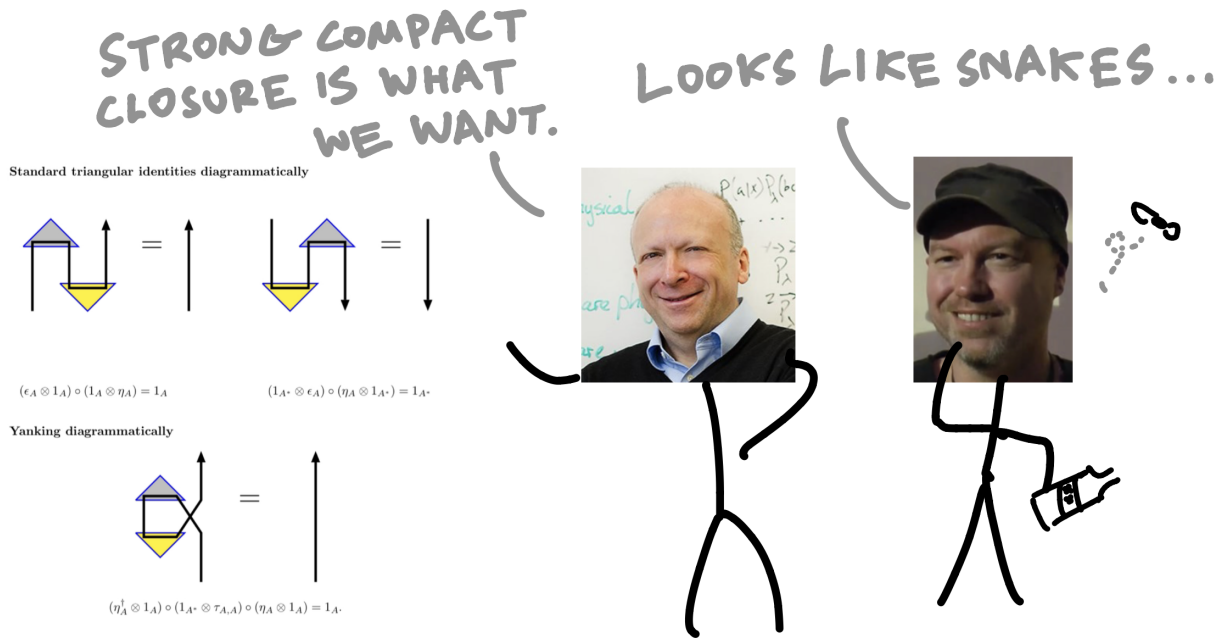


Figure 22: Category theorists and physicists such as Abramsky and Baez were excited about these diagrams, which looked like string diagrams waiting to be made formal. The graphical cups and caps in the important diagram were determined to correspond to a special form of symmetric monoidal closed category called strong compact closed.

Figure 23: Diagrammatically, reasoning in a strongly compact closed category amounts to ignoring the usual requirement of processiveness and forgetting the distinction between inputs and outputs, so that "future" outputs could curl back and be "past" inputs. This formulation also gave insight into the structure of quantum mechanics. For example, the process-state duality of strong compact closure manifested as the Choi–Jamiołkowski isomorphism.

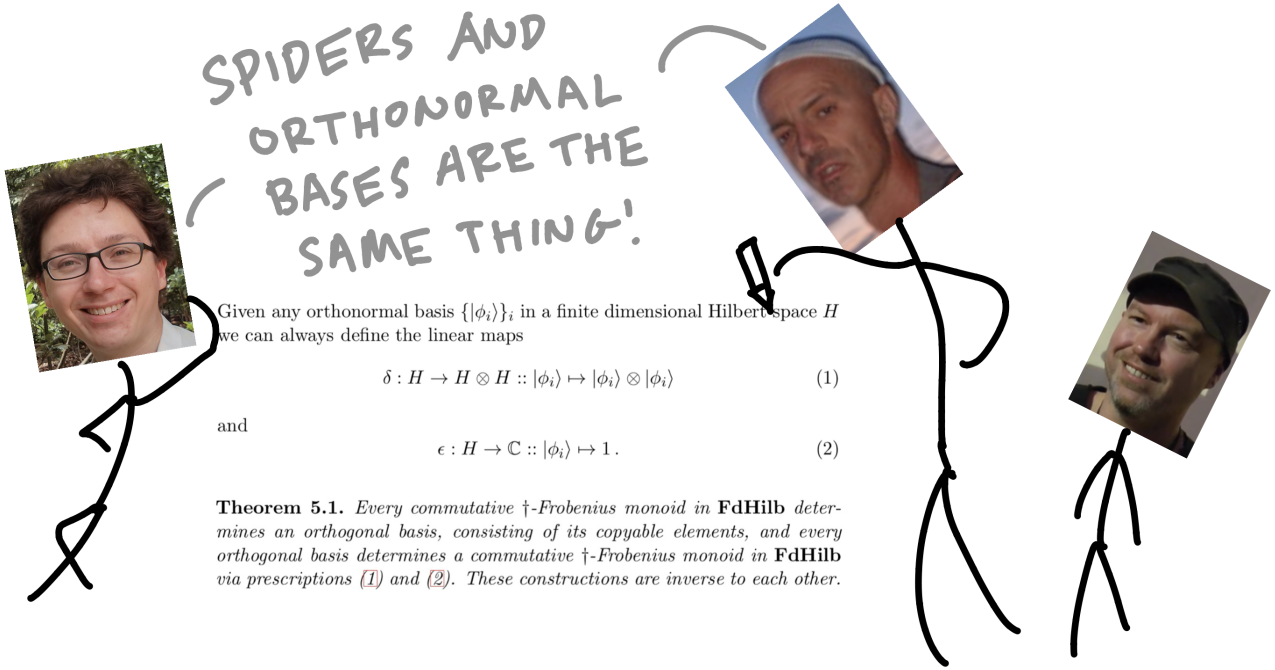


Figure 24: However, dealing with superpositions necessitated using summation operators within diagrams, which is cumbersome to write especially when dealing with even theoretically simple Bell states. An elegant diagrammatic simplification arose with the observation that special-†-frobenius algebras, or spiders, correspond to choices of orthonormal bases CITE in **FdHilb**, the ambient setting of finite-dimensional hilbert spaces.

classicalquantumstructuralism

Figure 25: Not only did this remove the need for summation operators, it also revealed that strong compact closure was a derived, rather than fundamental structure, since spiders induce compact closed structure.

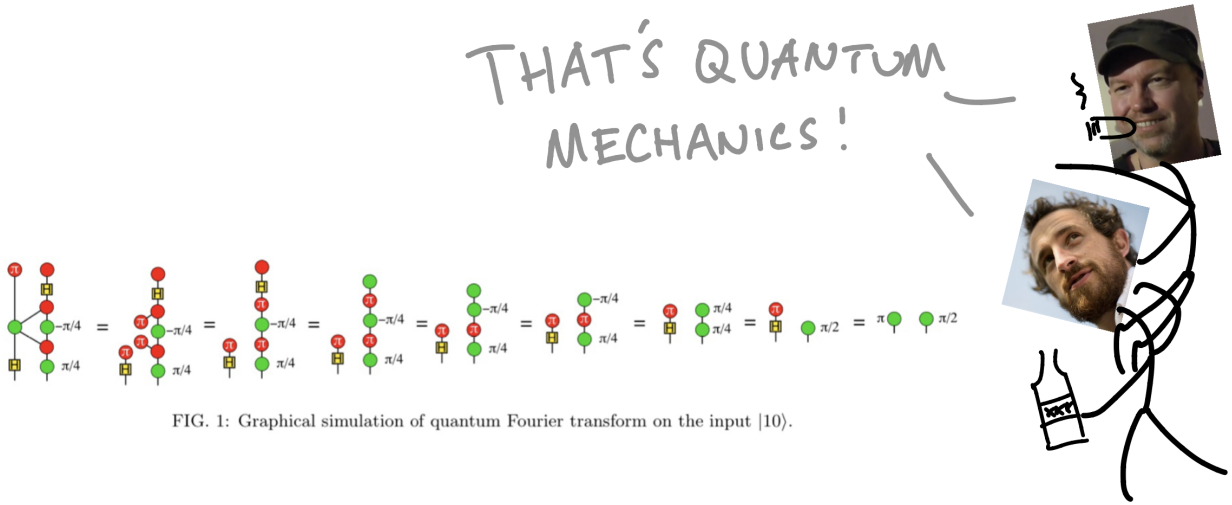


FIG. 1: Graphical simulation of quantum Fourier transform on the input $|10\rangle$.

Figure 26: And so the stage was set for a purely diagrammatic treatment of ZX quantum mechanics. The story of ZX diverges away from our interest, so I will summarise what happened afterwards. In no particular order, the development of ZX went on to accommodate a third axis of measurement to yield a ZXW calculus CITE , the systems were proven to be complete CITES , there are at the time of writing two expository books CITES , and ZX-variants are becoming an industry standard for quantum circuit specification and rewriting CITE .

0.7.4 *Enter computational linguistics*

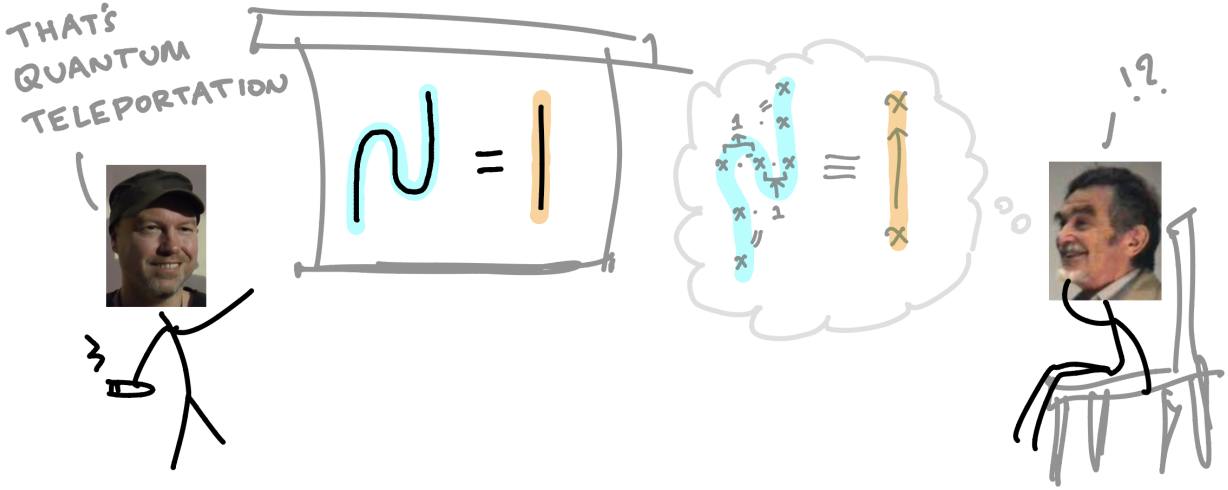


Figure 27: Somewhere in Canada at the turn of the millenium, Bob met Jim, who saw something familiar about the diagram for quantum teleportation. The snake equation for compact closure looked a lot like the categorified version of introducing and eliminating pregroup types.

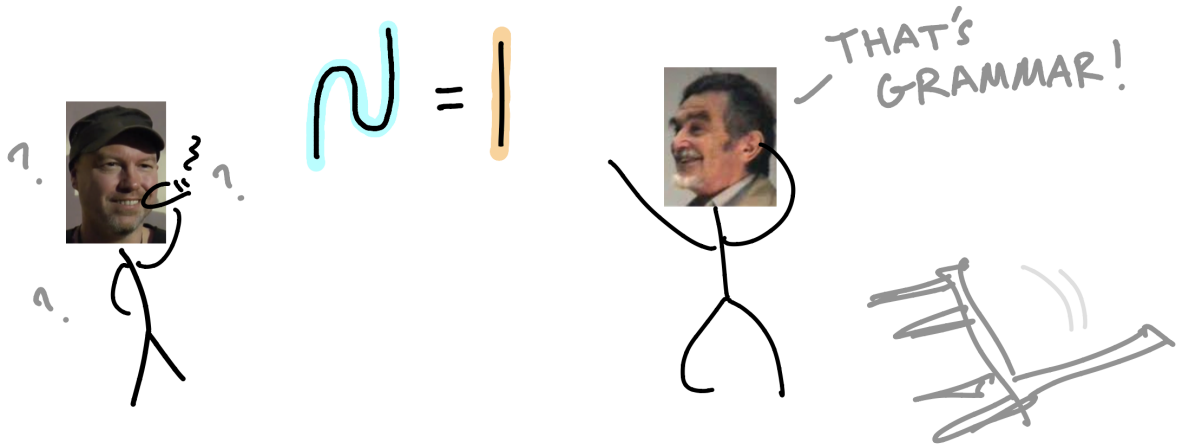


Figure 28: Bob and Jim's meeting put the adjectives *compositional* and *categorical* on the same table, but the cake wasn't ready. Two more actors Steve and Mehrnoosh were required to introduce *distributional*, which refers to Firth's maxim CITE "you shall know a word by the company it keeps". In its modern incarnation, this refers generally to vector-based semantics for words, where it is desirable but not necessarily so (as in the case of generic latent space embeddings by an autoencoder) that proximity of vectors models semantic closeness.

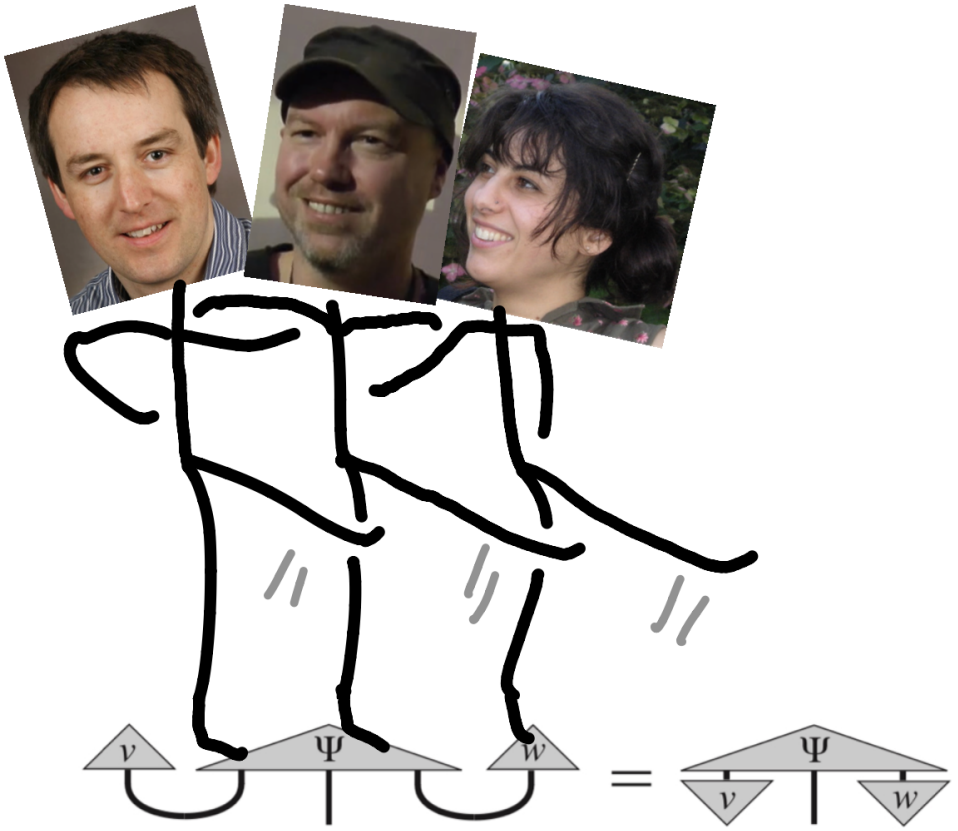


Figure 29: Steve Clark was a professor in the computer science department at Oxford, and he was wondering how to compose vector-based semantic representations. Steve asked Bob, who realised suddenly what Jim was talking about. Mediated by the linguistic expertise of Mehrnoosh who was a postdoctoral researcher in Oxford at the time, pregroup diagrams were born. The basic types n and s are assigned finite-dimensional vector spaces, concatenation of types the kronecker product $\otimes$, and by the isomorphism of dual spaces in finite dimensions there is no need to keep track of the left- and right- inverse data. Words become vectors, and pregroup reductions become bell-states, or bell-measurements, depending on whether one reads top-down or bottom-up. There was simply no other game in town for an approach to computational linguistics that combined linguistic compositionality with distributional representations.

where the reversed triangles are now the corresponding Dirac-bra's, or in vector space terms, the corresponding functionals in the dual space. This simplifies the expression that we need to compute to:

$$(\langle \overrightarrow{v} | \otimes 1_S \otimes \langle \overrightarrow{v} |)|\overrightarrow{\Psi}\rangle$$

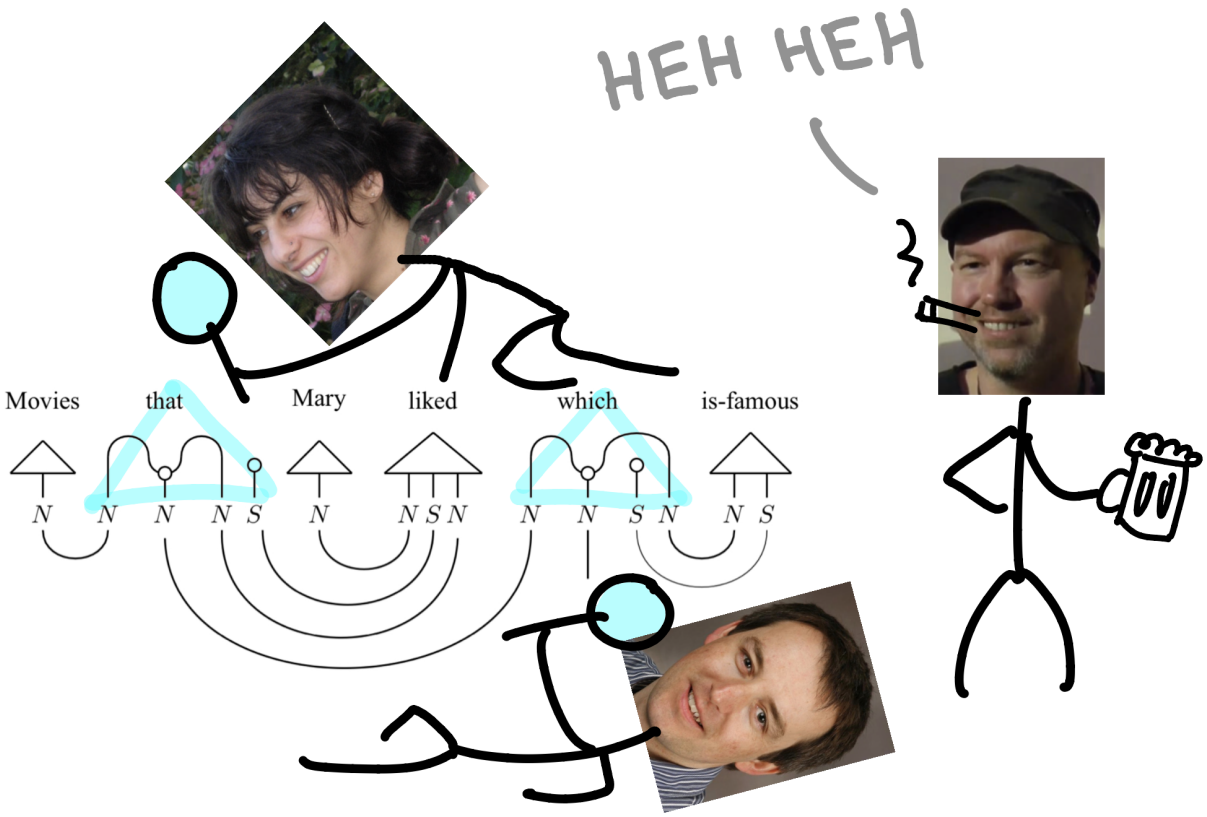


Figure 30: In *the frobenius anatomy of relative pronouns*CITE , the trio realised that spiders could play the role of relative pronouns, which was genuinely novel linguistics. If one follows the noun-wire of "movies", one sees that by declaring the relative pronoun to be a vector made up of a particular bunch of spiders-as-multiwires, "movies" is copied to be related to the "liked" word, copied again by "which" to be related to the "is-famous" word, and a third time to act as the noun in the whole noun-phrase. This discovery clarified a value proposition: insights from quantum theory could be applied in the linguistic setting, and linguistics offered a novel use-case for quantum computers. For example, density matrices were used to model semantic ambiguity CITE , and natural language experiments were performed on real quantum computeres CITE .

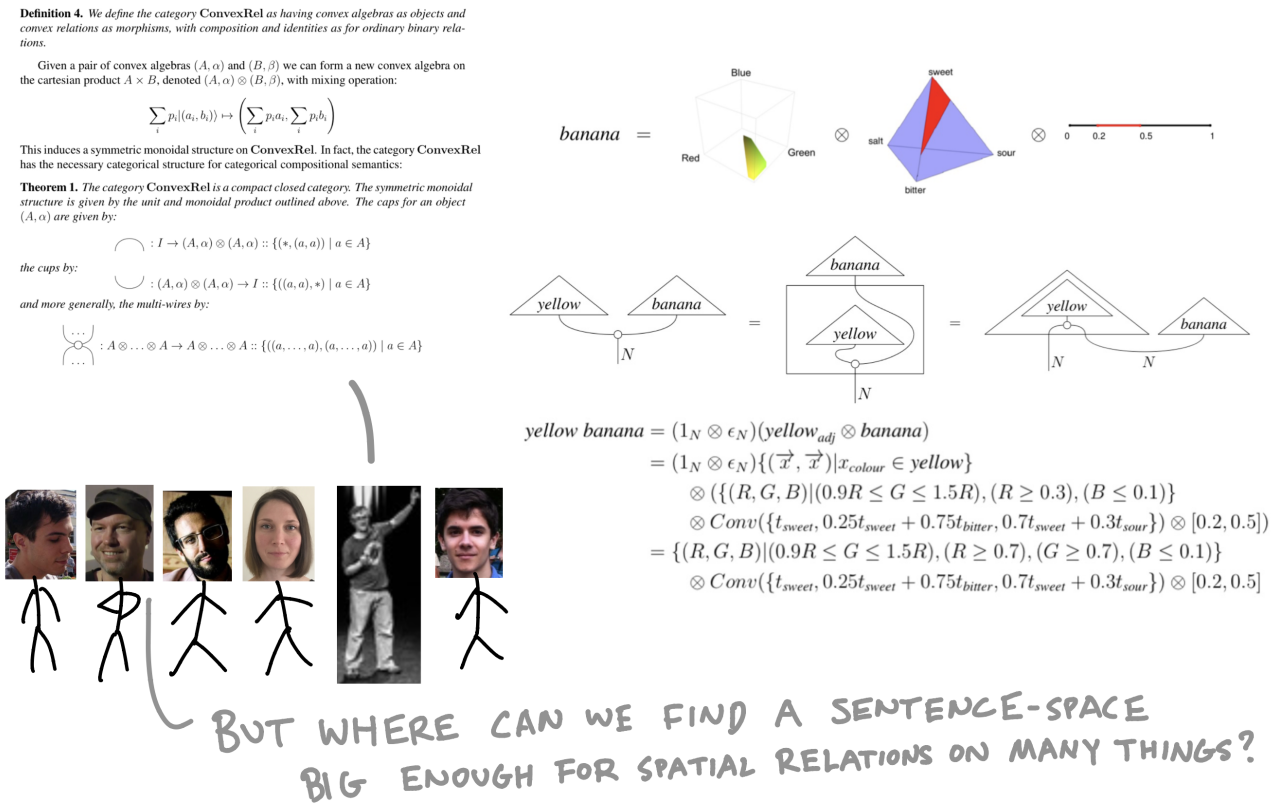


Figure 31: Keeping the structure of the diagrams but seeking set-relational rather than vector-based semantics, a bridge was made between linguistics and cognitive science in *interacting conceptual spaces I*CITE . Briefly, Gärdenfors posits that spatial representations of concepts mediate raw sense data and symbolic representations – e.g. red is a region in colourspace – and moreover that concepts ought to be spatially convex – e.g. mixing any two shades of red still gives red. This paper created a new point in the value proposition: that new mathematics would arise from investigating the linguistic-quantum bridge, e.g. generalised relations CITE . Although labelled as if it is the first in a series, the paper never saw a sequel by the same title, blocked by an apparently simple but actually tricky theoretical problem. The problem is that while this convex-relational story worked for conceptual adjectives modifying a single noun such for "sweet yellow bananas", there was difficulty in extending the story to work for multiple objects interacting in the same space, as in "cup on table in room". It couldn't be worked out what structure a sentence-wire in **ConvexRel** ought to have in order to accommodate (in principle) arbitrarily many objects and spatial relations between them.

DisCoCat then diverges from the story I want to tell. In no particular order, QNLP was done on an actual quantum computer CITE , some software packages were written CITE , and some art was made CITE .

0.7.5 *I killed DisCoCat, and I would do it again.*

Figure 32: It is a common evolutionary step in linguistics that theories 'break the sentential barrier', moving from sentence-restricted to text- or discourse-level analysis CITE . The same thing happened with DisCoCirc, due to a combination of practical constraints and theoretical ambition. On the practical side, wide tensors were (and remain) prohibitively expensive to simulate classically and actual quantum computers did not (and still do not) have many qubits, hence in practice pregroup diagrams were reduced to thinner and deeper circuits, often with the help of an additional simplifying assumption that sentence wires were pairs of noun wires in the illustrated form on the left. Theoretically, seeking dynamic epistemic logic, Bob had an epiphanous hangover (really) where he envisioned that these "Cartesian verbs" could be used in service of compositional text meanings, and he called this idea DisCoCirc CITE .

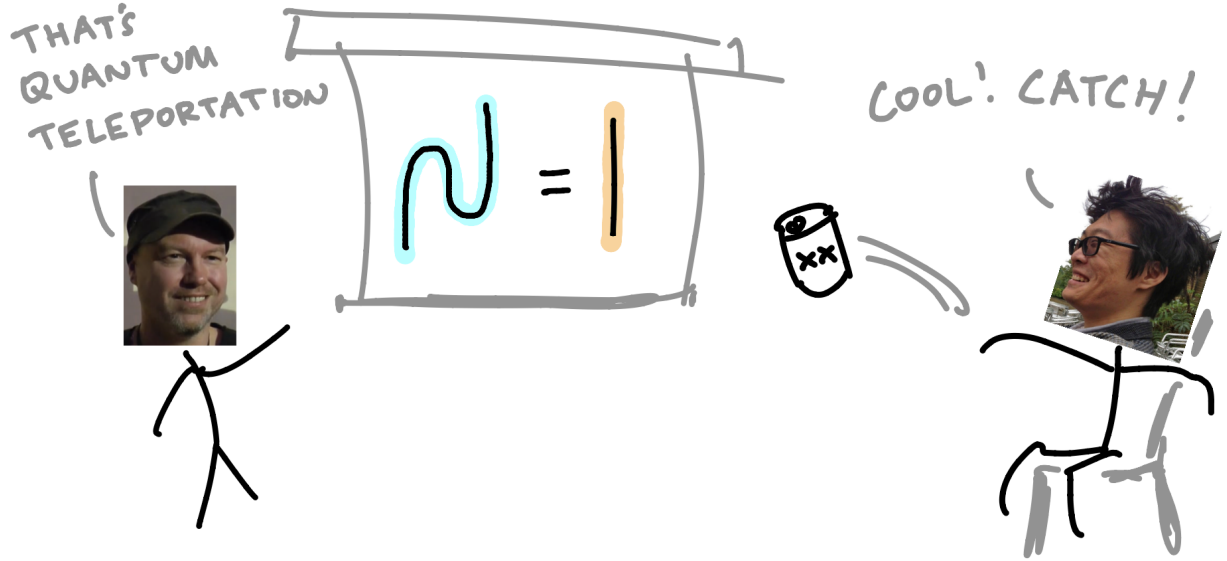


Figure 33: I met Bob in my master's in 2019, where he taught the picturing quantum processes course. When quantum teleportation was explained in half a minute by a diagram, I decided to pursue a DPhil in diagrammatic mathematics. In the last lecture, I threw Bob a cider, after which he seemed to like me. I did not know he was an alcoholic.

I was shanghaied into thinking about diagrams for language. I was deeply dissatisfied with the content from the standpoint my own intellectual integrity. Firstly, there seemed to me an unspoken claim that the presence of cups in pregroup diagrams (which implied a noncartesian and hence large tensor product) made it necessary to use quantum computers to effectively compute pregroup diagrams. I just could not believe that my brain required quantum computation to understand language. This implicit claim of kinship between quantum and linguistics was further entrenched by the analysis of the relative pronoun in terms of frobenius algebras, since spiders in $\mathbf{Vect}^{\otimes}$ were the *sine qua non* of categorical quantum mechanics. The best steelman for spiders I have is that frobenius algebras (which are central to bicategories of relations CITE) just happen to be a ubiquitous mathematical structure that are well-suited to express the mathematics of connections, both in language and in quantum.

Second, representing the content of a sentence as a vector in a sentence-vector-space did not sit well with me, since this move meant that the only meaningful thing one could do with two sentences was take their inner-product as a measure of similarity. Moreover, I had the theoretical concern that language is in principle indefinitely productive, so one could construct a sentence that marshalled indefinitely many nouns, and at some point for any finite vector space s one would run out of room to encode relationships, or else they would be cramped together in a way that did not suit intuitions about the freedom of constructing meanings using language. I always believed in the existence of a simple, practical, and intuitive categorical, compositional, and distributional semantics; I just didn't believe that the role of quantum – however helpful or interesting – was *necessary*.

My first unsatisfactory attempt was in my Master's thesis CITE . It had been known for a while that a free autonomous category construction by Delpeuch CITE could potentially eliminate some of the cups in pregroup diagrams, yielding what amounted to a method to transform a pregroup diagram into a monoidal string diagram in the shape of a context-free grammar tree. This trick had the limitation that freely adding directed cups and caps to a string diagrammatic signature did not turn a symmetric monoidal category into a (weakly) compact closed one, rather just into a monoidal category where the original wires had braidings, but all the new left and right dual wires did not; this presented difficulties in accounting for iterated duals for higher-order modifiers such as adverbs in grammatical types, and had nothing to say about spiders. I tried to generalise this trick to 'freely' adding arbitrary diagrammatic gadgets to string diagrams, but my assessor Samson pointed out that it was nontrivial to determine whether such constructions were faithful. In retrospect the free autonomous completion of a parameterised CITE markov category CITE is in the ballpark of dequantumfying pregroup diagrams, but I didn't learn about them until later, and that still wouldn't have addressed the issues that come with only having a sentence-wire.

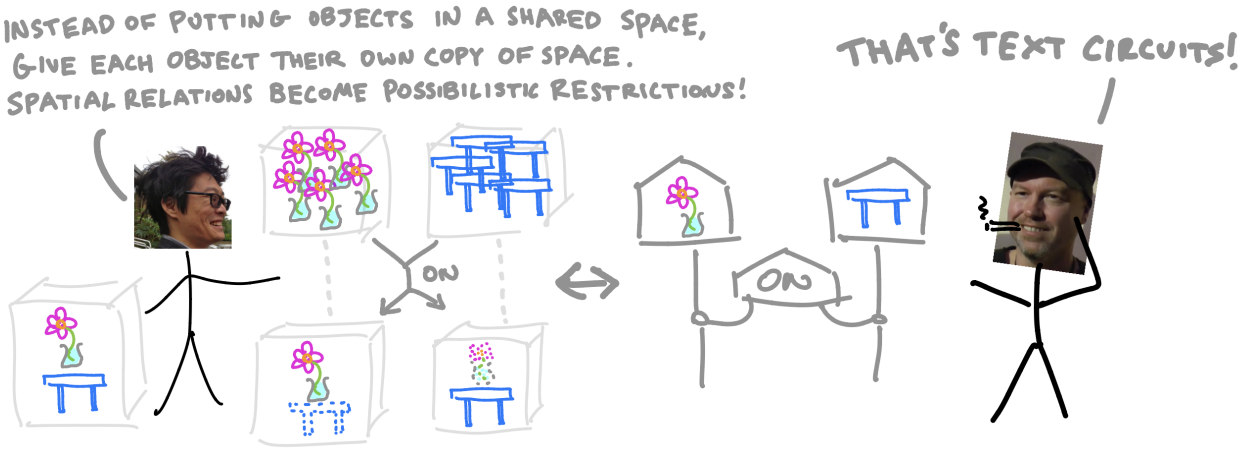


Figure 34: Then COVID happened. During the first lockdown, I visited Bob's garden under technically legal circumstances, and I suggested a solution to the longstanding problem of representing linguistic spatial relationships. My theoretical concern was the culprit: the initial attempts at the problem failed because the approach was to find a single sentence object s in which one could paste the data of arbitrarily many distinct spatial entities. The simple solution was a change in perspective.

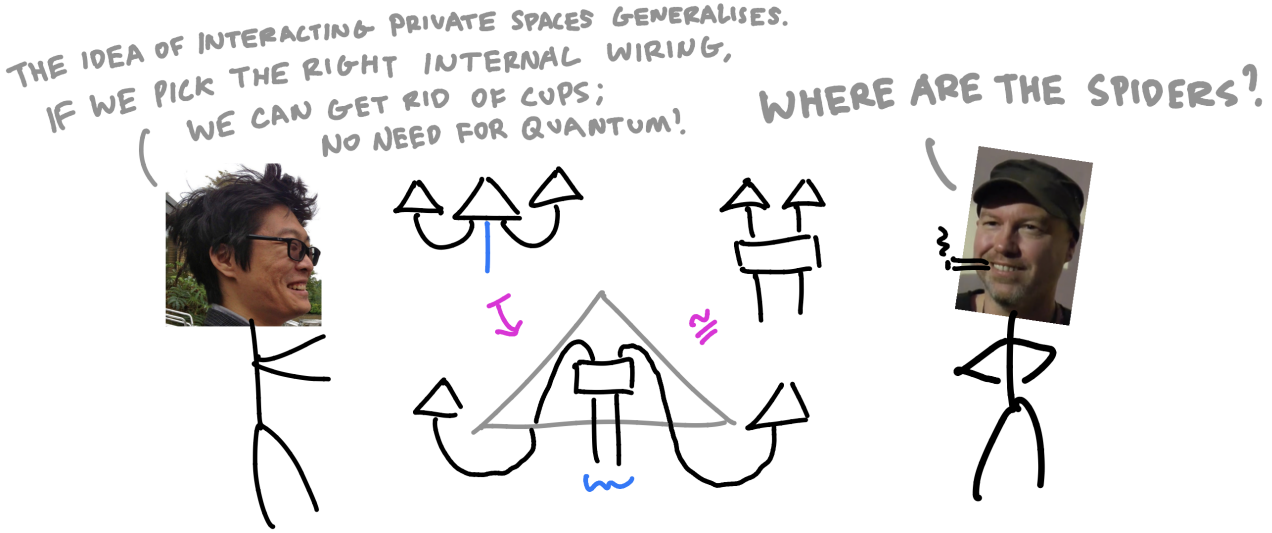


Figure 35: That this move of splitting up the sentence-wire into a sentence-dependent collection of wires was sufficient to solve what had appeared to be a difficult problem prompted some re-examination of foundations. The free autonomisation trick in conjunction with sentence-wire-as-tensored-nouns seemed promising, but it became clear that right way to drown a DisCoCat thoroughly was to explain and eliminate the spiders.

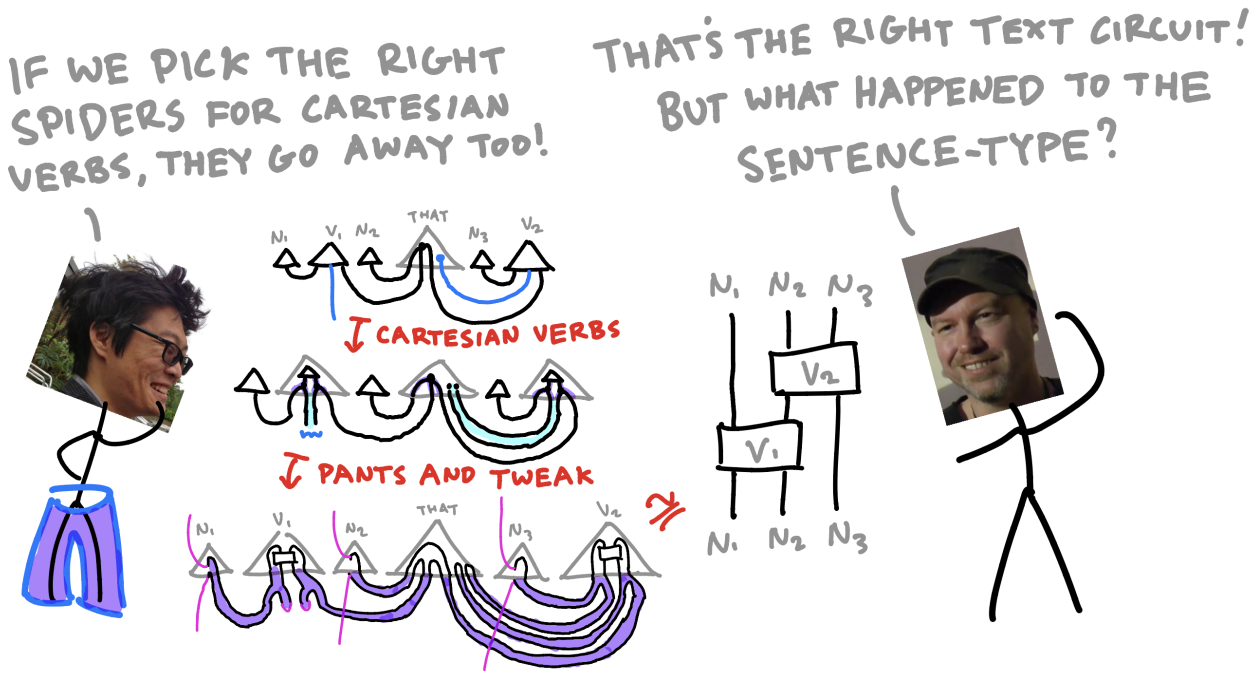


Figure 36: I then discovered that by interpreting spiders as the well-known "pair of pants" algebra in a compact closed monoidal setting allowed for a procedure in which the final form was purely symmetric monoidal – the absence of cups and caps meant that there was no practical necessity to interpret diagrams on quantum computers: any computer would suffice. The role of spiders for relative pronouns was illuminated in the presence of splitting the sentence wire: the pair-of-pants are the algebra of morphism composition, and splitting the sentence wire into a collection of nouns allowed relative-pronoun-spiders to pick out the participating nouns to compose relationships onto.

Figure 37: A coherent conservative generalisation of DisCoCat with less baggage had emerged, or rather, DisCoCirc was placed to formally subsume DisCoCat. It was now understood that the sentence type was a formal syntactic ansatz for the sake of grammar, which was to be interpreted in the semantic domain not as a single wire, but as a sentence-dependent collection of wires. It was further realised that the complexity of pregroup diagrams was due to grammar – the topological deformation of semantic connections to fit the one-dimensional line of language – whereas the essential connective content of language could be expressed in a simple form that distilled away the bureaucracy of syntax.

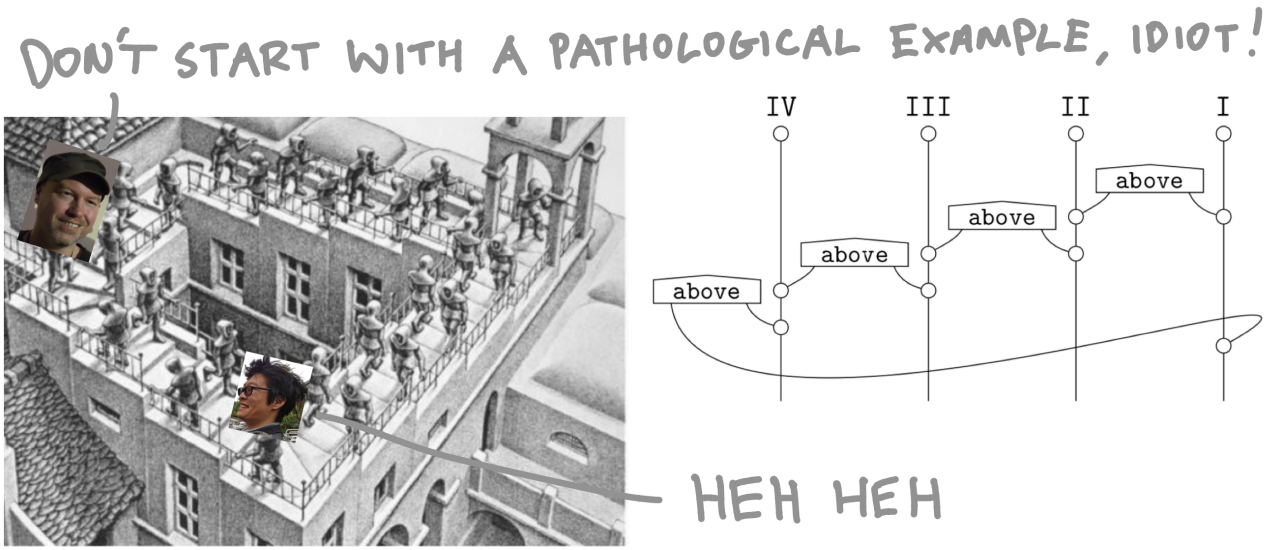


Figure 38: We wrote up the story about spaces in CITE , the spiritual successor to *interacting conceptual spaces I*. We could formally calculate the meanings of sentences that used linguistic spatial relations, all using a simple and tactile diagrammatic calculus.

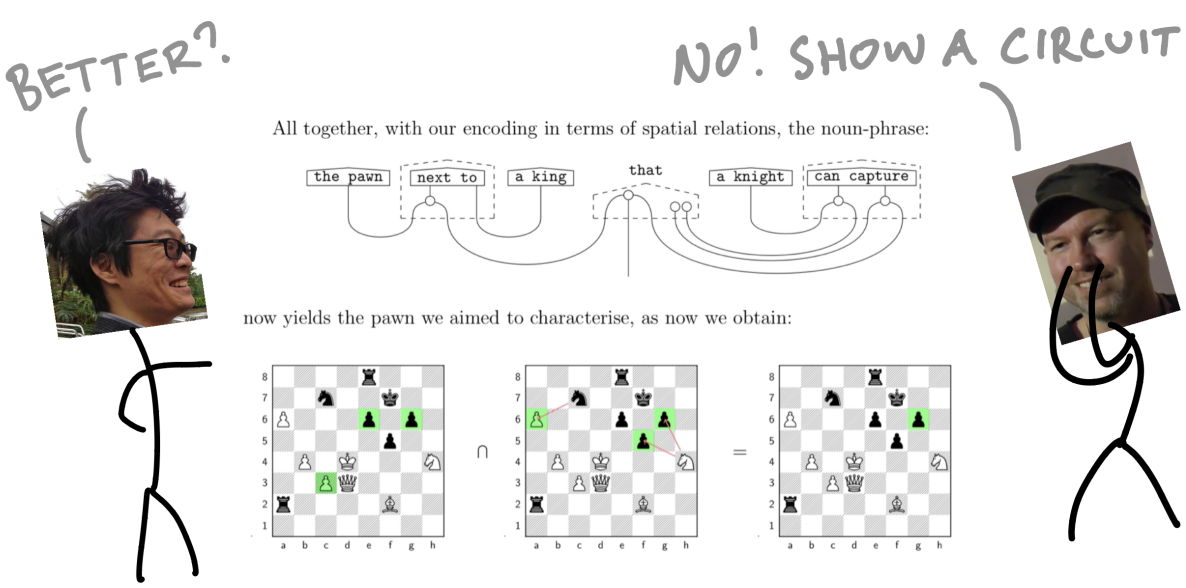


Figure 39: The paper on spatial relations actually came very late, because I was busy with Bob's ludicrous request to go turn "all of language" into circuits. I bitched and moaned about how I wasn't a linguist and how it was an impossible task, but I was in too deep to back out.

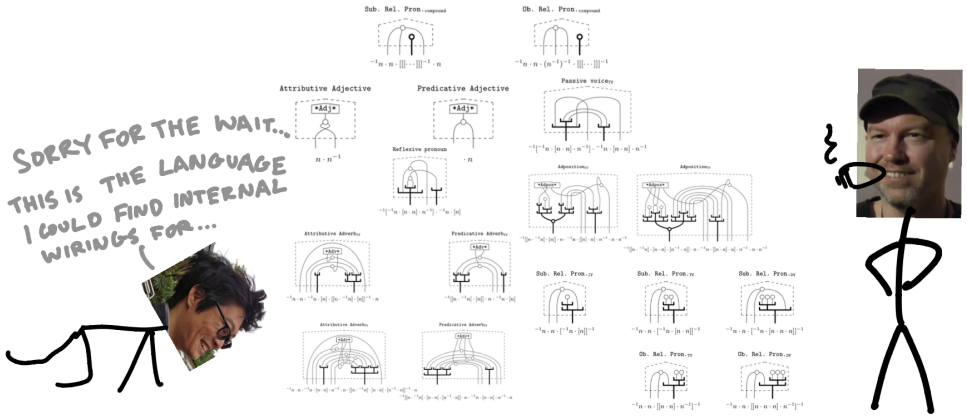
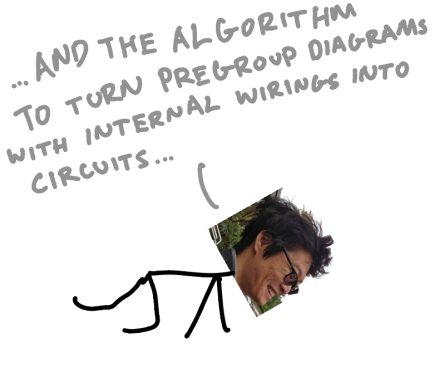


Figure 40: I suppose the nice thing about aiming for the moon is that even failure might mean you leave orbit. So I settled for what I thought was a sensible fragment of English, for which I devised internal wirings and an algorithm that transformed pregroup diagrams with the internal wirings into circuit form. Many tiring diagrams later, I presented my results in the first draft of "distilling text into circuits".



Algorithm 1: Grammar to Circuit Algorithm

Input: A sentence S
Result: A text circuit
Data: ***Parse***, ***PrnRes***, NOUNS, PRONOUNS
BASICWIRES $\leftarrow \{w \in S \cap \text{NOUNS}\}$; // 'BASICWIRES' is nouns of S
DIAGRAM $\leftarrow$ ***Parse***(S) ; // 'DIAGRAM' is pregroup diagram with wrapping
if *'DIAGRAM' has single output wire* o **then**
  Append appropriate deletion to o in DIAGRAM ; // delete output wire
  for $w \in$ *BASICWIRES* **do**
    Erase w's box from DIAGRAM, creating open wire ; // copy nouns
    Append rightmost output of a copy-spider to the open wire;
    Pull free input of copy-spider to the top of DIAGRAM;
    Pull free output of copy-spider to the bottom of DIAGRAM;
    Label this new wire with w;
  end
  while *There remain wrapped wires in DIAGRAM* **do**
    Apply unwraping from Definition 3 ; // unwrap
  end
  for $w_1, w_2 \in$ *BASICWIRES* **do**
    if $w_1 \in$ *PRONOUNS and* $w_2 \notin$ *PRONOUNS and* ***PrnRes***(w_1, w_2) **then**
      Attach outputs of a copy-spider to w_1, w_2 inputs;
      Attach inputs of upside-down copy-spider to w_1, w_2 outputs;
      relabel merged wire as w_2 ; // resolve pronouns
    end
  end
  while *There remain deletions, cups, or caps of non-wrapped wires* **do**
    Use spider-fusion to remove deletions, cups, and caps;
  end
  Simplify to obtain text circuit ; // clean up
  return DIAGRAM ; // return text circuit
else
  return False ; // (Fail if input ungrammatical)
end

Figure 41: Bob had a good point. Everything worked, but we had no understanding as to why, and accordingly, whether or not it would all break. At this point in time, Jonathon Liu, who was a masters' student I taught during COVID, had committed the error of thinking diagrams were cool, and was now hanging out with me and Bob. After understanding the procedure, Jono independently devised the same arcane internal wirings as I had, but neither of us could explain how we did it. So we had evidence of an underlying governing structure that was coherent but inarticulable.

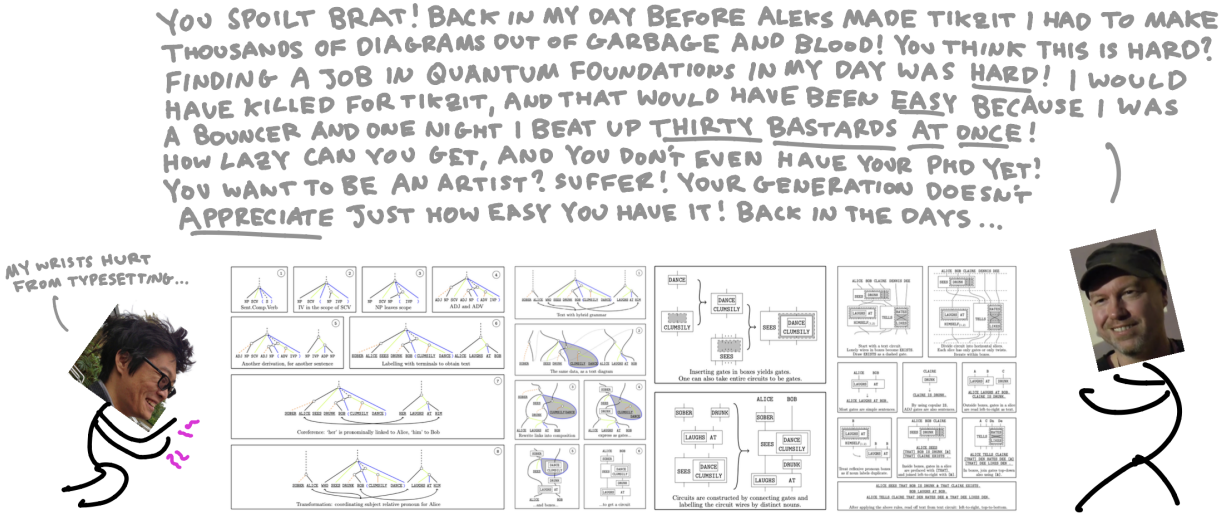


Figure 42: I realised that our intuitions were coming from an implicit productive grammar, rather than a parsing one, and that the path of least resistance for obtaining formal guarantees for the language-to-circuit procedure was to just handcraft a generative grammar for the fragment of language we were interested in. This meant scrapping everything in the first draft and starting again from scratch. Bob always had a word of gentle encouragement, giving me the motivation to persevere.

So now we had two ways to obtain text circuits. One from pregroups (which Jono had extended the technique for to CCGs in his master's thesis CITE), and one from handcrafted productive grammars. Then came time for me to write my thesis. Three salient questions arose. Firstly, what is the relationship between these two ways of getting at text circuits? Secondly, how do text circuits stand in relation to other generative grammars? Thirdly, what is it that text circuits allow us to do?

These questions are now what the rest of the thesis seeks to answer.

1

Internal wirings: what, why, and where from?

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

1.1 How do we communicate using language?

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

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

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

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

`Alice likes flowers that Bob gives Claire.`

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

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

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

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

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

Example 1.1.1 (`Alice likes flowers that Bob gives Claire.`)**.** Let's say Charlie is using a context-free grammar to produce sentences, and Dennis a pregroup grammar.

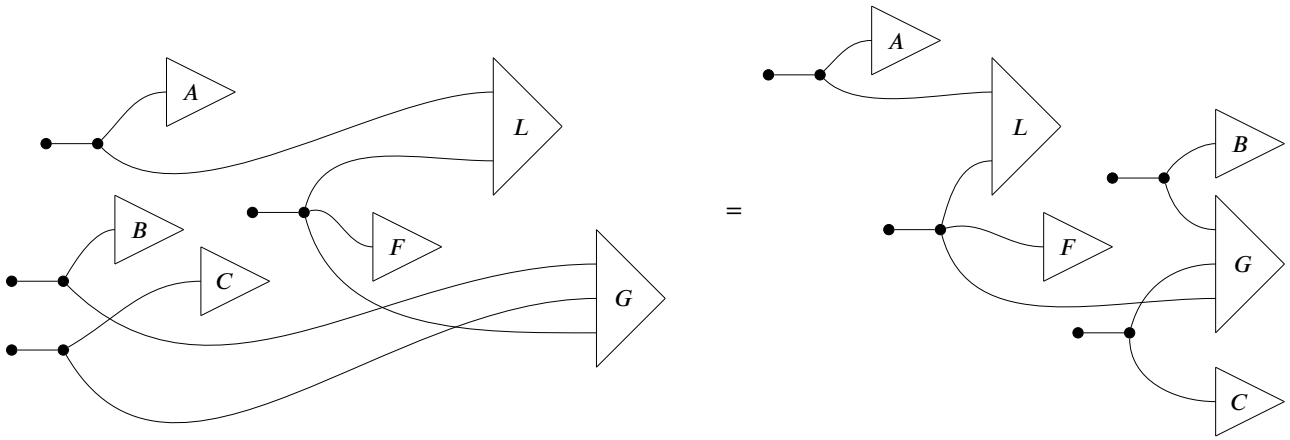


Figure 1.1: Charlie and Dennis agree on the conceptual organisation entities and relations up to the words for those entities and relations. Just as a running example that does not affect the point, let's say we can gloss a thought in first order logic as $\exists a \exists b \exists c \exists f \ : \ A(a) \wedge B(b) \wedge C(c) \wedge F(f) \wedge L(a,f) \wedge G(b,c,f)$. In diagrammatic first order logic [], this is equivalently presented as the following diagrams (and any other diagram that agrees up to connectivity.) For example, Charlie could ask Dennis comprehension questions such as `WHO GAVE WHAT? TO WHO?`, and if Dennis can always correctly answer – e.g. `BOB GAVE FLOWERS. TO CLAIRE.` – then both Charlie and Dennis agree on the relational structure of the communicated thought to the extent permitted by language.

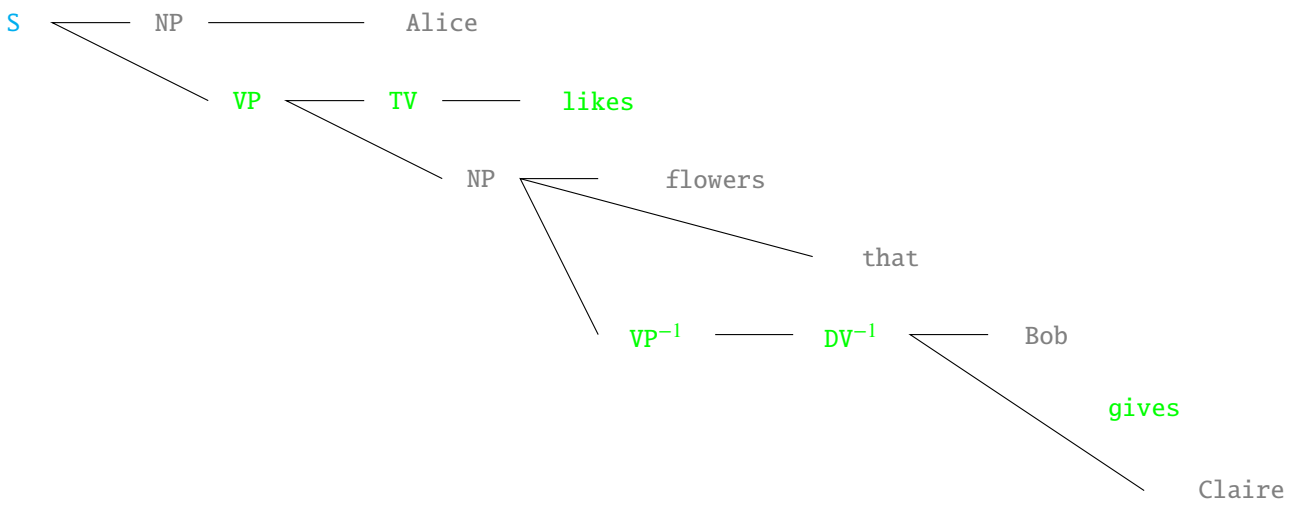


Figure 1.2: The rule of the game is that Charlie and Dennis can agree on a string-diagrammatic encoding strategy before having to communicate with each other. Here is one such strategy. Charlie might generate the example sentence as depicted.

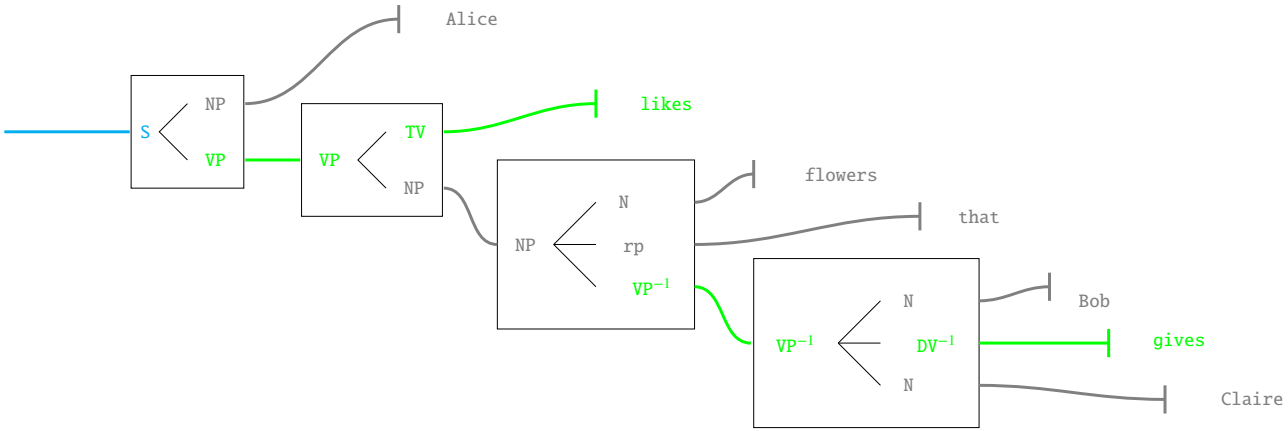


Figure 1.3: Mathematically, it makes no difference if we take the Poincaré dual of the tree, so that zero-dimensional nodes become one-dimensional wires, and branchings become zero-dimensional points linking wires – but we can just as well depict those points as boxes to label them more clearly.

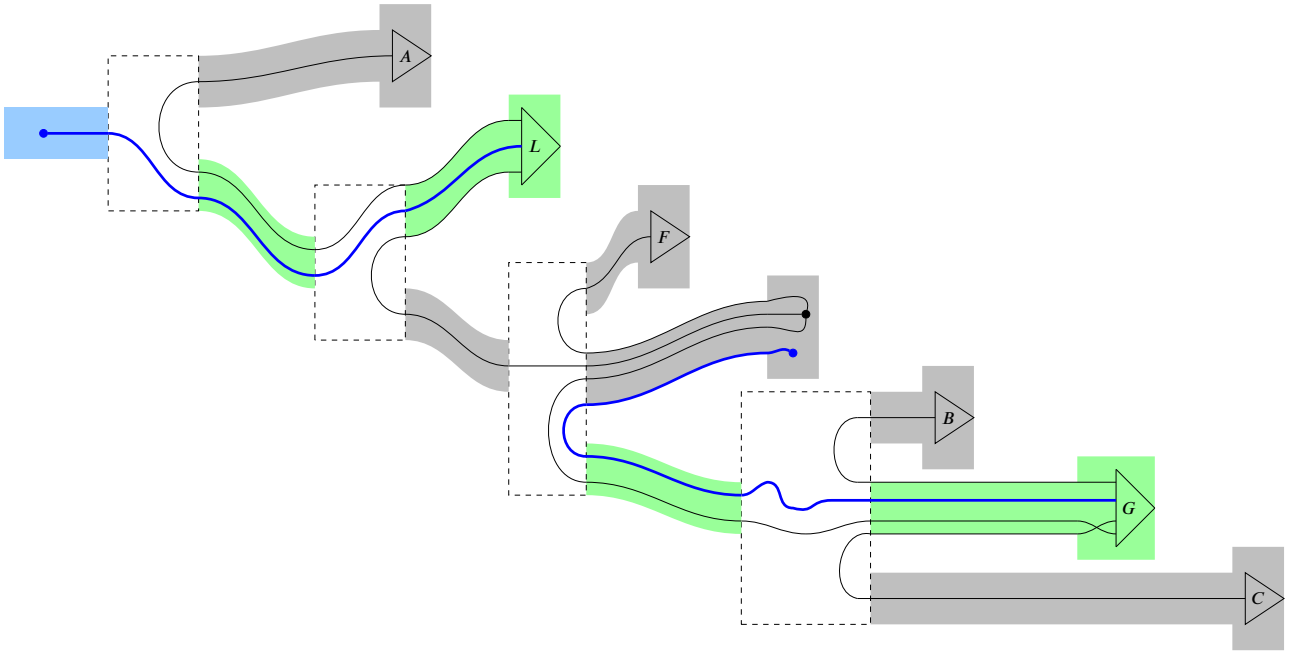


Figure 1.4: Now that Charlie can express their grammatical structure string-diagrammatically, they can try to deform their first-order-logic diagram – representing what they mean to communicate – subject to the constraint that every one of their branchings (the structure of the CFG) is something recoverable by Dennis using just pregroup reductions. To do so, Charlie introduces a formal blue wire to mimic Dennis's sentence-type, and stuffs some complexity inside the labels in the form of internal wirings: a multiwire configuration for `that`, and a twist for `gives`. Those internal wirings are the content of Charlie and Dennis's shared strategy. In passing, I'll remark that by the outside-in convention for functor boxes 1.13, this diagram constitutes a monoidal functor from this particular CFG to pregroup diagrams, where nonlabel tree-nodes are partial monoidal closure evaluators. Replacing rigid autonomous closure with cartesian closure and n, s with e, t recovers montague semantics for CFGs (c.f. Curry-Howard-Lambek correspondence for the case of typed lambda-calculus and cartesian closed categories, and all of Heim and Kratzer [HK98]), and interpreting the closure in a compact closed setting recovers montague semantics for CCGs [YK21].

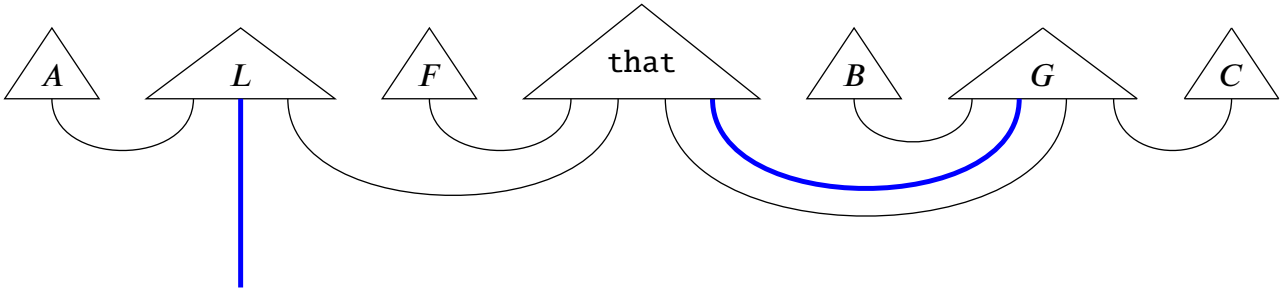


Figure 1.5: So, when Dennis receives the sentence, Dennis's pregroup derivation yields a pregroup diagram that is connectively equivalent to what Charlie stuffed inside the context-free grammar structure. So now the two have strong equivalence between their grammars in the sense that every one of Charlie's branches is resolved by one of Dennis's reductions. As is convention for pregroup diagrams, we only use types n and s – the latter denoted by a blue wire here – and we'll leave the directionality (rigid autonomous turning number) of wires implicit, so you can either trust me that everything typechecks or do it yourself.

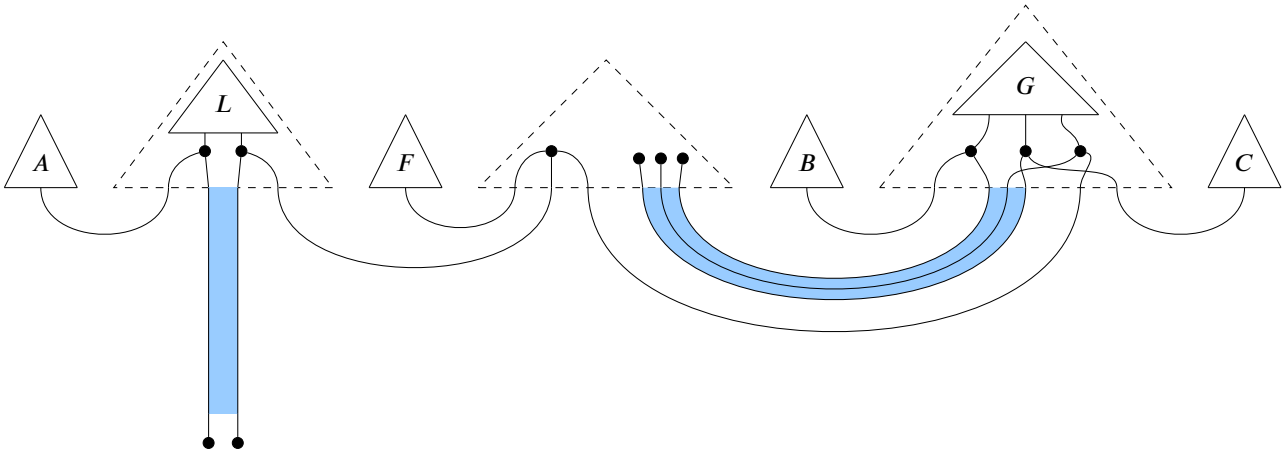


Figure 1.6: Now to fully recover Charlie's intended FOL-diagram, Dennis refers to the internal wirings from their shared strategy, and fills those in.

Example 1.1.2 (`Bob gives Claire flowers. Alice likes flowers.`)**.** Now we try the same content as the previous example but presented as a text with two sentences.

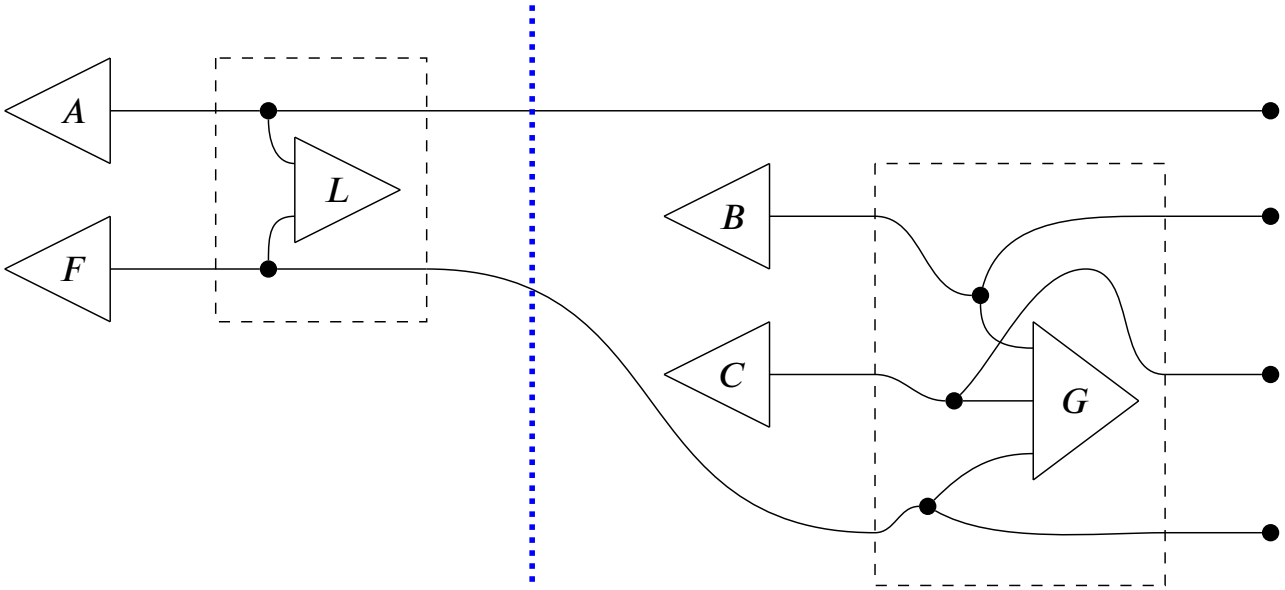


Figure 1.7: Charlie's diagram morphed to fit a text circuit. The dotted blue line is a formal mark to indicate a sentential boundary. Observe how new discourse elements are introduced as states, and how open wires correspond to ongoing discourse and deletions mark completed discourse. This diagram also indicates that text circuits can be given semantics in FOL.

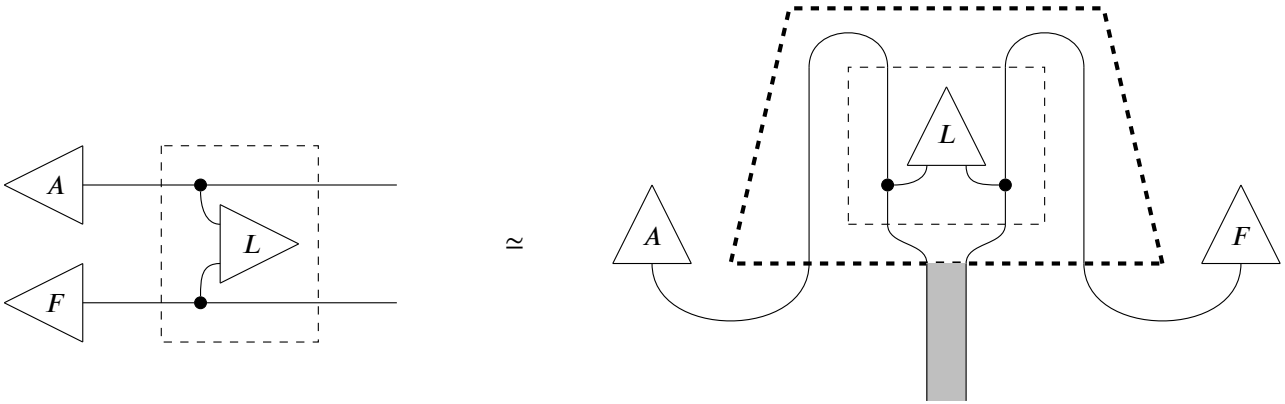


Figure 1.8: Dennis already knows how to parse individual sentences to extract the FOL using internal wirings. Observe there is a mathematical complication that arises in determining how many noun-wires should go into the sentence wire-bundle; we need to account for this later.

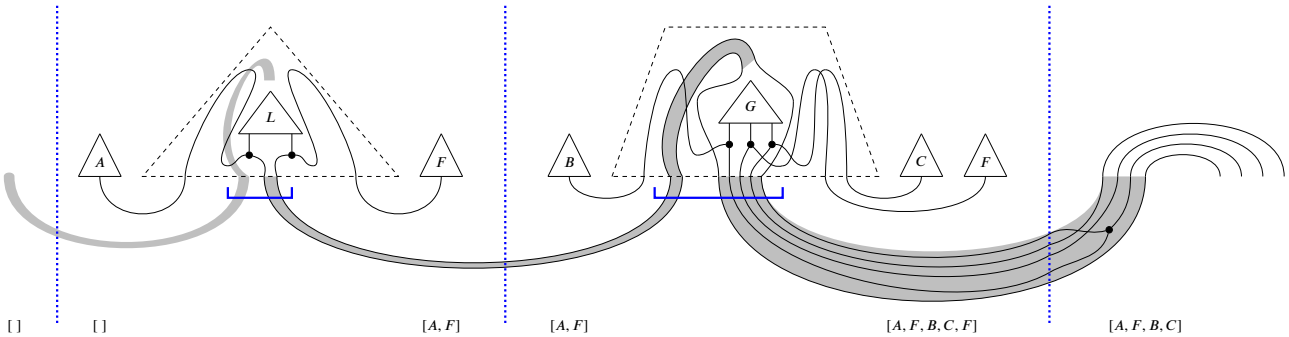


Figure 1.9: To deal with text, Dennis can pass a growing bundle of sentence wires along horizontally.

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

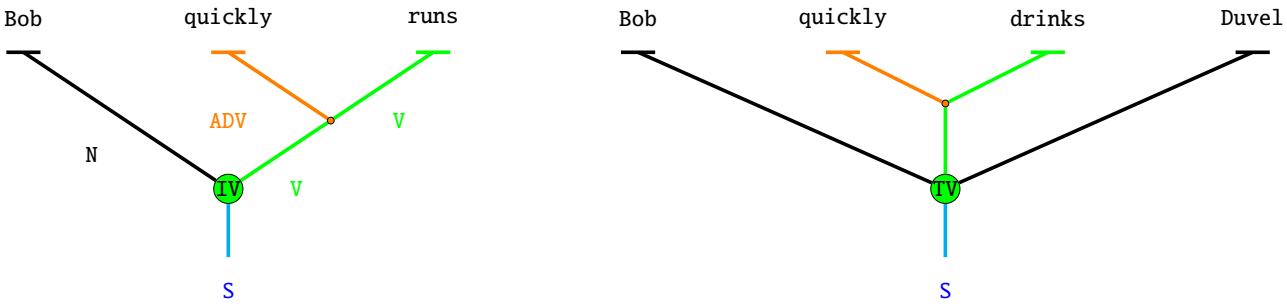
1.1.1 *An issue with functorial semantics of internal wirings*

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

Figure 1.10:

Example 1.1.3 (Nonfunctoriality of internal wirings for productive grammars)**.**

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



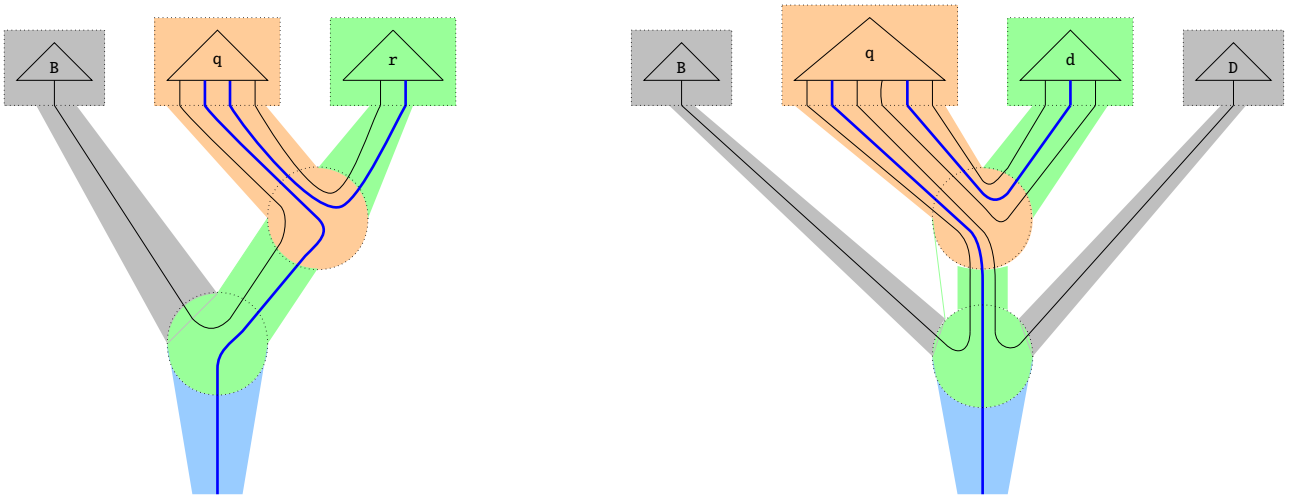


Figure 1.11: Now suppose we want to describe a functor from this context free grammar to a pregroup grammar with just types n and s. We know how verb states ought to look, and we know that adverbs ought to modify a verb. We can get pretty close with a first sketch, depicting the desired action of the functor using the outside-in convention for functor boxes, and we can slim them down to tubes. Now the simplicity of the CFG reveals a complication. Since there are two possible kinds of verbs, there are two possible kinds of adverbs, and accordingly two possible kinds of adverb introduction rules. A functor from the CFG to a pregroup diagram can't send the single adverb introduction rule to two different things at the same time.

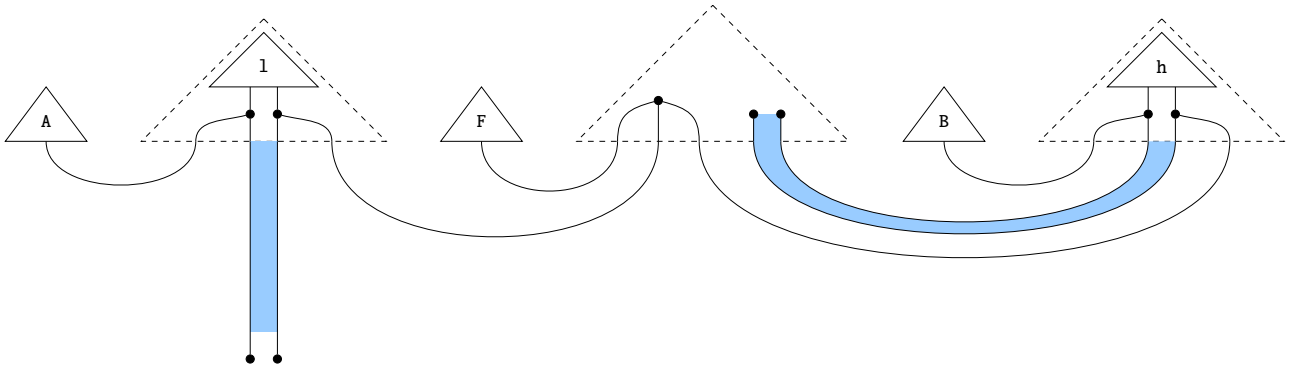


Figure 1.12:

Example 1.1.4 (Nonfunctoriality of internal wirings for parsing grammars)**.**

Compare `Alice likes flowers that Bob hates` to the sentence in Figure 1.6; here the object relative pronoun `that` is connected to a transitive verb `hates` rather than a ditransitive `gives`. The internal wirings work fine in this example, but now `that` deletes two wires instead of three; a functor can't map the same word-state to two possible instantiations.

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

1.2 *Discrete Monoidal Opfibrations*

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

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

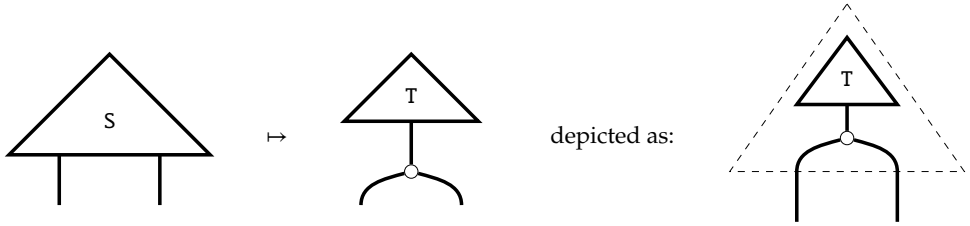


Figure 1.13: There are two conventions for depicting the action of a monoidal functor on parts of a string diagram. The first follows source-to-target *outside-in*. This convention is used for other work in internal wirings, since it is well-suited for describing functors that send atomic generators in their domain to more complex diagrams in their domain.

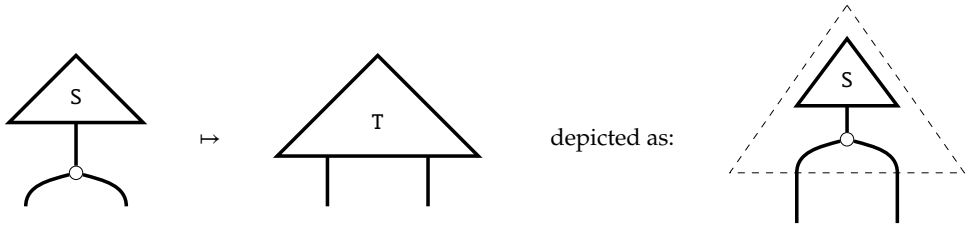


Figure 1.14: The other convention, following CITE , is *inside-out*. For the following section, we will define the coherence conditions of discrete monoidal fibrations using this convention.

Figure 1.15: Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \to \mathcal{D}$. Then we have this diagrammatic representation of a morphism $\mathbf{F}A \xrightarrow{\mathbf{F}f} \mathbf{F}B$ in $\mathcal{D}$.

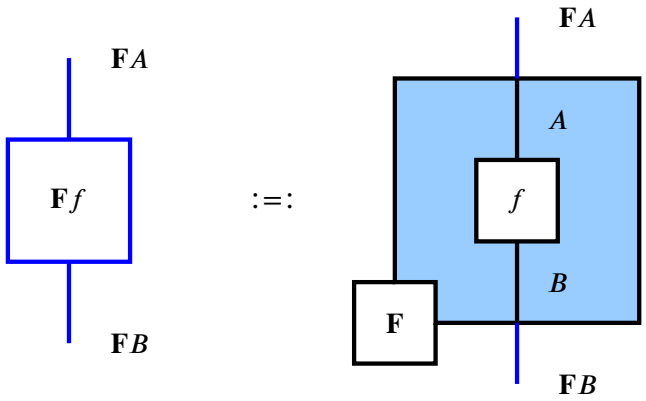


Figure 1.16: The use of a functor box is like a window from the target category $\mathcal{D}$ into the source category $\mathcal{C}$; when we know that a morphism in $\mathcal{D}$ is the image under $\mathbf{F}$ of some morphism in $\mathcal{C}$, the functor box notation is just a way of presenting all of that data at once. Since $\mathbf{F}$ is a functor, we must have that $\mathbf{F}f ; \mathbf{F}g = \mathbf{F}(f ; g)$. Diagrammatically this equation is represented by freely splitting and merging functor boxes vertically. **N.B.** sequential merging of two boxes requires that the two wires to-be-connected within the boxes – in this case labelled B – need to be the same; a case where merging is disallowed is when $Ff ; Fg$ typechecks in the outside/target category, but $f ; g$ does not in the inside/source category because the functor identifies nonequal wires.

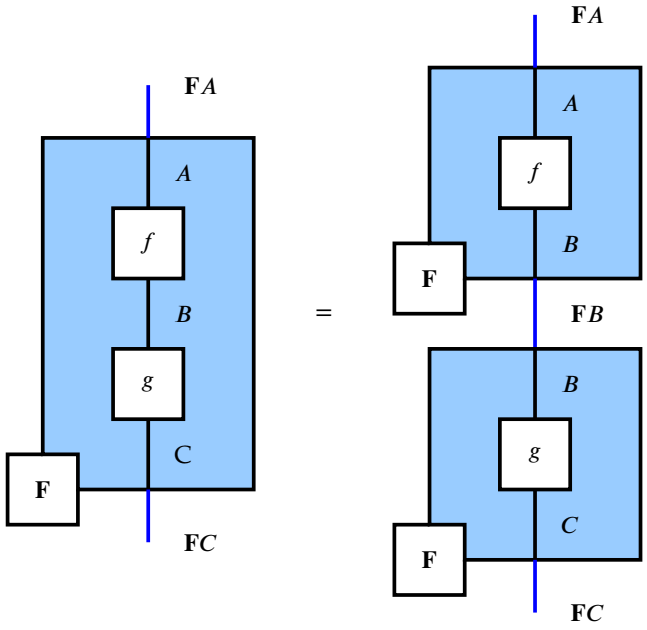


Figure 1.17: Assume that $\mathbf{F}$ is strict monoidal; without loss of generality by the strictification theorem CITE , this lets us gloss over the associators and unitors. For $\mathbf{F}$ to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A \otimes_D \mathbf{F}B = \mathbf{F}(A \otimes_C B)$. Diagrammatically these structural constraints amount to these equations.

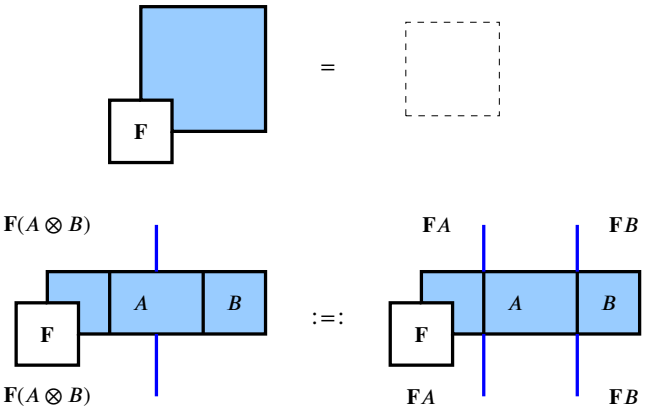


Figure 1.18: What remains is the monoidality of $\mathbf{F}$, which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes' way of respecting sequential composition, splitting horizontally is how they respect parallel composition.

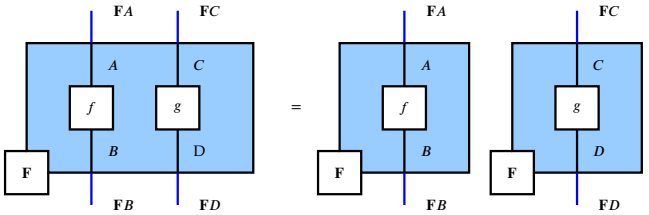


Figure 1.19: And for when we want $\mathbf{F}$ to be a (strict) symmetric monoidal functor, we are just asking that boxes and twists do not get stuck on one another.

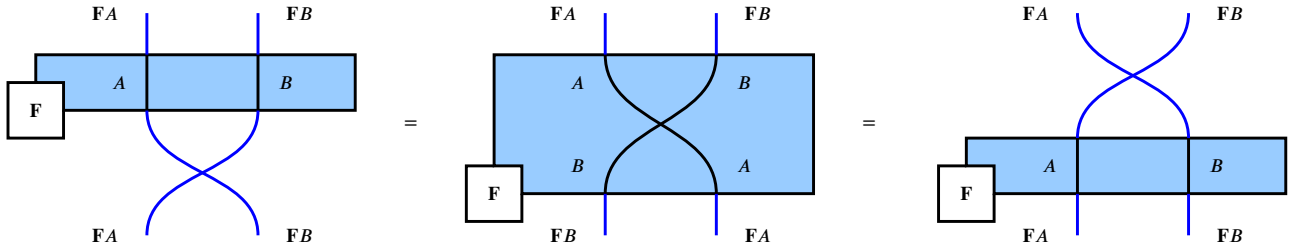


Figure 1.20: To motivate fibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a functor box out of the bottom. When can we do the reverse? That is, take a morphism in $\mathcal{D}$ and *slide it into* a functor box? We know that in general this is not possible, because not all morphisms in $\mathcal{D}$ may be in the image of $\mathbf{F}$. So instead we ask "under what circumstances" can we do this for a functor $\mathbf{F}$? The answer is when $\mathbf{F}$ is a discrete fibration.

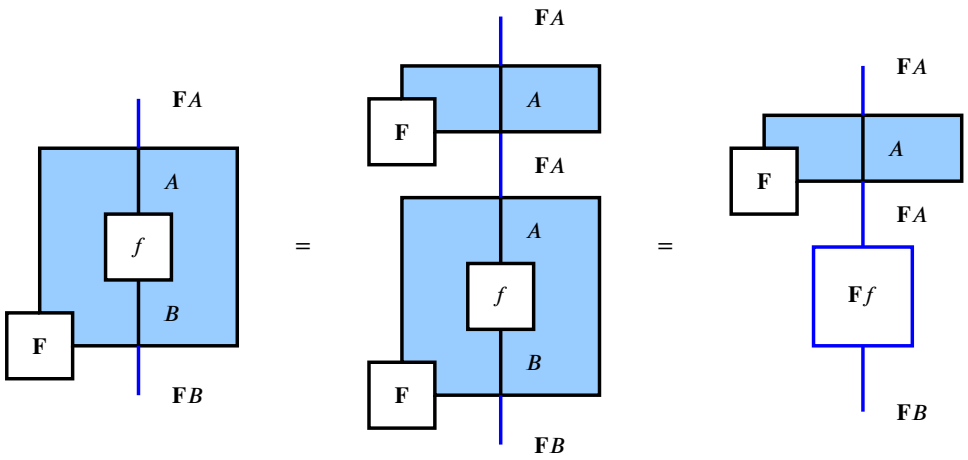


Figure 1.21:

Definition 1.2.1 (Discrete opfibration). $\mathbf{F} : \mathcal{C} \to \mathcal{D}$ is a *discrete fibration* when: for all morphisms $f : \mathbf{F}A \to B$ in $\mathcal{D}$ with domain in the image of $\mathbf{F}$, there exists a unique object Φ_f^A and a unique morphism $\phi_f : A \to \Phi_f^A$ in $\mathcal{C}$, such that $f = \mathbf{F}\phi_f$. Diagrammatically, we can present all of the above as an equation reminiscent of sliding a morphism *into* a functor box from below. The process inside the box is called *the lift* of the process that was slid in. The collection of all lifts over a wire or box is called *the fibre over* that wire or box.

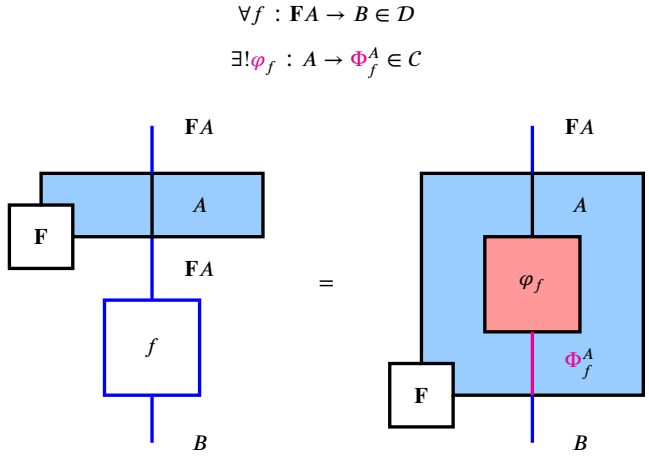
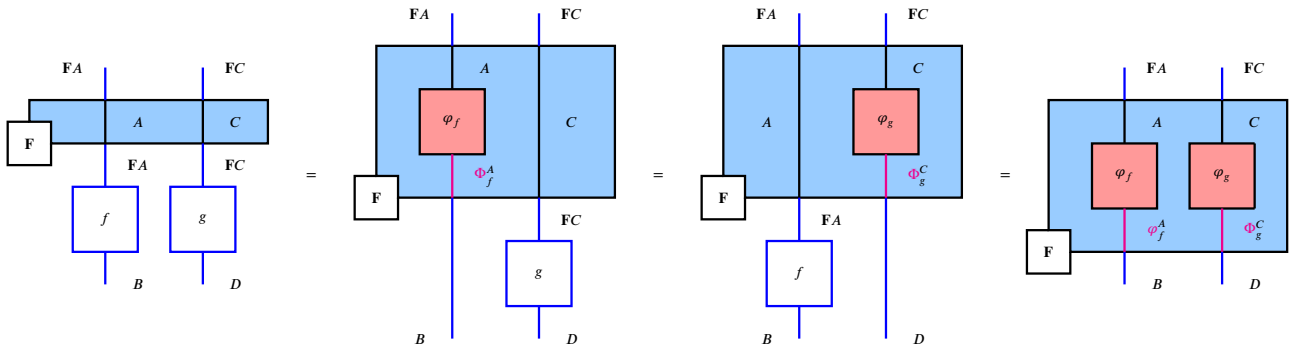


Figure 1.22:



Definition 1.2.2 (Monoidal discrete opfibration)**.** We consider **F** to be a *(strict, symmetric) monoidal discrete opfibration* when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the depicted equations relating lifts to interchange hold. The diagrammatic motivation for the additional coherence equations is that – if we view the lifts of opfibrations as sliding morphisms into functor boxes – we do not want the order in which sliding occurs to affect the final result. In this way, lifts behave as 'graphical primitives' in the same manner as interchange isotopies and symmetry twists.

1.2.1 What are they good for?

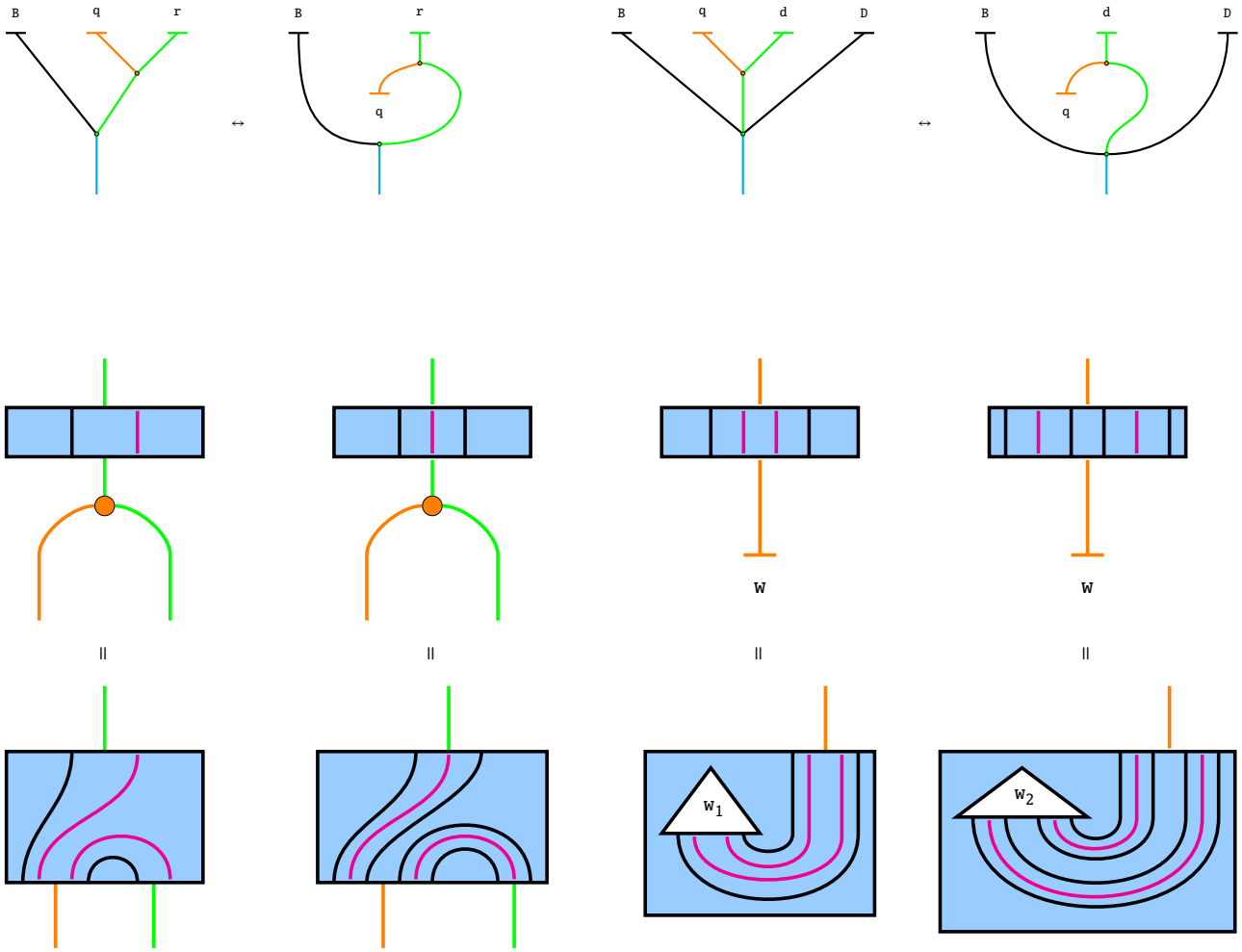


Figure 1.23: Now we try to use monoidal discrete opfibrations to help us solve the speaker's nonfunctoriality problem (Example 1.10). First we flip over the labels and introduction rules for adverbs. Call this a *dependent CFG*, or *dCFG*. Treating the label as a test rather than a state will allow the fibration-box to choose the right version based on the domain wires as it expands top-down. In this case, since CFGs are planar, flipping causes no confusion, since we can always flip the labels back over. There are several ways to do this formally, by e.g. specifying a new string-diagram signature from the old one or assuming rigid autonomous completion, and it doesn't matter which we use.

Figure 1.24: Recall that opfibrations can decide which lift to depict given a choice of codomain wires. We would like to encode the dependency of the upside-down adverb labels and introduction rules as lifts that depend on the lift of the verb wire, which may be either an intransitive or transitive verb.

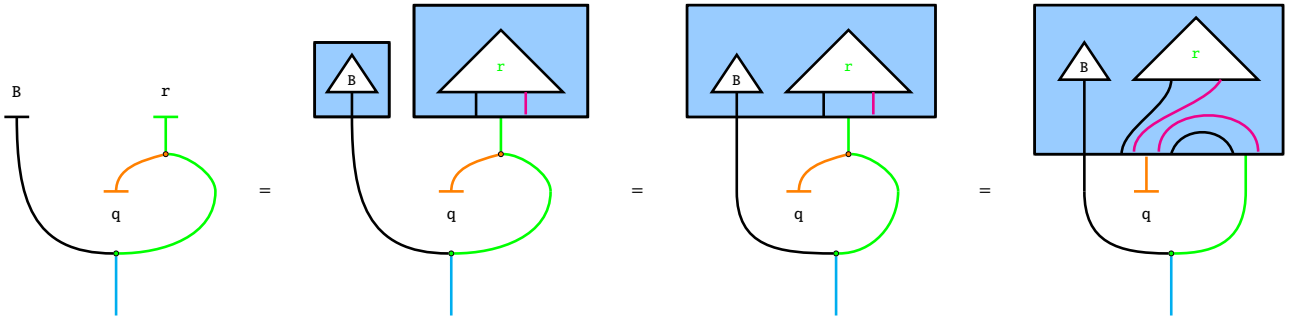


Figure 1.25: Instead of *us* making the choice, we can force the choice using the information of the CFG structure. Starting from a dCFG diagram, the state-labels have unique lifts: noun labels in CFGs correspond uniquely to noun-states in pregroup diagrams, and verb labels to verb-states which may be either intransitive or transitive. This obtains the first equation. The second equation is obtained by monoidality. The third "eating downwards" equation is obtained by the opfibration property; note that because the codomain wires before the lift are already decided to be those of an intransitive verb's pregroup type, the correct adverb introduction rule can be selected for the lift.

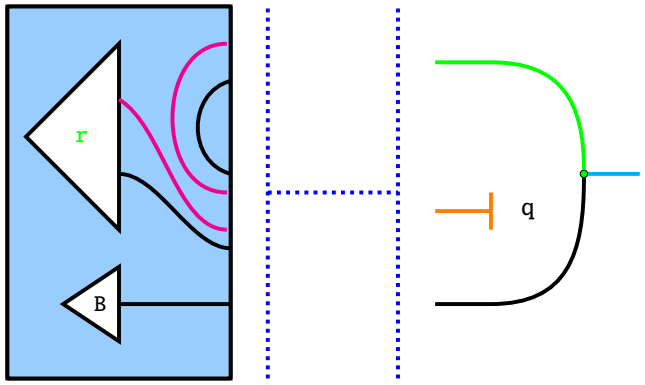


Figure 1.26: But there's a technical problem. We have been assigning wires from the codomain of the lift to the dCFG implicitly, by grouping wires together visually to indicate which wires inside the functor box correspond to wires outside. However, when we consider the algebraic data available, all we know is depicted in the figure: we need some way to assign the wires. Solving the wire assignment probem will be the focus of the next section.

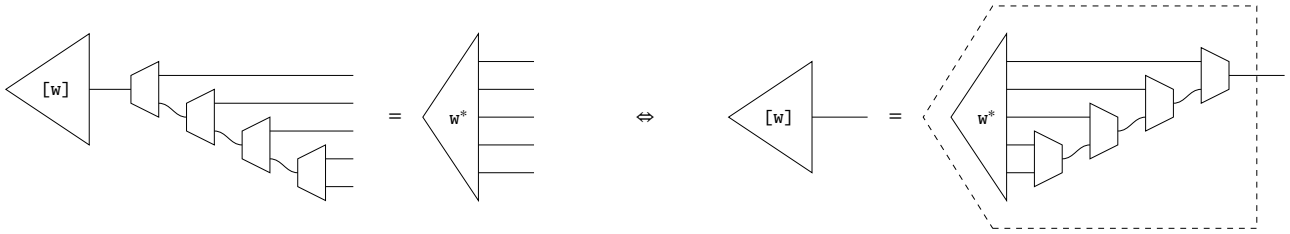
1.3 *Strictified diagrams for monoidal categories*

The crux of the issue sketched in Figure 1.26 is that while pregroup *proofs* – viewed as sequent trees – syntactically distinguish the roots of subtrees, interpretation as pregroup *diagrams* in a monoidal category forgets the subtree structure of the specific proof the diagram arises from. But it is precisely this forgotten structure that contains the algebraic data we require to keep track of (co)domain data diagrammatically. So a solution would be to force the diagrams in the blue domain recording pregroup data to hold onto this proof structure. For this purpose we use strictified diagrams for monoidal categories, defined in the margins.

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

Construction 1.3.3 (Pregroups with bracketing)**.** Where **PGD** is a rigid monoidal category generated by pregroup states and (directed) cups, we define pregroups-with-bracketing as a category denoted $\overline{\mathbf{PGD}}^{*}$, which is obtained as follows. Throughout, denote the rigid monoidal tensor product as $\otimes$ and the strictified tensor as $\bullet$.

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



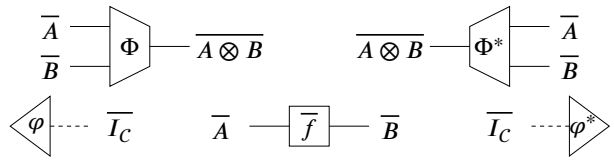
The $\mathtt{w}^*$ and `[w]` generate a freely strictified rigid autonomous category $\overline{\mathbf{PGD}}^{\dagger}$, from which we obtain the desired $\overline{\mathbf{PGD}}^{\star}$ as a subcategory generated by:

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

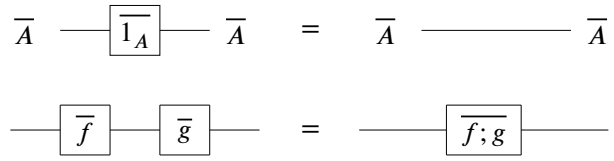
Definition 1.3.1 (Strictified string diagrams)**.** (Presentation taken from CITE) Fix an arbitrary (non-strict) monoidal category $\mathcal{C}$. The *strictification* $(\overline{\mathcal{C}}, \bullet)$ is defined as follows (where strictness of $\overline{\mathcal{C}}$ entitles use of string-diagrammatic notation):

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

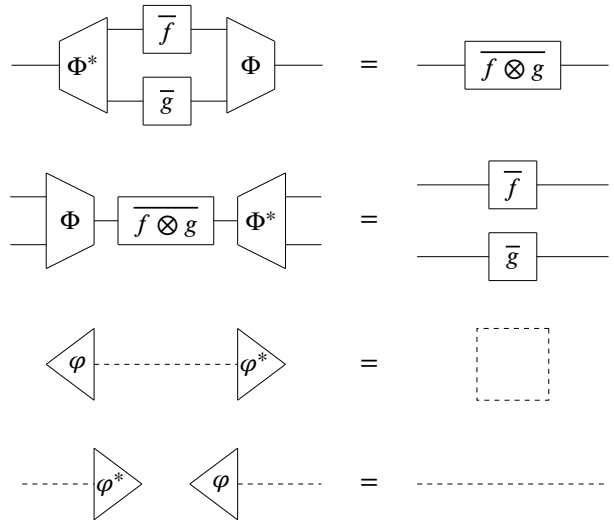
(2) The following generators, with $\overline{f} : \overline{A} \to \overline{B}$ for each $f \in \mathcal{C}(A, B)$, where we adopt the convention of notating the monoidal unit with a dashed line:



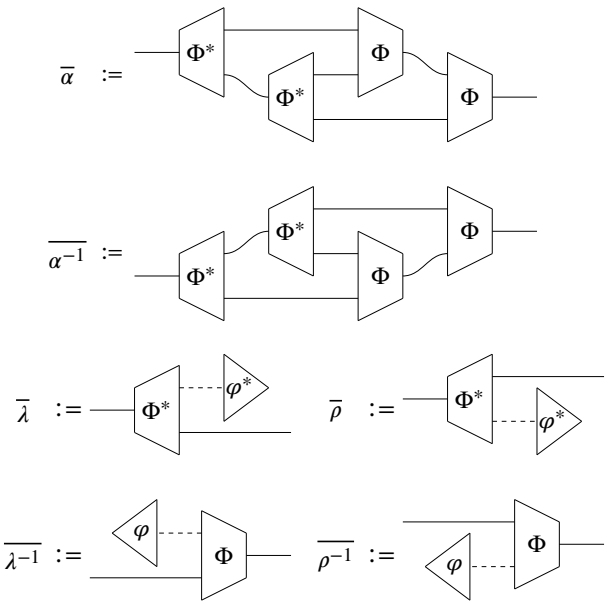
(3) The following functoriality equations:



(4) The following adapter equations:

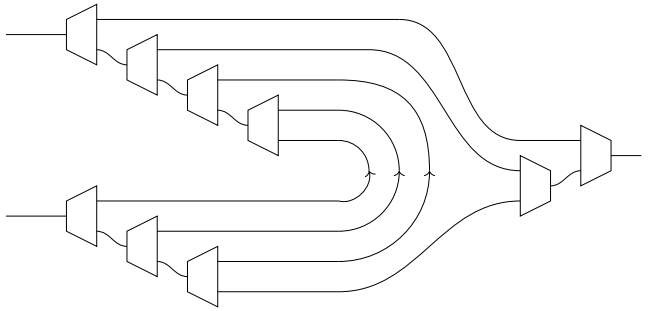


(3) The following representations of the natural isomorphisms in the definition of a monoidal category:



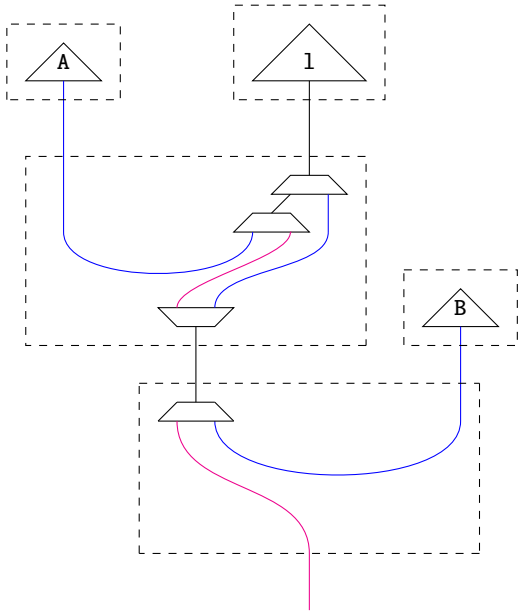
Proposition 1.3.2 ($\bar{\mathcal{M}}$ and $\mathcal{M}$ are monoidally equivalent). *Proof.*
CITE □

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

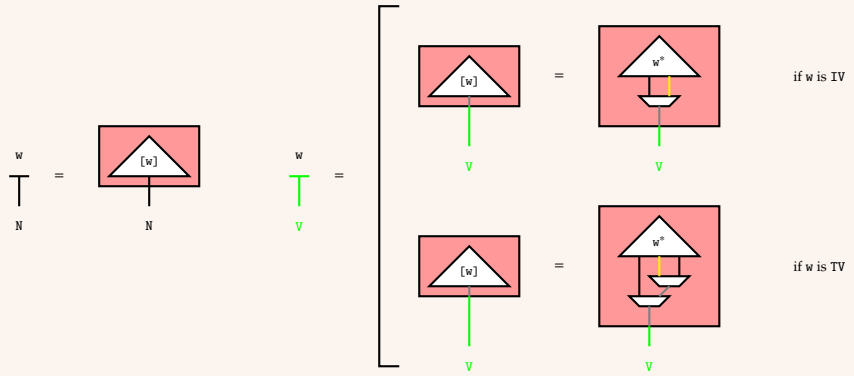


Example 1.3.4 (Pregroups with bracketing recover proof trees)**.** The essence of the construction is to maintain a correspondence with proof-tree structure: a left-bracketed collection of types corresponds to a pregroup typing that is stuck together as the outcome of a sequent rule and must thereafter travel together. Starting from bracketed word states `[w]`, point 2 of the construction maintains an invariant correspondence that bracketed collections of types are the roots of proof steps.

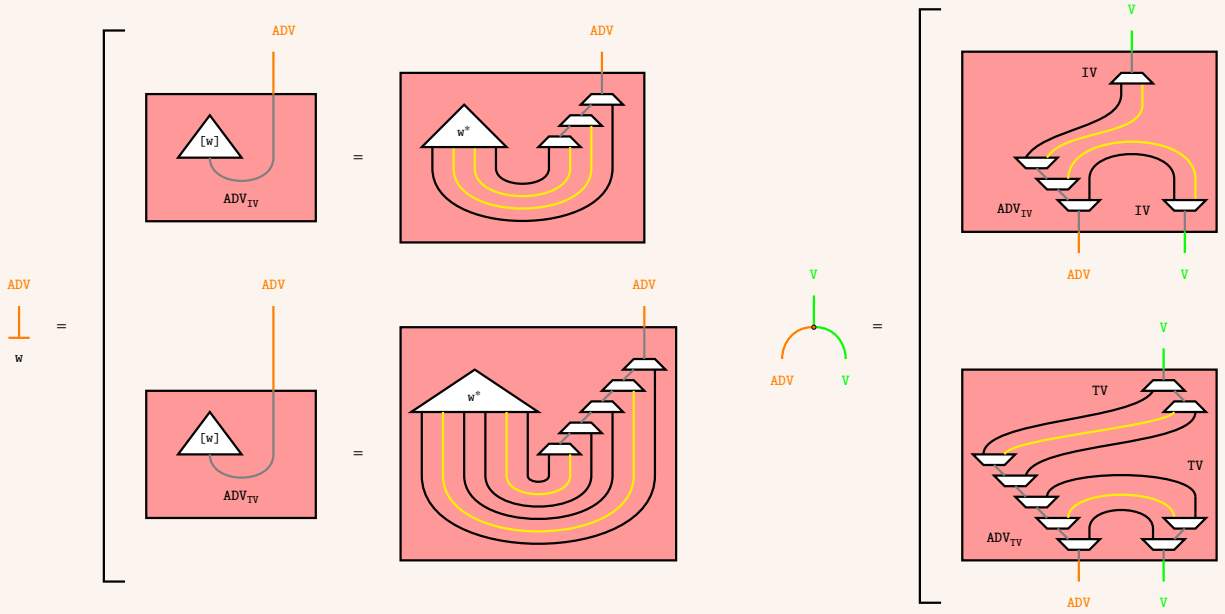
$$\dfrac{\dfrac{\texttt{Alice} : n \qquad \texttt{likes} : {}^{-}n \cdot s \cdot n^{-}}{\texttt{Alice␣likes} : s \cdot n^{-}} \qquad \texttt{Bob} : n}{\texttt{Alice␣likes␣Bob} : s}$$



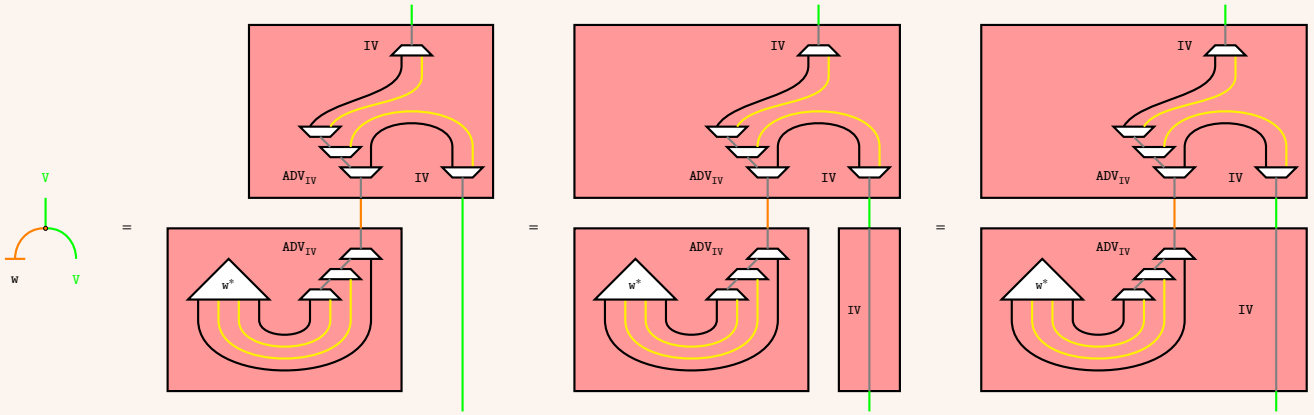
Construction 1.3.5 (Discrete monoidal opfibration from pregroups with bracketing into dependent CFGs). We aim to elucidate the pregroup with bracketing in sufficient detail to describe the functor into dCFGs; in particular, we need to know what the generators of the pregroup are. The fibre over a noun state in the dCFG is the corresponding noun-state in the bracketed pregroup. The fibre over a verb state in the dCFG is either an intransitive or transitive word-state, depending on the word `w`. Note that only the `[w]` are available as generators, through we may reason about them as if they are tensor-bracketings of `w`*.



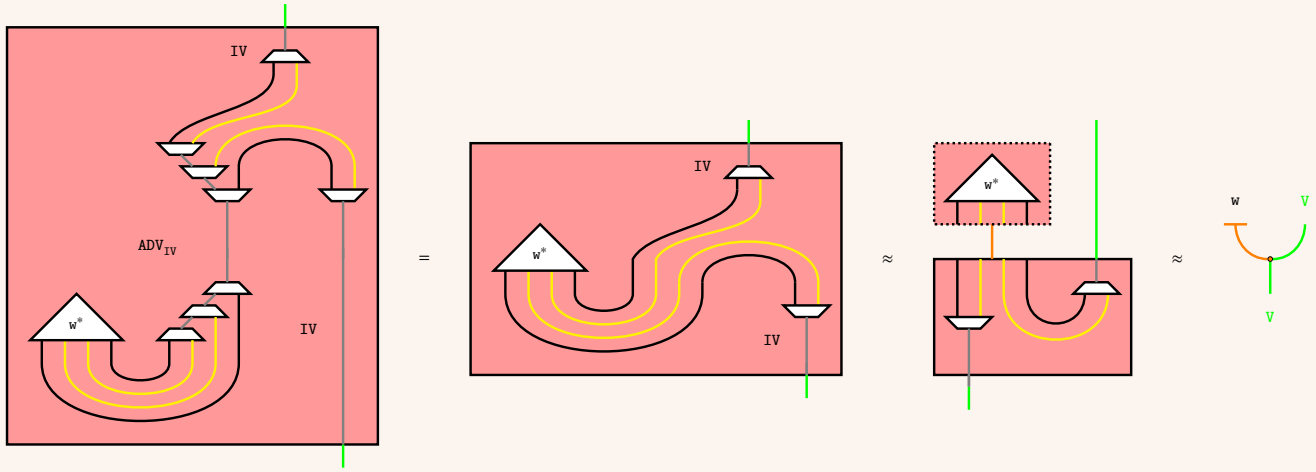
We depict the case of adverbs explicitly. Here are two lifts of the adverb label in the fibre of the opfibration corresponding to the intransitive and transitive case, and the corresponding lifts for the introduction rule for adverbs.



The correspondence of introduction rules in the dCFG to proof steps in the sequent formulation of pregroup proofs (c.f. Example 1.3.4) is obtained obliquely, because labels for dependent types are facing the wrong way; hence the cups for the lifts of labels. We can observe the correspondence by the following diagrams. The second equation is a supplied choice of lift on the V wire, so that the third monoidality equation allows the top and bottom boxes to typematch.



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



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

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

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

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

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

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

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

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

The three dependent types are:

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

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

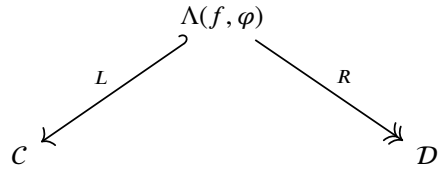
Proposition 1.3.6 (Construction 1.3.5 is a discrete monoidal fibration)**.** *Proof.* Monoidality is evident. For unique lifts, we observe that for each bracketing of wires, construction 1.3.3 guarantees a unique lift for each introduction rule in the dCFG, and Construction 1.3.5 guarantees a unique lift for each label. For the additional interchange condition of Definition 1.22, it suffices to observe that the introduction rules of dependent labels uniquely determine the codomain of the lift given the domain, and that by design in Construction 1.3.5, independent labels as states have predetermined lifts. □

1.4 *Monoidal cofunctor boxes*

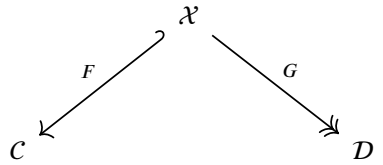
Now that we know how to solve the wire-assignment problem with the help of monoidal discrete opfibrations from strictified diagrams that do bracketing, we can at last see what monoidal cofunctor boxes are.

Definition 1.4.3 (Bijective-on-objects functor)**.** From CITE, Defn 2.8 . A functor $F : \mathcal{C} \to \mathcal{D}$ is *bijective-on-objects* if for all $d \in \mathcal{D}$, there exists a $c \in \mathcal{C}$ such that $Fc = d$.

Proposition 1.4.4 (Cofunctors as spans of functors)**.** From CITE, Prop 2.10 , all cofunctors $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ correspond to spans of functors where the left leg L is bijective on objects and the right leg R is a discrete opfibration:



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



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

Definition 1.4.5 (Monoidal cofunctor)**.** A monoidal cofunctor, following Proposition 1.4.4, is a span of functors such that the left leg is monoidal and bijective-on-objects, and the right is a monoidal discrete opfibration by Definition 1.22.

Construction 1.4.6 (Monoidal cofunctor box)**.** A monoidal cofunctor box first uses the inside-out convention for functor boxes for the right leg, and then the outside-in convention for the left leg.

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

Definition 1.4.1 (Cofunctors)**.** From CITE Defn 2.2 . A *cofunctor* $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ consists of a function $f : \mathcal{C}_0 \to \mathcal{D}_0$ which I'll call *lowering*, together with a *lifting operation* φ, a function that maps pairs of objects of $\mathcal{C}$ and certain morphisms in $\mathcal{D}$ to morphisms of $\mathcal{C}$:

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

The following conditions are required:

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

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

2. The lifts of identities are identities.

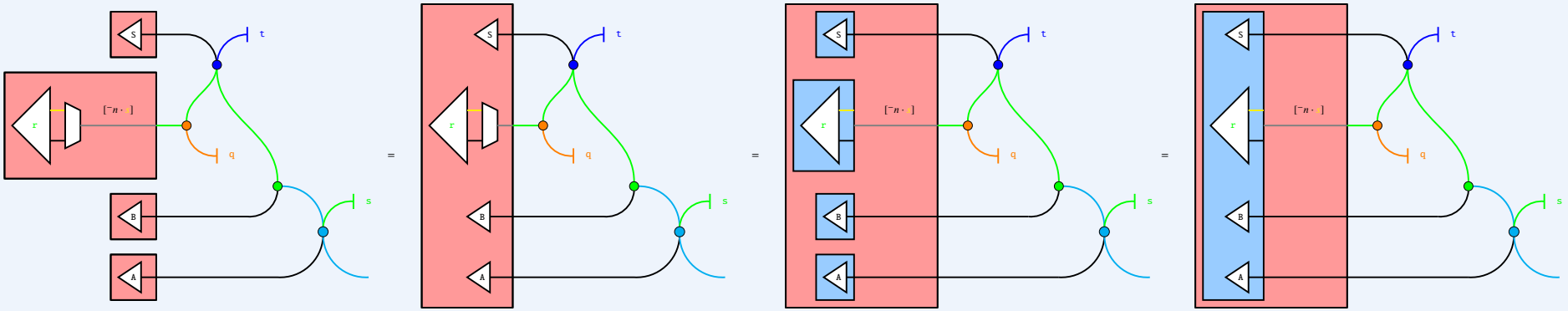
$$\varphi(c, 1_{f(c)}) = 1_c$$

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

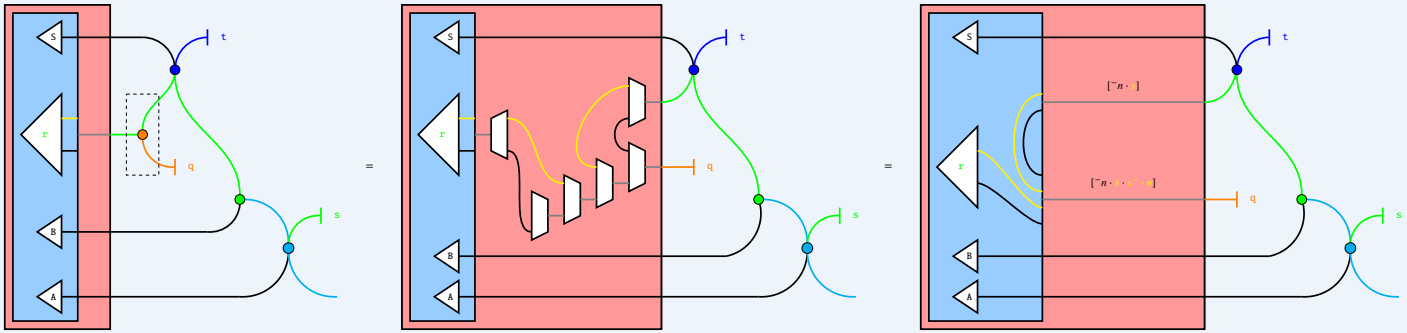
$$\varphi(c, v \circ u) = \varphi(\mathrm{cod}(\mathrm{cod}(\varphi(c,u))), v) \circ \varphi(c,u)$$

Remark 1.4.2. Conditions 2 and 3 of the definition of cofunctor are reminiscent of functors. It is instructive but tedious to calculate with the base definition. We use the slick alternative formulation by Bryce Clarke.

Example 1.4.7 (Turning dCFGs into pregroup diagrams with a monoidal cofunctor)**.** The apex is given by Construction 1.3.3. The right leg is the monoidal discrete opfibration from Construction 1.3.5 into the dCFG. In the first diagram below, we apply the opfibration to the word states, which have unique lifts. The second diagram follows by monoidality. In the third, we apply the left leg of the cofunctor using the outside-in convention, which is the evidently bijective-on-objects monoidal functor to the ambient rigid autonomous category of pregroup diagrams from the free strictification. In the fourth, we apply the monoidality of the inner functor.

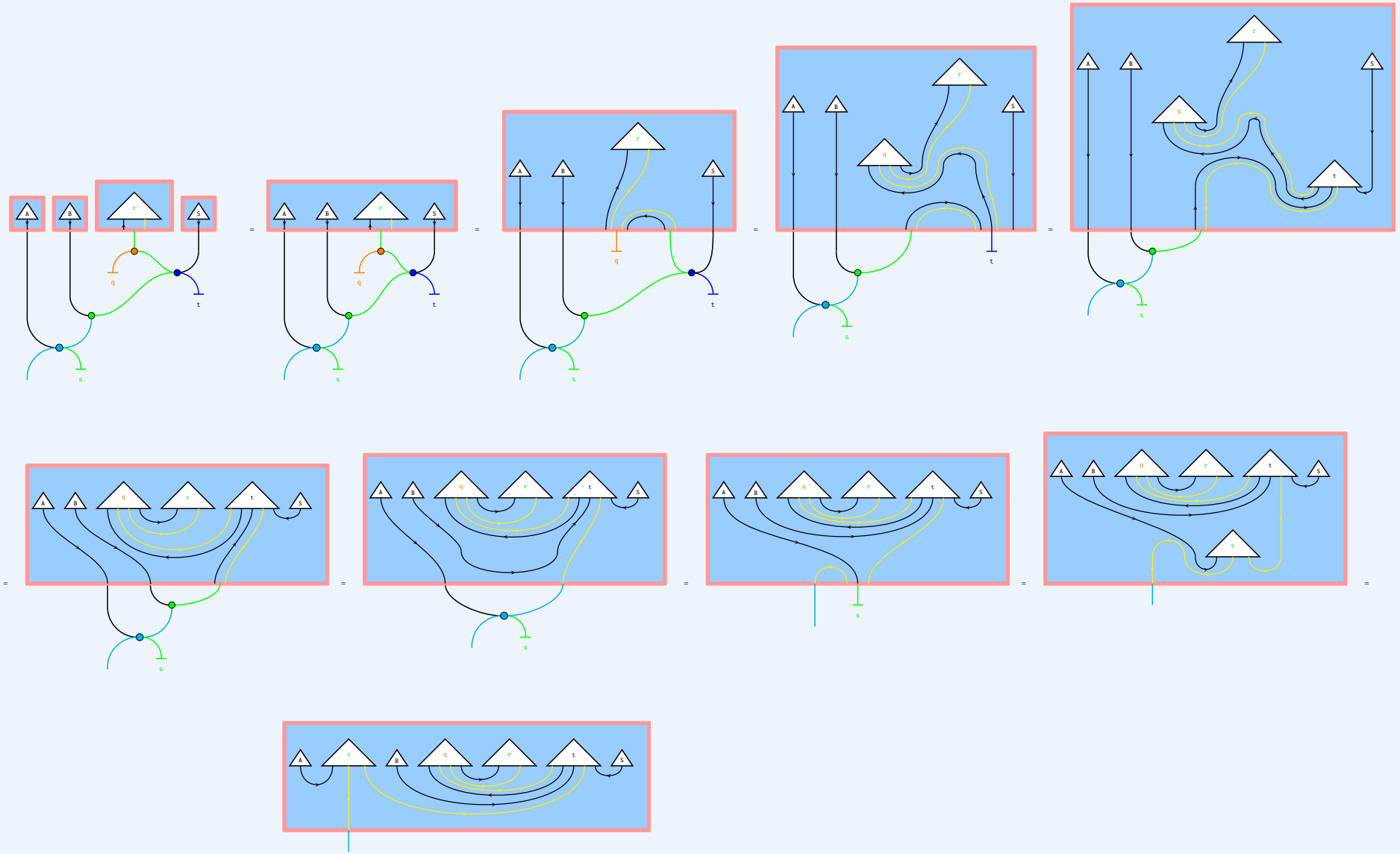


For the first equation, the magenta opfibration box now has already chosen a codomain for the lift, so it can eat the next morphism highlighted in a dashed box. For the second equation, note that eating-rules swap over for the different conventions of functor boxes: while inside-out functor boxes need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



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

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



1.5 *Monoidal kinda-confunctor boxes*

Now we'd like to use the cofunctor technology we have developed to tackle the other problem, the nonfunctoriality of internal wirings for the listener (Example 1.12.) We run into a problem again: the right leg cannot be a discrete opfibration into pregroup diagrams.

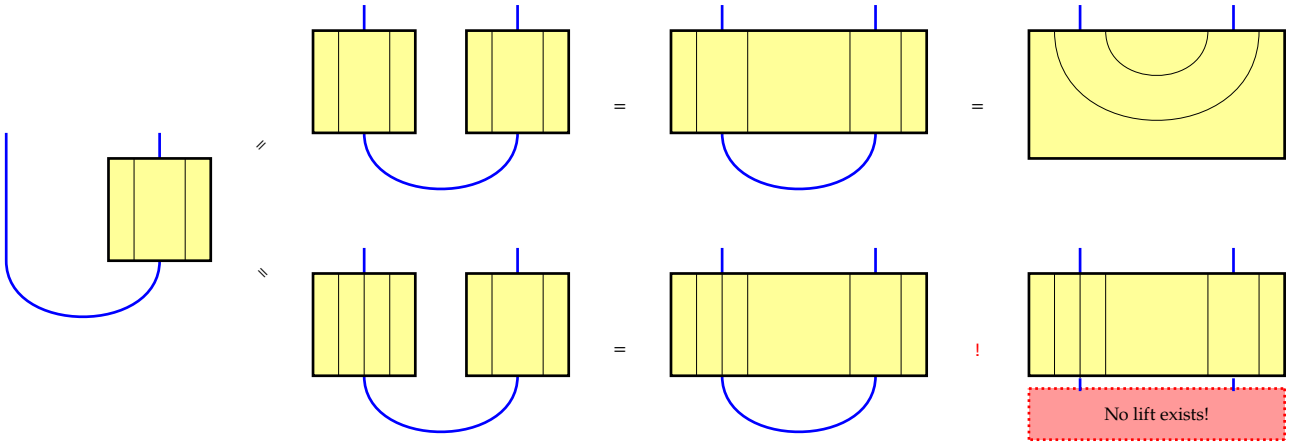


Figure 1.27:

Example 1.5.1 (Lifts for cups are not always defined)**.**

Starting from the leftmost diagram, in order to let the functor box eat the whole diagram, we need to first choose a lift for the left-sentence wire for the cup. Recalling Figures 1.6 and 1.12, there are at least two lifts for the sentence-wire in pregroup diagrams, for the case of two or three noun-wires. Everything works smoothly when the lifts on the two sentence wires of a cup match. When if we make the wrong choice and they don't, there is no lift, because there is no such thing as a cup that has two wires on one end and three on the other. Recall from Definition 1.5.2 that a unique lift is required for *every possible* codomain inside the functor box; so we do not have a discrete opfibration, and so we cannot have a cofunctor.

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

Definition 1.5.2 (Kinda-opfibration)**.** $\mathbf{F} : C \to \mathcal{D}$ is a *kinda-opfibration*[3] when for all morphisms $f : \mathbf{F}A \to B$ in $\mathcal{D}$ with domain in the image of $\mathbf{F}$, there exists some object Φ_f^A and some $\phi_f : A \to \Phi_f^A$ in C, such that $f = \mathbf{F}\phi_f$.

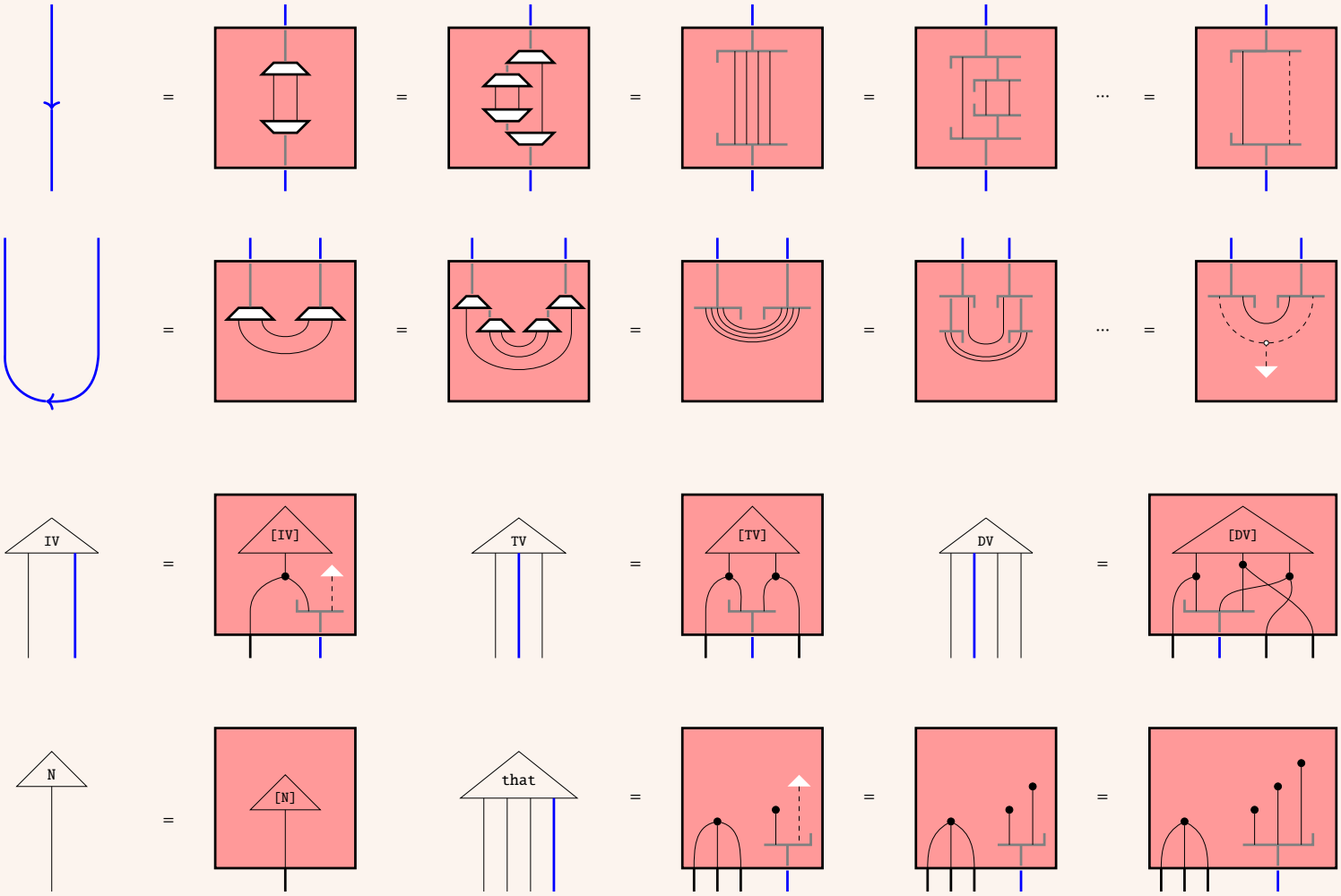
Definition 1.5.3 (Monoidal kinda-opfibration and kinda-cofunctor)**.** Mentally search and replace discrete for kinda- in Definition 1.22 and Proposition 1.4.4.

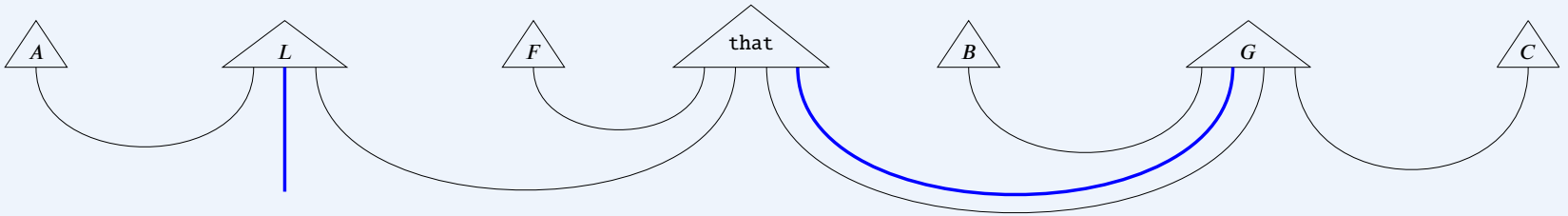
[2] The job we want the right leg to do is just to bookkeep choices of lift in its fibres. Discrete opfibrations do too much by enacting safety standards and curtailing our choice. Nevermind universal properties, we want to make our own decisions, good or bad.

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

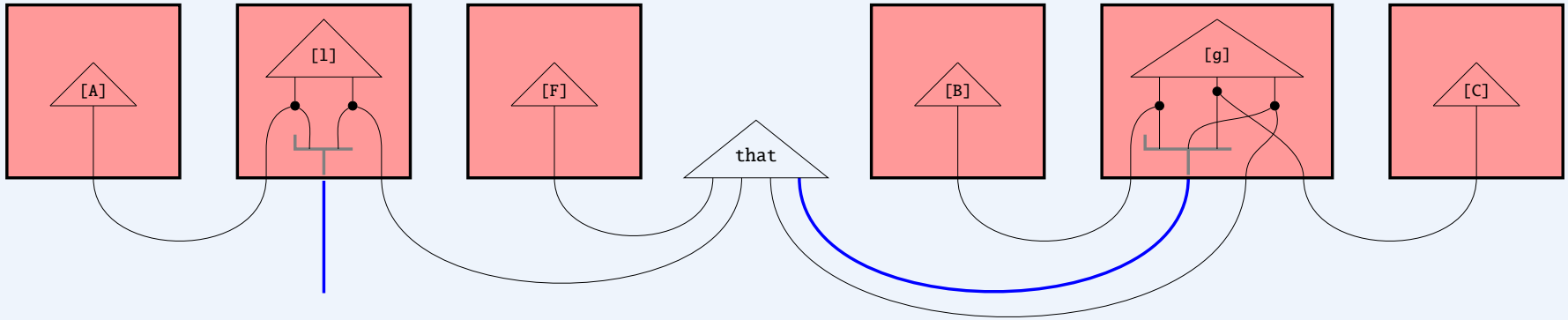
Example 1.5.4 (A monoidal kinda-opfibration into pregroup diagrams from a subcategory of bracketed pregroups with spiders)**.**

Source category is the free strictification of a single wire equipped with a spider, with generator-states implied by the following equations. Target is pregroup diagrams in n, s. Sentence type is bracketings of nouns: alternate bracketing notation is introduced for brevity. Strictified unit is used to distinguish the bracketed single noun wire from the plain noun wire, which is its own lift. Bracketing monoidal units gives the Dyck language, which may be in principle used to distinguish turning numbers in the rigid autonomous setting, omitted for brevity. For more internal wirings for other grammatical categories, see CITE . This data suffices to model how the listener recovers the data conveyed by the speaker, c.f. Example 1.12. Solution to Example 1.12 on next page.

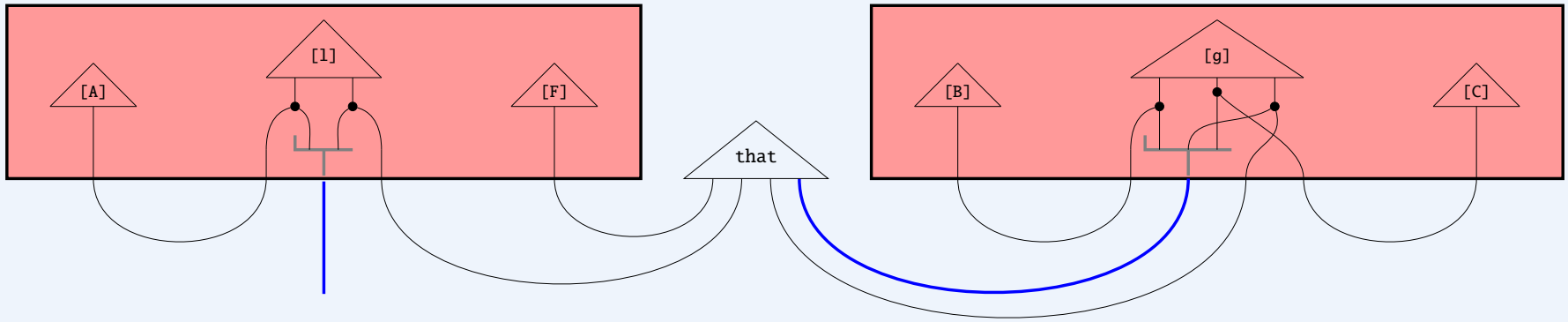




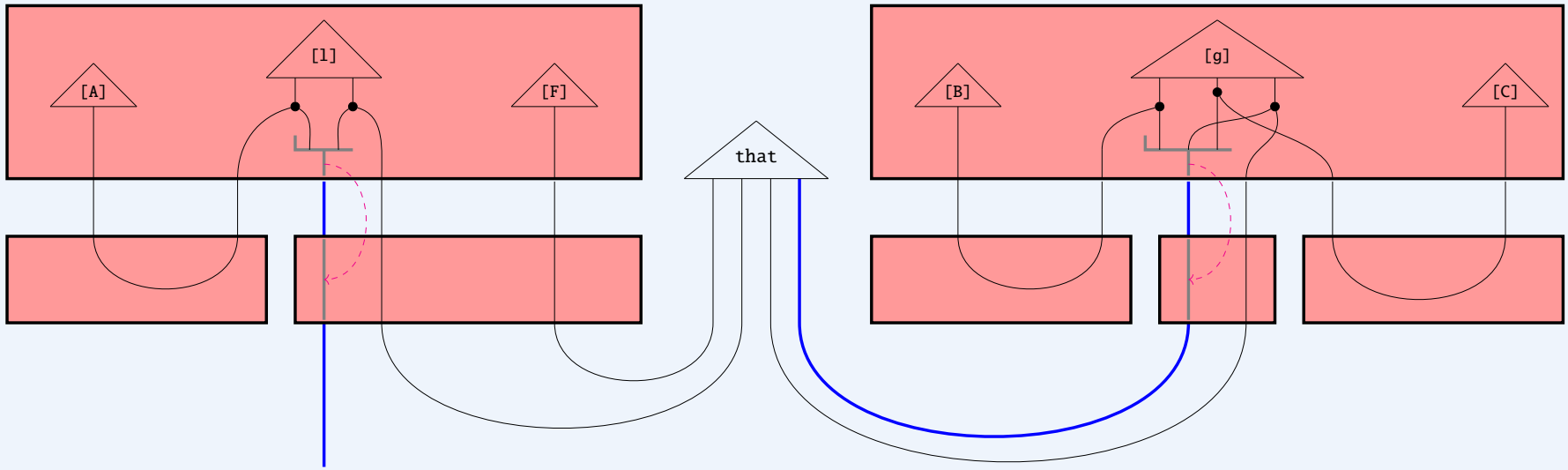
Some lifts are determined.



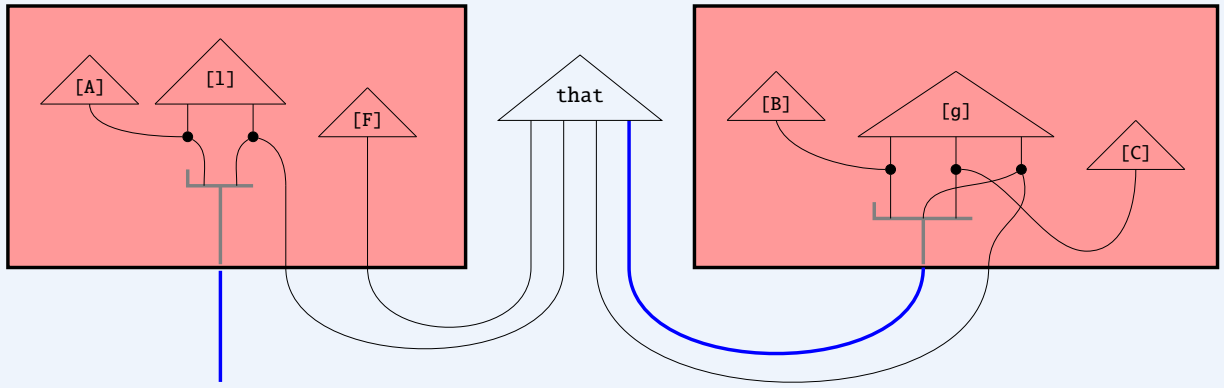
Merge.



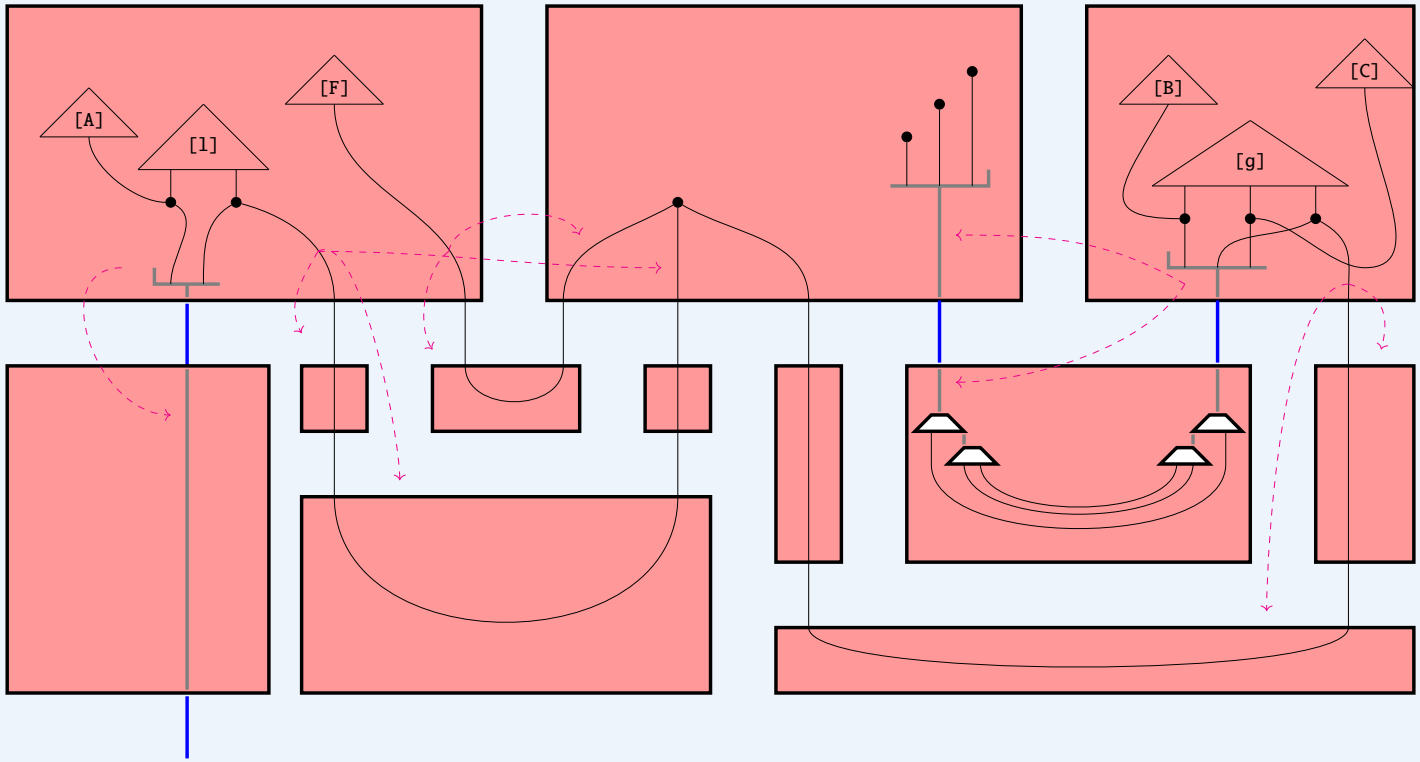
Noun-cups have determined lifts. Typematching lifts inferred from connectivity.



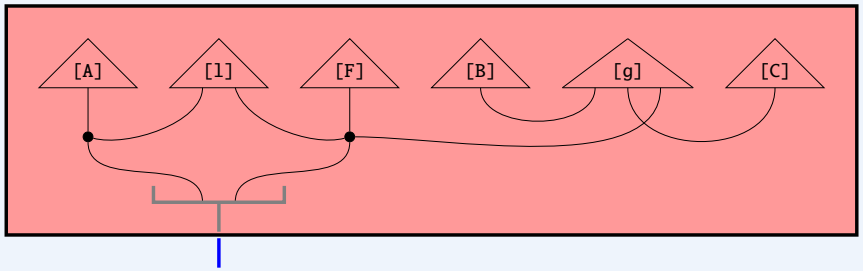
Merge.



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



Merge. Cancel brackets. Simplify spiders.



1.6 Discussion and Limitations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Example 1.6.1 (Exclamations: how meaning affects grammar)**.** The following examples from [Lakofflecture] illustrate how whether a phrase is an *exclamation* affects what kinds of grammatical constructions are acceptable. By this argument, to know whether something is an exclamation in context is an aspect of meaning, so we have cases where meaning determines grammar. Observe first that the following three phrases are all grammatically acceptable and mean the same thing.

`nobody knows how many beers Bob drinks`

`who knows how many beers Bob drinks`

`God knows how many beers Bob drinks`

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

`somebody knows how many beers Bob drinks`

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

God doesn't know how many beers Bob drinks

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

Bob drinks God knows how many beers

Bob drinks who knows how many beers

But it is awkward to have:

Bob drank nobody knows how many beers

And it is not acceptable to have:

Bob drank Alice knows how many beers

Example 1.6.2 (Garden path sentences)**.** So-called "garden path" sentences illustrate that listeners have to make choices to resolve lexical ambiguities. One such garden-path sentence is The old man the boat, where typically readers take The old man as a noun-phrase and the boat as another noun-phrase. We can sketch how the readers might think with a (failed) pregroup grammar derivation:

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

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

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

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

Example 1.6.3 (Ambiguous scoping)**.** Consider the following sentence:

$$\texttt{Everyone loves someone}$$

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

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

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

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

$$[\![\texttt{everyone}]\!] = \lambda(\lambda x.V(x)).\forall x : V(x) \tag{1.1}$$

$$[\![\texttt{loves}]\!] = \lambda x \lambda y.\texttt{loves}(x, y) \tag{1.2}$$

$$[\![\texttt{someone}]\!] = \lambda(\lambda y.V(y)).\exists y : V(y) \tag{1.3}$$

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

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

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

Example 1.6.4. (grammar pieces – as simple as it gets) `Bob runs. Bob quickly runs. Bob drinks beer. Bob quickly drinks beer.`

2
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, and a brief note on how text circuits relate to Montague's "Universal Grammar".

2.1 *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 weak n-categories CITE , 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 CITE . 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.

2.1.1 *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 2.1.1.

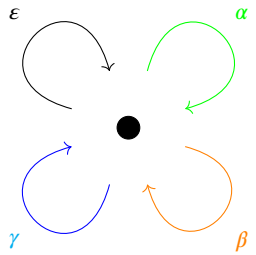
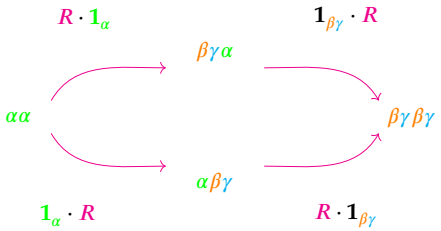
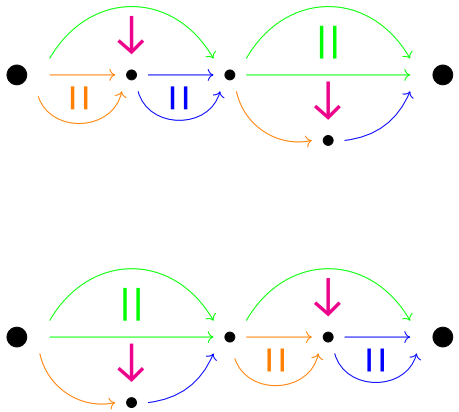


Figure 2.1: The category in question can be visualised as a commutative diagram.

WE CONSIDER REWRITES TO BE EQUIVALENT, BUT NOT EQUAL. In a string-rewrite system, rewrites are applied one at a time. This means that even for our simple example, there are two possible rewrites from $\alpha \cdot \alpha$ to obtain $\beta \cdot \gamma \cdot \beta \cdot \gamma$. Here are the two rewrites viewed in two equivalent ways, first on the left informally where strings are nodes in a graph and rewrites are labelled transitions and secondly on the right as two distinct commuting 2-diagrams.





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

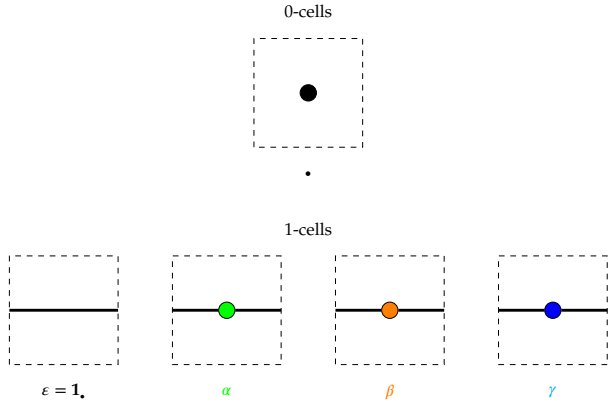


Figure 2.2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from tables as they come for free. Observe that composition of identities enforces the behaviour of the empty string, so that for any string x, we have $\epsilon \cdot x = x = \epsilon \cdot x$.

Figure 2.3: For a concrete example, we can depict the string $\alpha \cdot \gamma \cdot \gamma \cdot \beta$ as a morphism in a commuting diagram.

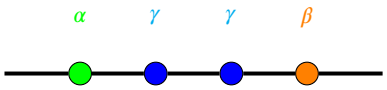


Figure 2.4: The string-diagrammatic view, where ⋆ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

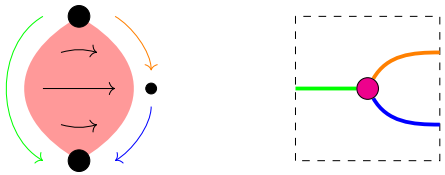


Figure 2.5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual of this view gives us a string diagram for the 2-cell R.

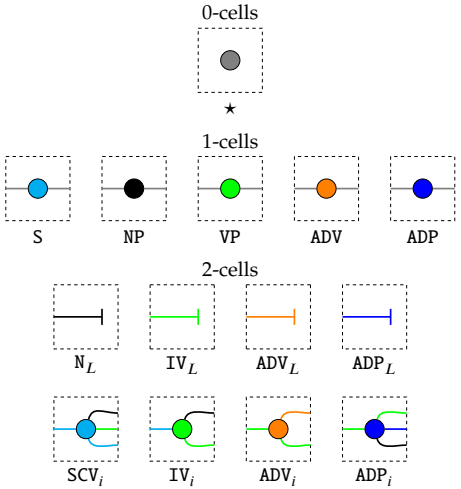
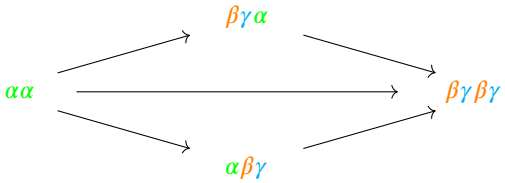
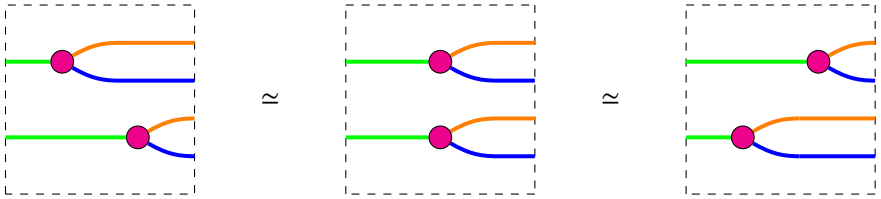


Figure 2.6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for `Alice sees Bob quickly run to school`.

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



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



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

2.1.2 *Tree Adjoining Grammars*

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.

Definition 2.1.1 (*Elementary* Tree Adjoining Grammar: Classic Computer Science style)**.** An elementary **TAG** is a tuple

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

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

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

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

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

Auxiliary trees satisfy the following constraints:

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

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

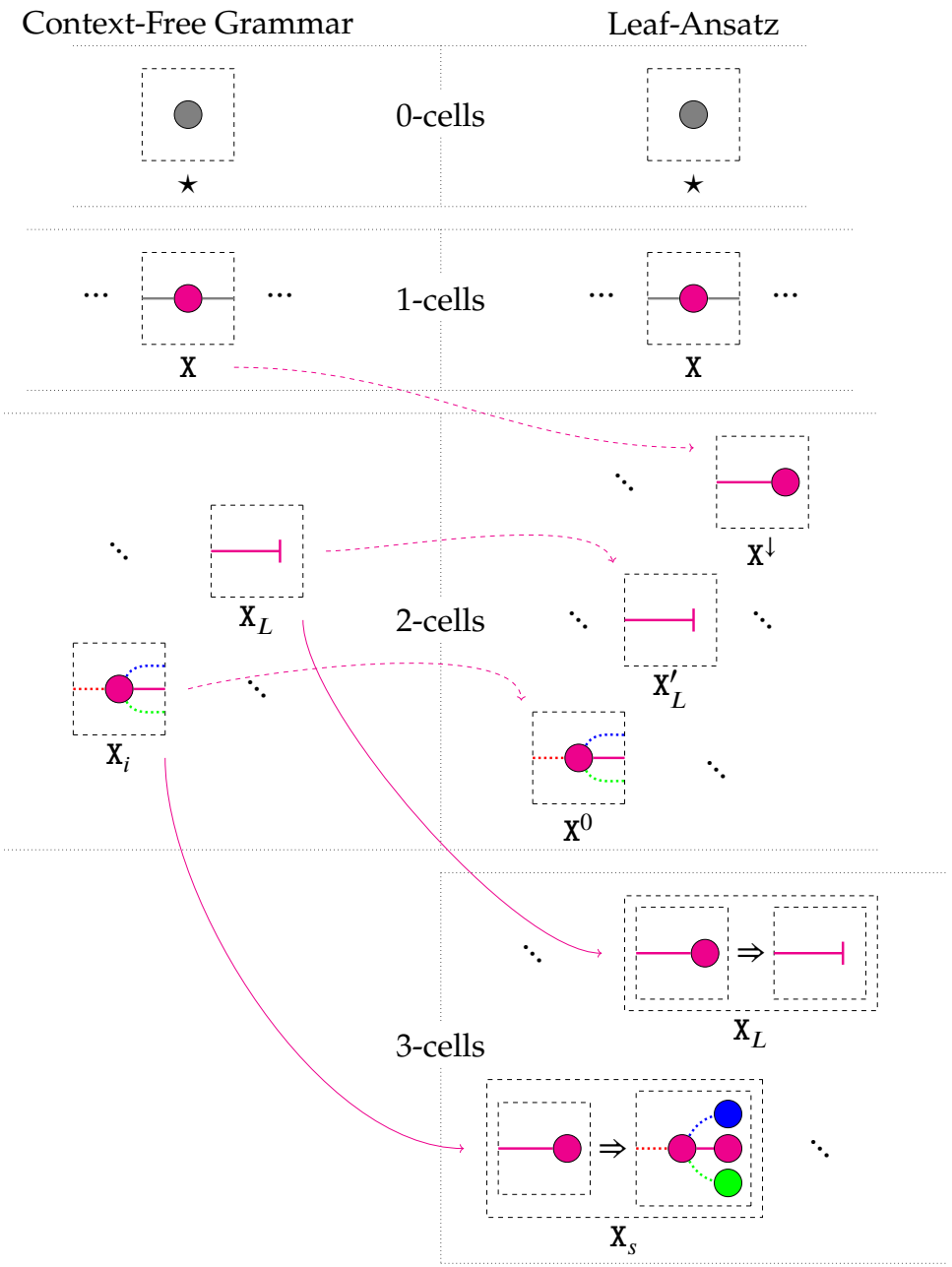


Figure 2.7:

Construction 2.1.2 (Leaf-Ansatz of a CFG)**.** Given a signature $\mathfrak{G}$ for a CFG, we construct a new signature $\mathfrak{G}'$ which has the same 0- and 1-cells as $\mathfrak{G}$. Now, referring to the dashed magenta arrows in the schematic below: for each 1-cell wire type $\mathtt{X}$ of $\mathfrak{G}$, we introduce a *leaf-ansatz* 2-cell $\mathtt{X}^{\downarrow}$. For each leaf 2-cell $\mathtt{X}_L$ in $\mathfrak{G}$, we introduce a renamed copy $\mathtt{X}'_L$ in $\mathfrak{G}'$. Now refer to the solid magenta: we construct a 3-cell in $\mathfrak{G}'$ for each 2-cell in $\mathfrak{G}$, which has the effect of systematically replacing open output wires in $\mathfrak{G}$ with leaf-ansatzes in $\mathfrak{G}'$.

Proposition 2.1.3. Leaf-ansatzes of CFGs are precisely TAGs with only initial trees and substitution.

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

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

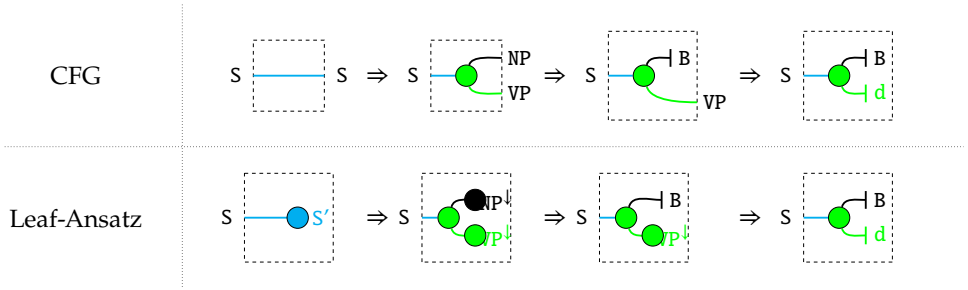


Figure 2.8: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction between nonterminals and terminals.

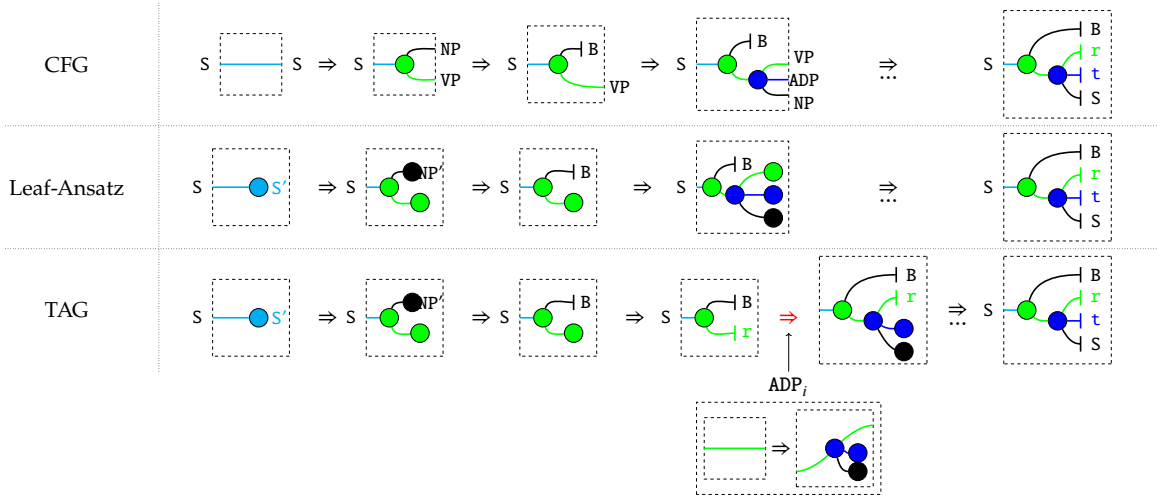
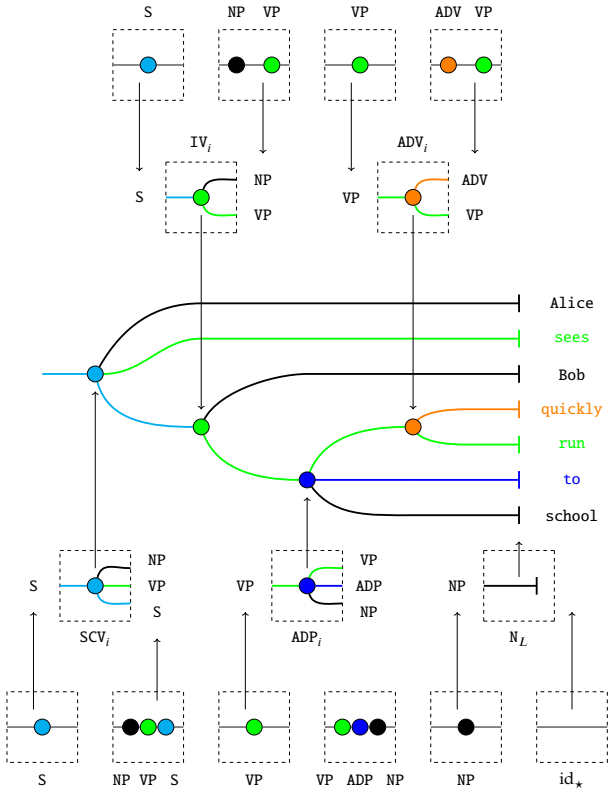


Figure 2.9: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted 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 2.10: Leaf-ansatz signature of `Alice sees Bob quickly run to school` CFG. One aspect of rewrite systems we adapt for now is the distinction between terminal and nonterminal symbols; terminal symbols are those after which no further rewrites are possible. We capture this string-diagrammatically by modelling terminal rewrites as 2-cells with target equal to the 1-cell identity of the 0-cell $\star$, which amounts to graphically terminating a wire. The generators subscripted L (for *label* or *leaf*) correspond to terminals of the CFG, and represent a family of generators indexed by a lexicon for the language. The generators subscripted i (for *introducing a type*) correspond to rewrites of the CFG. Reading the central diagram in the main body from left-to-right, we additionally depict the breakdown of the derivation in terms of rewrites of lower dimension from our signature.



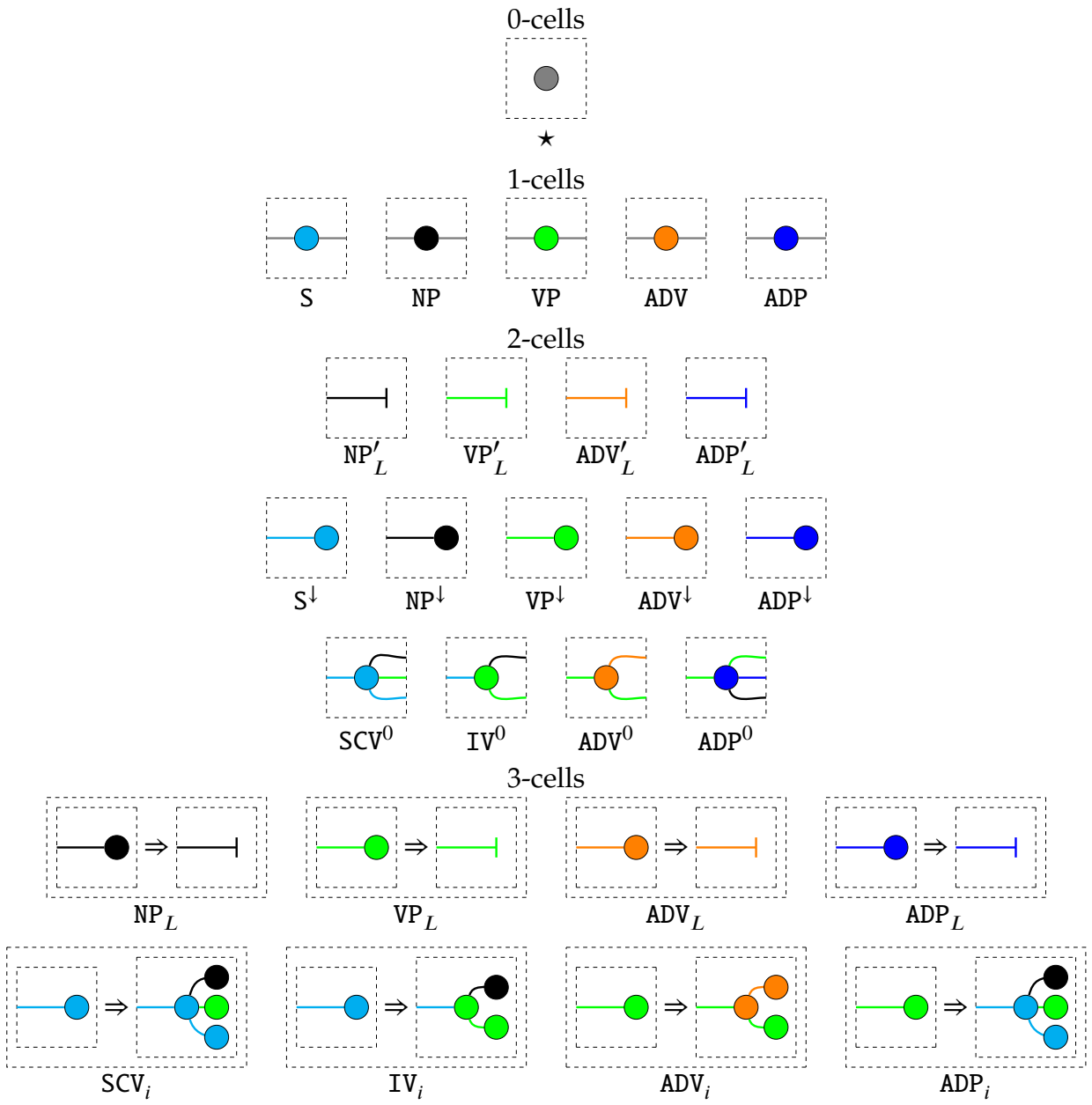
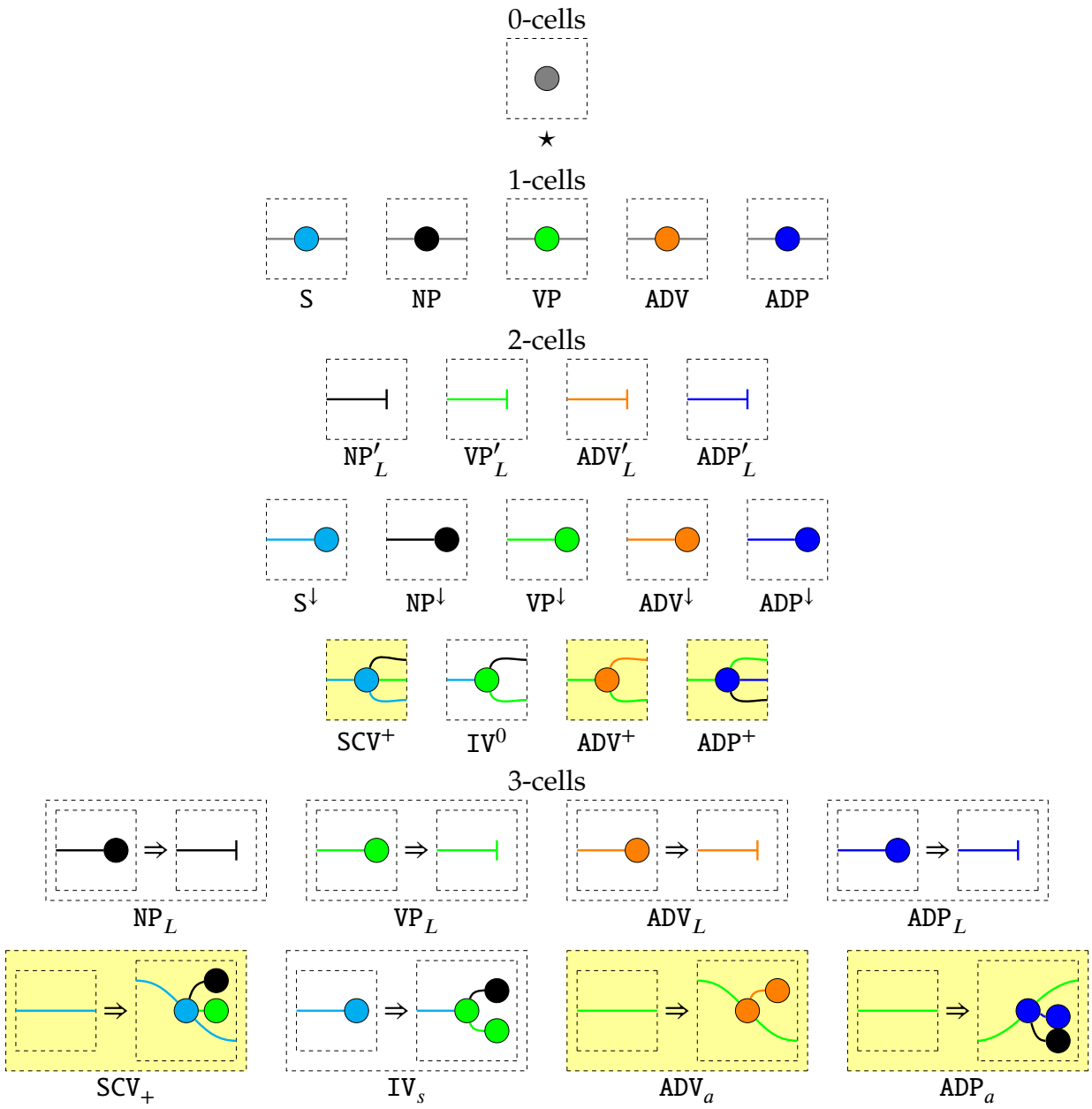


Figure 2.11: TAG signature of `Alice sees Bob quickly run to school`. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction yields as a corollary an alternate proof of Theorem [Joshi 6.1.1...]...

Corollary 2.1.4. For every context-free grammar $\mathfrak{G}$ there exists a tree-adjoining grammar $\mathfrak{G}'$ such that $\mathfrak{G}$ and $\mathfrak{G}'$ are strongly equivalent – both formalisms generate the same set of strings (weak equivalence) and the same abstract syntactic structures (in this case, trees) behind the strings (strong equivalence).

Proof. Proposition 2.7 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 $\mathfrak{G}'$. □



2.1.3 *Tree adjoining grammars with local constraints*

Definition 2.1.5 (TAG with local constraints: CS-style)**.** [Joshi] $G = (I, A)$ is a TAG with local constraints if for each node n and each tree t, exactly one of the following constraints is specified:

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

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

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

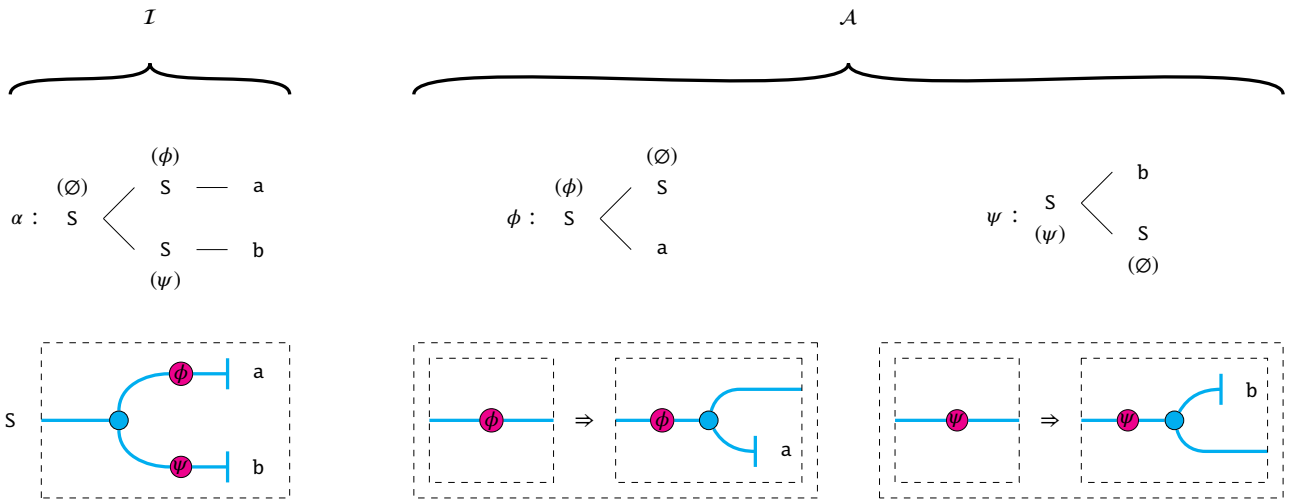


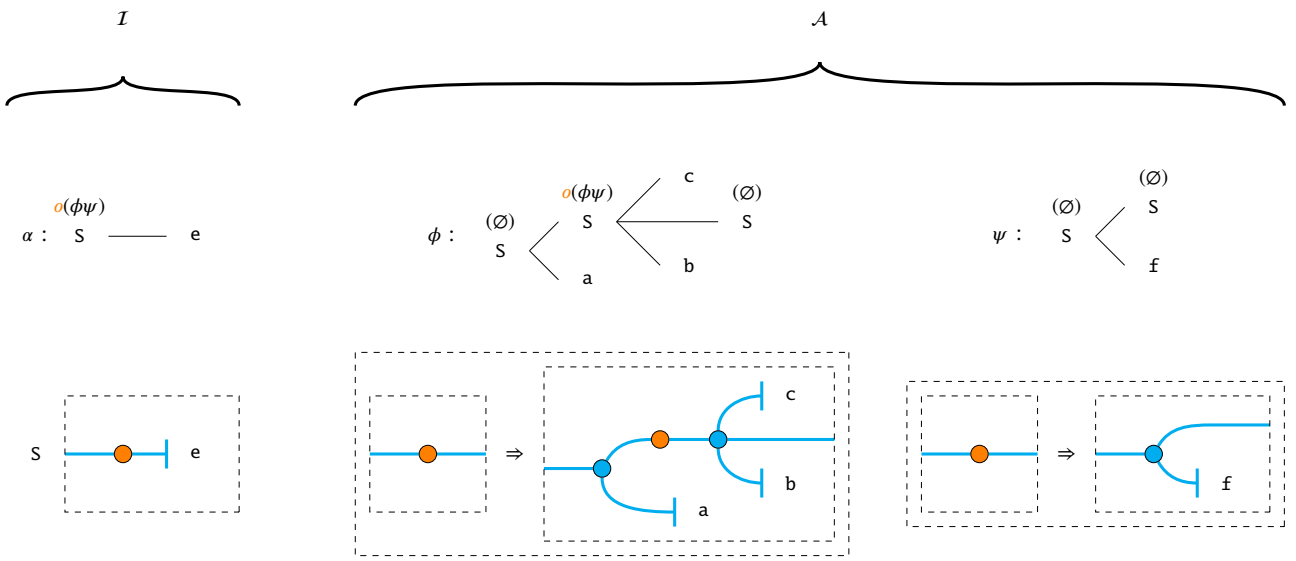
Figure 2.12: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree is presented as a 2-cell where the (SA) rules are rewritable nodes, that serve as sources of rewrites in the 3-cell presentations of the auxiliary trees.

2.1.4 *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 2.13: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree is presented as a 2-cell where the (OA) rule is given its own 2-cell, which is the source of rewrites in 3-cell presentations of auxiliary trees. We may capture the obligatory nature of the rewrite by asking that finished derivations contain no instance of the orange 2-cell.



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

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

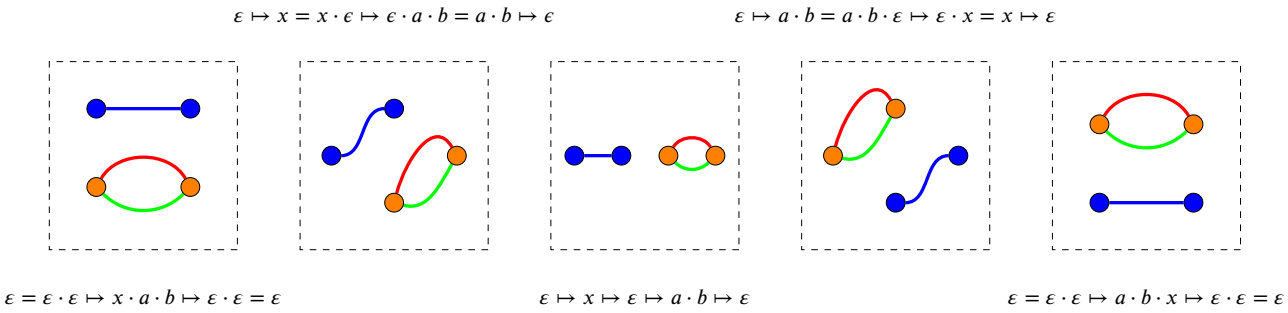


Figure 2.14: In our analogy with string rewrite systems, we might like that the following rewrites are equivalent, while respecting that they are not equal, representing x, a, b as blue, red, and green wires respectively. Such rewrites from the empty string to itself are more generally called *scalars* in the monoidal setting, viewed 2-categorically.

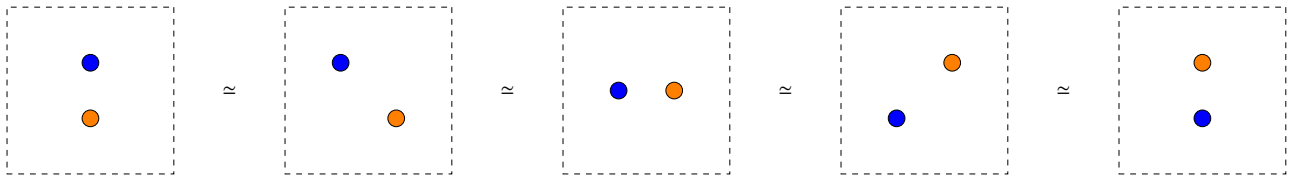


Figure 2.15: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument CITE is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting is that the empty string is equal to copies of itself, which creates the necessary space for manoeuvering; translating into the n-categorical setting, expressions are equivalent up to introducing and contracting identities.

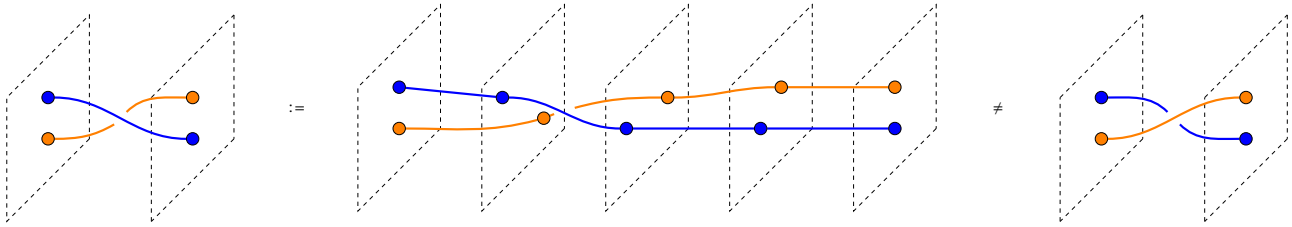


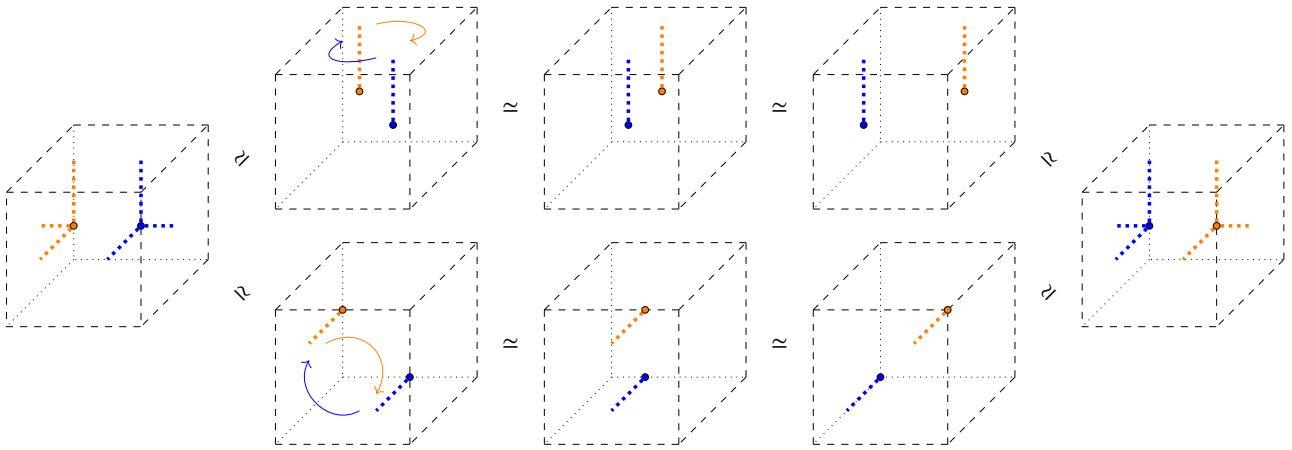
Figure 2.16: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies CITE , which are continuous bijective transformations that don't let wires double back on themselves, we can identify two different braidings that are not continuously deformable to one another in the 3-dimensional space of the vignette. We distinguish the braidings visually by letting wires either go over or under one another.

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

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

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

Figure 2.17: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise in the new dimension. Shown below are two ways to swap left-to-right sequentially composed dots by clockwise rotations in the forward-backward and up-down directions of composition:



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

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

Remark 2.1.6 (**THE IMPORTANT TAKEAWAY!**). Now have a combinatoric way to specify string diagrams that generalises PROPs for symmetric monoidal categories, which in addition permits us to reason with directed diagram rewrites. So we have a trinity of presentations of the same mathematical concept. String diagrams are the *syntax*. Symmetric monoidal categories are the *semantics*. n-categories are the *finite combinatoric presentation* of theories and rewrite systems upon them. By analogy, n-categories provide a *vocabulary* and *rewrites* for string-diagram *expressions*, which are interpreted with respect to the *context* of a symmetric monoidal category.

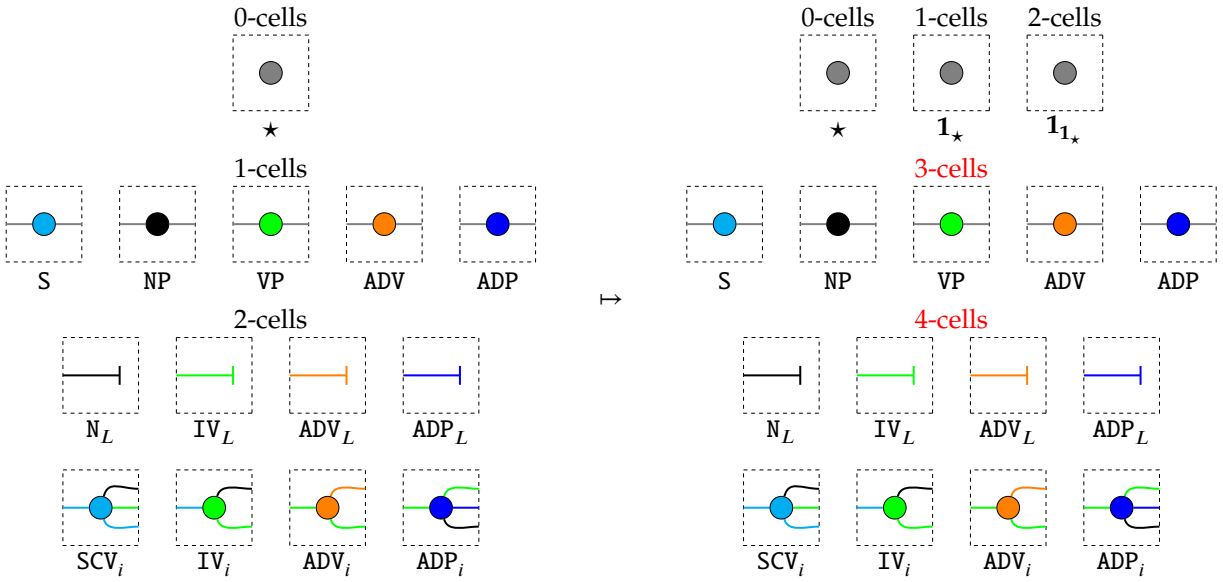


Figure 2.18: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with free swapping of symbol order.

2.1.5 *TAGs with links*

Definition 2.1.7 (TAGs with links: CS-style)**.** The following is a reproduction of the discussion on nodes with links from [Joshi]. Linking can be defined for any two nodes in an elementary tree. However, in the linguistic context we will require the following conditions to hold for a link in an elementary tree. If a node n_1 is linked to a node n_2 then:

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

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

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

Example 2.1.8. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n-categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out.

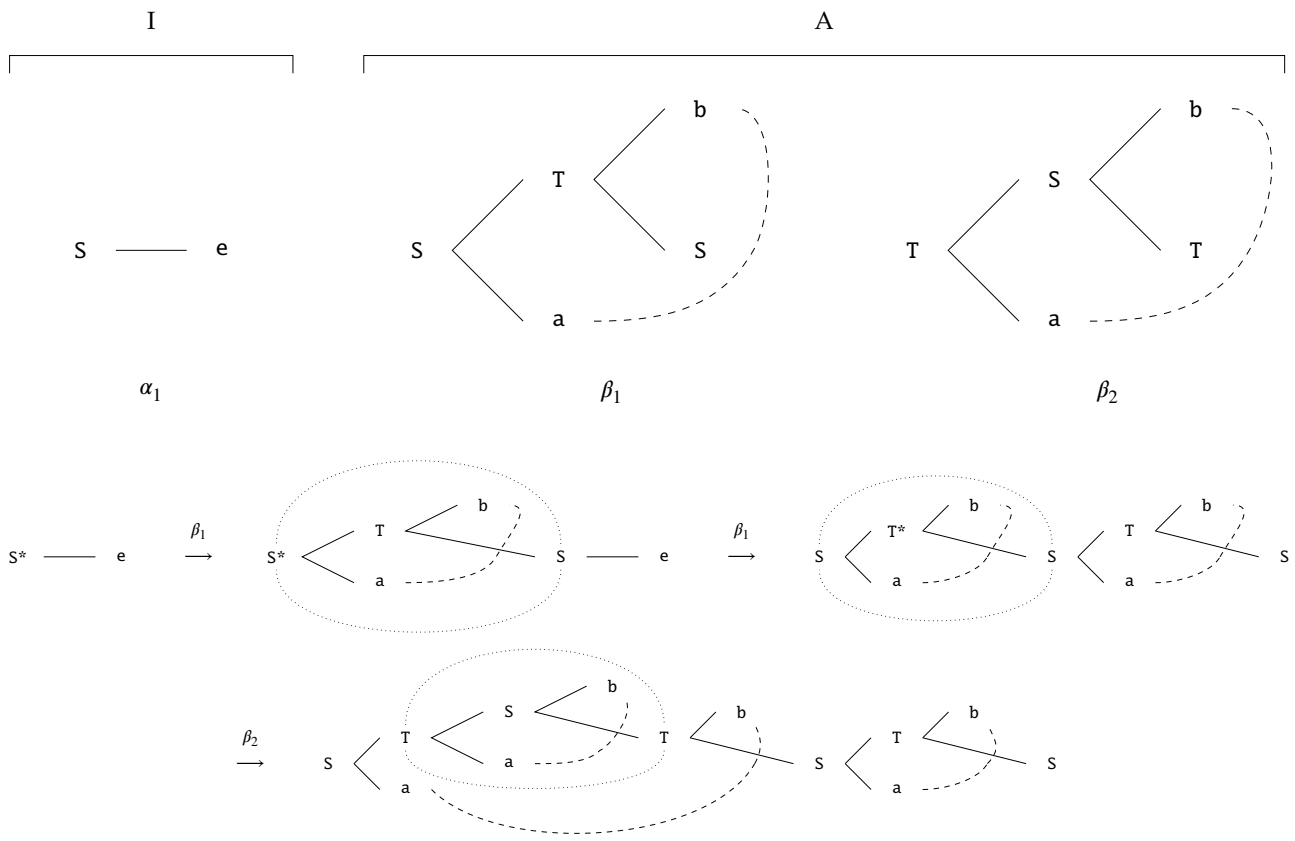


Figure 2.19: The TAG signature and example derivation are as above. Joshi stresses that adjoining *preserves* links, and that elementary trees may become *stretched* in the process of derivation, which are fundamentally topological constraints, akin to the "only (processive) connectivity" matters criterion identifying string diagrams up to isomorphism. Moreover, TAGs evidently have links of two natures: tree edges intended to be planar, and dashed dependency edges intended to freely cross over tree edges. It is easy, but a hack, to ask for planar processive isomorphisms for tree edges and extraplanar behaviour for dependency edges: these are evidently two different kinds of structure glued together, rather than facets of some whole. Weak n-categories offer a unified mathematical framework that natively accommodates the desired topological constraints while also granting expressive control over wire-types of differing behaviours. One method to recover TAGs true to the original conception is to stay in a planar 1-object-2-category setting while explicitly including wire-crossing cells for dependency links. The alternative method we opt for in Section ref is to work in a pure "only connectivity matters" setting, recovering the linear ordering of words by generating cells along a chosen wire. I do not know of any conceptual justification for why planarity is so often an implicit constraint in approaches to formal syntax. My best guesses are either that the first port of call for rewrites between 1-dimensional strings of words is a 2-dimensional setting, or it is a limitation of 2-dimensional paper as a medium of thought along with some confusion of map for territory.

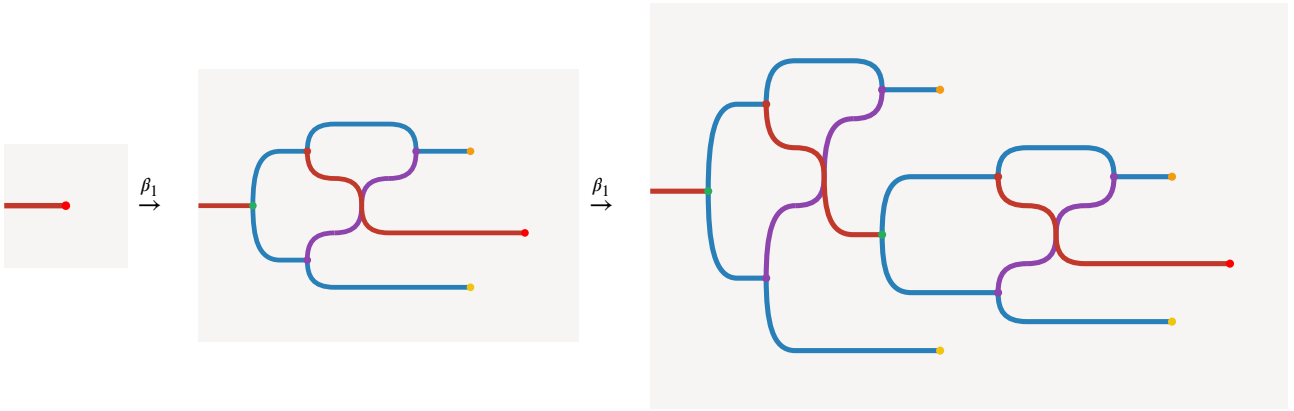


Figure 2.20: With our interpretation of TAGS as weak n-categorical signatures, We can recover each step of the example derivation automagically in `homotopy.io`; just clicking on where we want rewrites allows the proof assistant to execute a typematching tree adjunction. In the process of interpretation, we introduce a link wire-type (in purple), and include directed link generation and elimination morphisms for the T wire-type (in blue). A necessary step in the process of interpretation (which for us involves taking a Poincaré dual to interpret nodes as wires) is a typing assignment of the tree-branches connected to terminal nodes, which we have opted to read as sharing a T-type for minimality, though we could just as well have introduced a separate label-type wire.

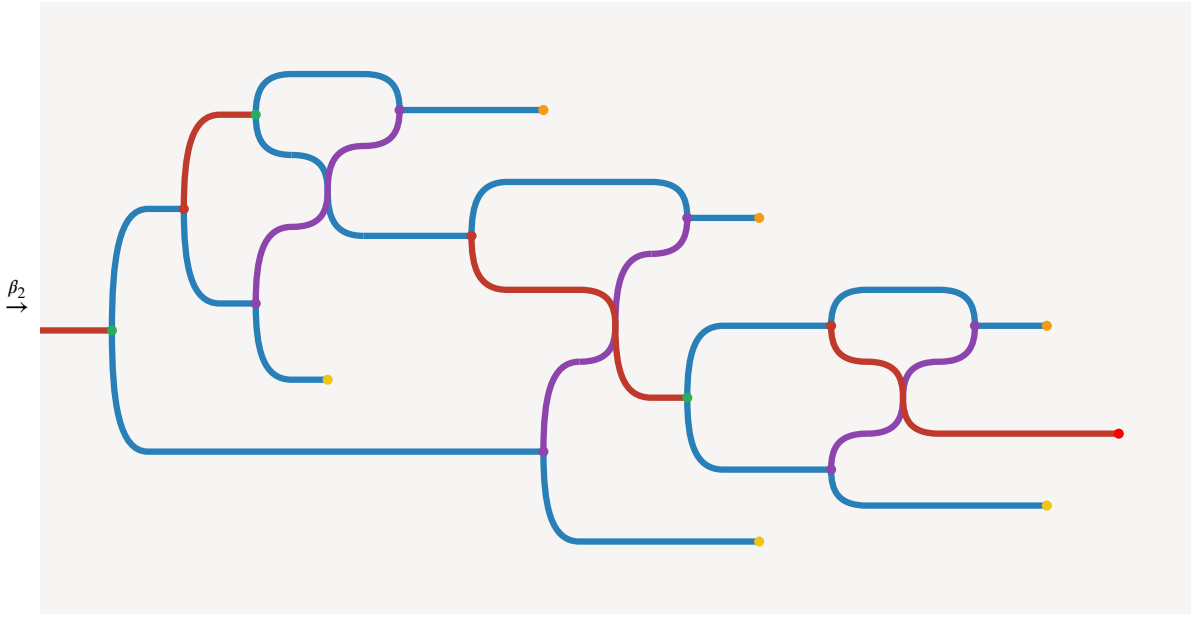


Figure 2.21: The intended takeaway is that even if you don't buy the necessity or formality of weak n-categories, there is always the fallback epistemic underpinning of a formal proof assistant for higher dimensional rewriting theories, which is rather simple to use if I have succeeded in communicating higher-dimensional intuitions in this section. **N.B.** In practice when using `homotopy.io` for the symmetric monoidal setting, it is simpler to suspend symmetric monoidal signatures to begin at 4-cells rather than 3-cells. The reason for this is that under- and over-braids still exist in the symmetric monoidal setting, and while sequentially composed braids are homotopically equivalent to the pair of identities, they are not uniquely so, thus these homotopies must be input manually. By beginning at 4-cells (or higher, due to the stabilisation hypothesis CITE), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

Definition 2.1.9 (Linguistic Tree Adjoining Grammars)**.** A *Linguistic Tree Adjoining Grammar* is given by the following data:

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

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

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

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

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

2.1.6 *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 2.1.10 (TAGs in `homotopy.io`). We spell out how the data of a TAG becomes an n-categorical signature by enumerating cell dimensions:

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

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

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

2.2 *A generative grammar for text circuits*

2.2.1 *A circuit-growing grammar*

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

Definition 2.2.1 (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*

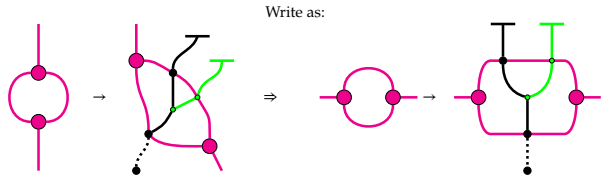


Figure 2.22: **How to read the diagrams in this section:** we will be making heavy use of pink and purple bubbles as frames to construct circuits. We will depict the bubbles horizontally, as we are permitted to by compact closure, or by reading diagrams with slightly skewed axes.

Simplifications and limitations: For now we only consider word types as in Definition 2.2.1, though we will see how to engineer extensions later. We only deal with propositional statements, without determiners, in only one tense, with no morphological agreement between nouns and their verbs and referring pronouns, and we assume that adverbs, adjectives stack indefinitely and without further order requirements: e.g. `Alice happily secretly finds red big toy shiny car that he gives to Bob` is a sentence we consider grammatical enough. For now, we consider only the case where adjectives and adverbs appear before their respective noun or verb. Note that all of these limitations apart from the limited lexicon can principle be overcome by the techniques we developed in Section 2.1.6 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 2.23. The minor distinction is that rewrite rules are sensitive to twists in

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

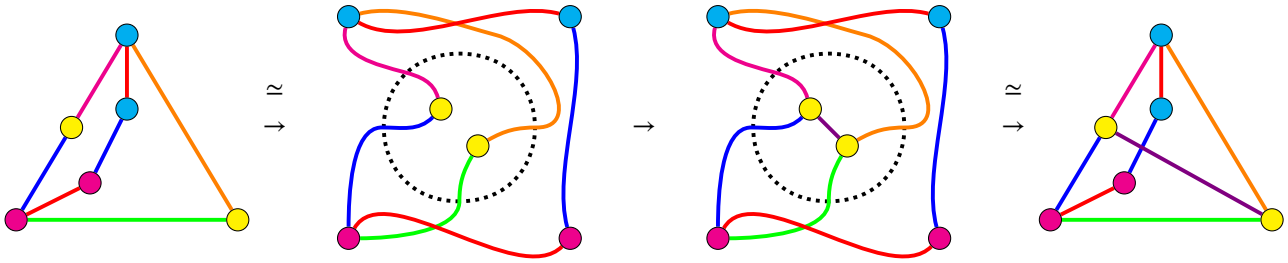
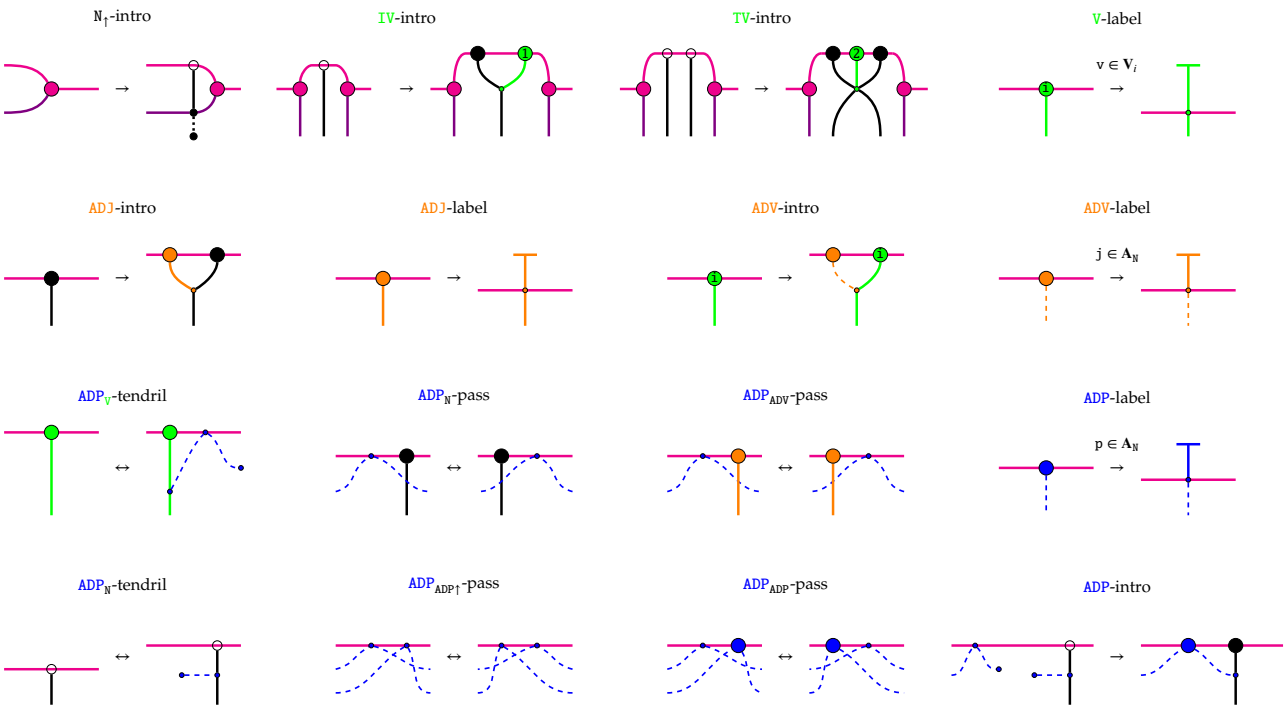


Figure 2.23: In this toy example, obtaining the same rewrite that connects the two yellow nodes with a purple wire using only graph-theoretically-local rewrites could potentially require an infinite family of rules for all possible configurations of pink and cyan nodes that separate the yellow, or would otherwise require disturbing other nodes in the rewrite process. In our setting, strong compact closure homotopies handle navigation between different spatial presentations so that a single rewrite rule suffices: the source and target notated by dotted-black circles. Despite the expressive economy and power of finitely presented signatures, we cannot "computationally cheat" graph isomorphism: formally we must supply the compact-closure homotopies as part of the rewrite, absorbed and hidden here by the $\simeq$ notation.

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

2.2.2 *Simple sentences*

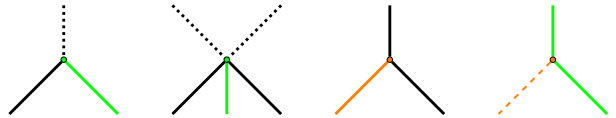
Simple sentences are sentences that only contain a single intransitive or transitive verb. Simple sentences will contain at least one noun, and may optionally contain adjectives, adverbs, and adpositions. The rules for generating simple sentences are as follows:



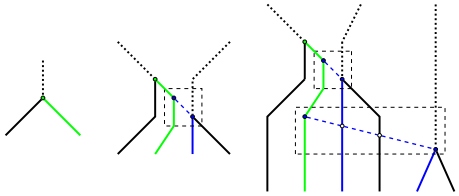
The $\mathtt{N}_\uparrow$-intro rule introduces new unsaturated nouns from the end of a simple sentence. The IV-intro rule applies when there is precisely one unsaturated noun in the sentence, and the TV-intro rule applies when there are precisely two. Both verb-introduction rules saturate their respective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

Definition 2.2.2 (CSG for simple sentences)**.** We may gauge the expressivity of simple sentences with the following context sensitive grammar.

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

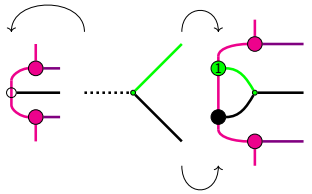


Adpositions require several helper-generators; we depict for example the beginning of the sequence of derivations that result from appending adpositions to an intransitive verb (the generators are implicit in the derivations):



Proposition 2.2.3. Up to labels, the simple-sentence rules yield the same simple sentences as the CSG for simple sentences.

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



□

2.2.3 *Complex sentences*

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

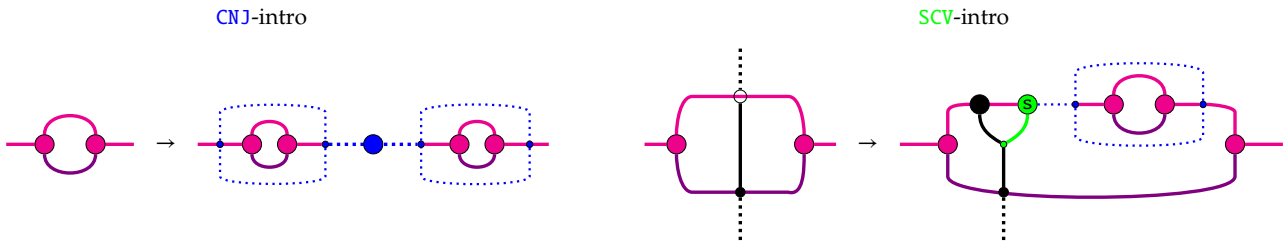
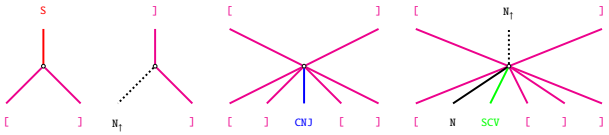


Figure 2.24: The dotted-blue wires do not contentfully interact with anything else, but this noninteraction disallows overgeneration cases where adpositional phrases might interject between SCV verbs and their sentential complement, e.g. `Alice sees at lunch Bob drink`. The dotted-blue wires also indicate a diagrammatic strategy for extensions to accommodate noun phrases, to be explored later.

Definition 2.2.4 (Sentence structure)**.** A sentence can be:

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

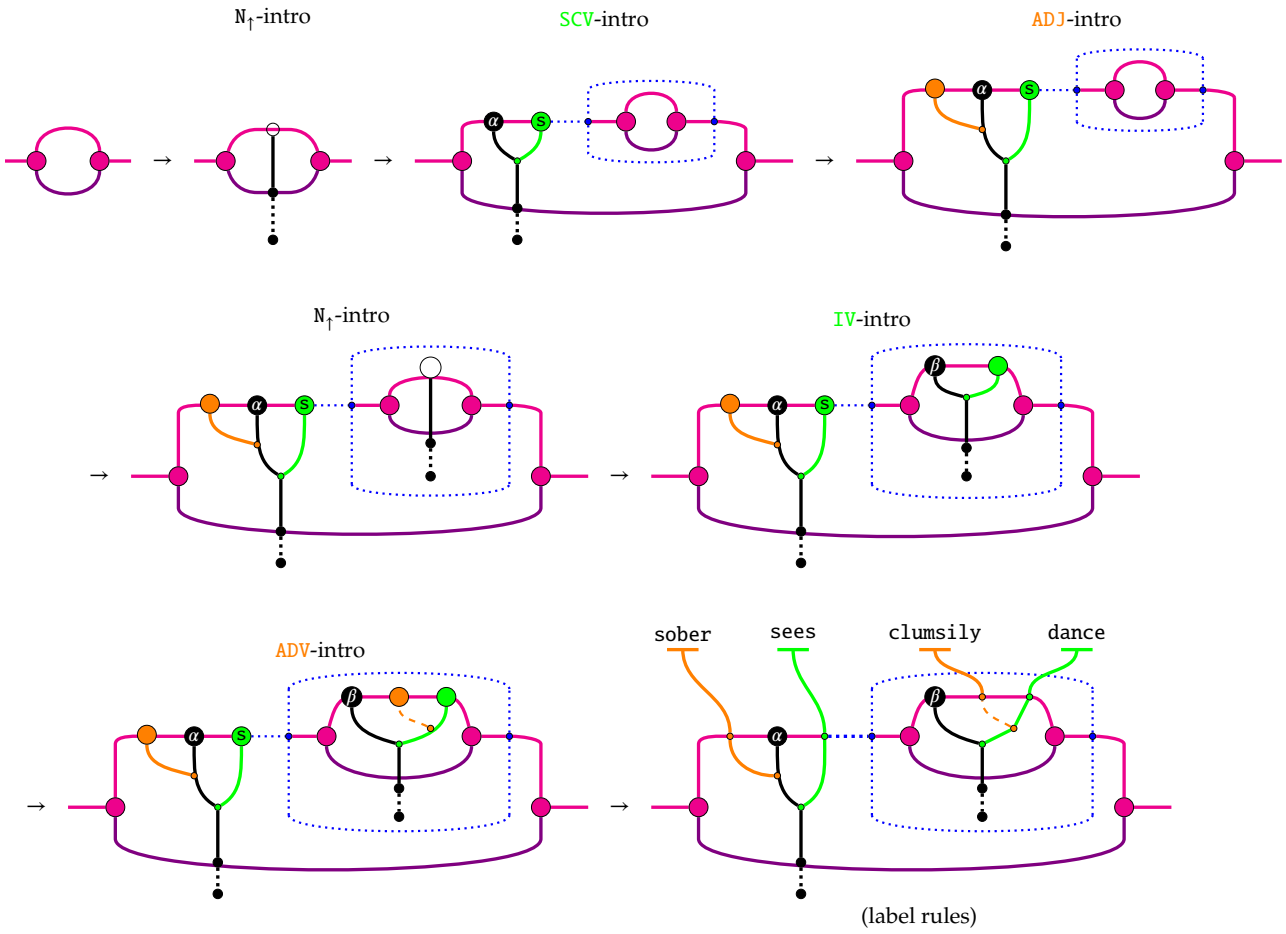
As a CSG, these considerations are respectively depicted as:



Proposition 2.2.5. Up to labels, the rules so far yield the same sentences as the combined CSG of Definitions 2.2.2 and 2.2.4.

Proof. Same correspondence as Proposition 2.2.3, ignoring the dotted-blue guards. □

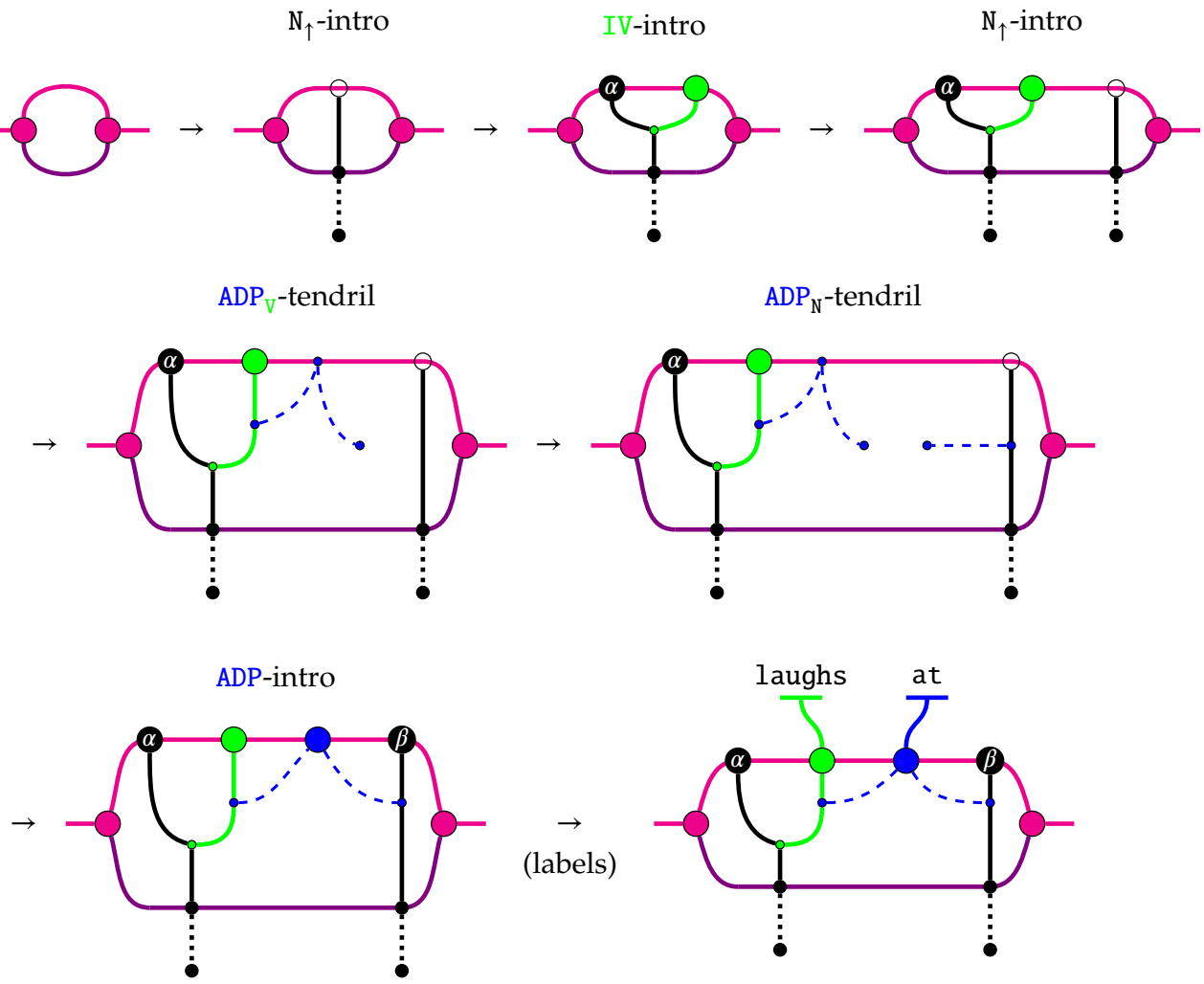
Figure 2.25:



Example 2.2.6 (`sober` α `sees drunk` β `clumsily dance.`). Now we can see our rewrites in action for sentences. As a matter of convention – reflected in how the various pass- rules do not interact with labels – we assume that labelling occurs after all of the words are saturated. We have still not introduced rules for labelling nouns: we delay their consideration until we have settled coreferential structure. For now they are labelled informally with greeks.

Figure 2.26:

Example 2.2.7 (α `laughs at` β)**.** Adpositions form by first sprouting and connecting tendrils under the surface. Because the tendril- and pass- rules are bidirectional, extraneous tendrils can always be retracted, and failed attempts for verbs to find an adpositional unsaturated noun argument can be undone. Though this seems computationally wasteful, it is commonplace in generative grammars to have the grammar overgenerate and later define the set of sentences by restriction, which is reasonable so long as computing the restriction is not computationally hard. In our case, observe that once a verb has been introduced and its argument nouns have been saturated, only the introduction of adpositions can saturate additionally introduced unsaturated nouns. Therefore we may define the finished sentences of the circuit-growing grammar to be those that e.g. contain no unsaturated nodes on the surface, which is a very plausible linear-time check by traversing the surface.



2.2.4 *Text structure and noun-coreference*

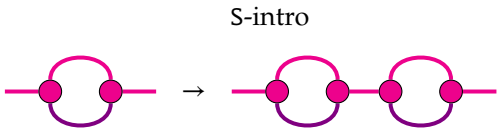


Figure 2.27: Only considering words, text is just a list of sentences. However, for our purposes, text additionally has *coreferential structure*. Ideally, we would like to connect "the same noun" from distinct sentences as we would circuits.

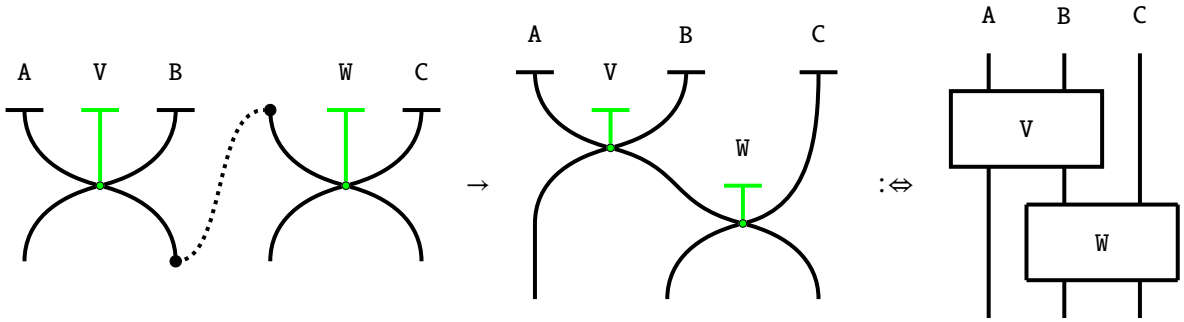


Figure 2.28: We choose the convention of connecting from left-to-right and from bottom-to-top, so that we might read circuits as we would text: the components corresponding to words will be arranged left-to-right and top-to-bottom. Connecting nouns across distinct sentences presents no issue, but a complication arises when connecting nouns within the same sentence as with reflexive pronouns e.g. `Alice likes herself`.

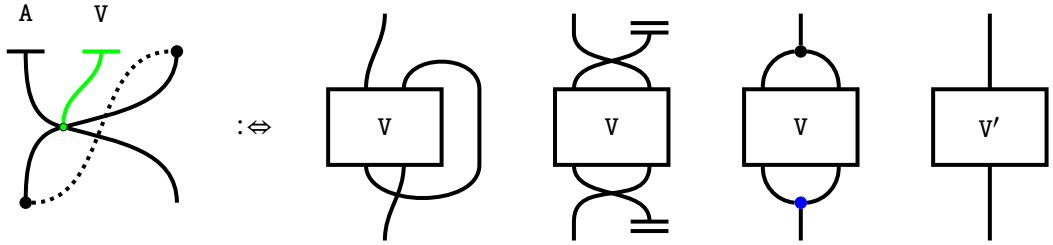
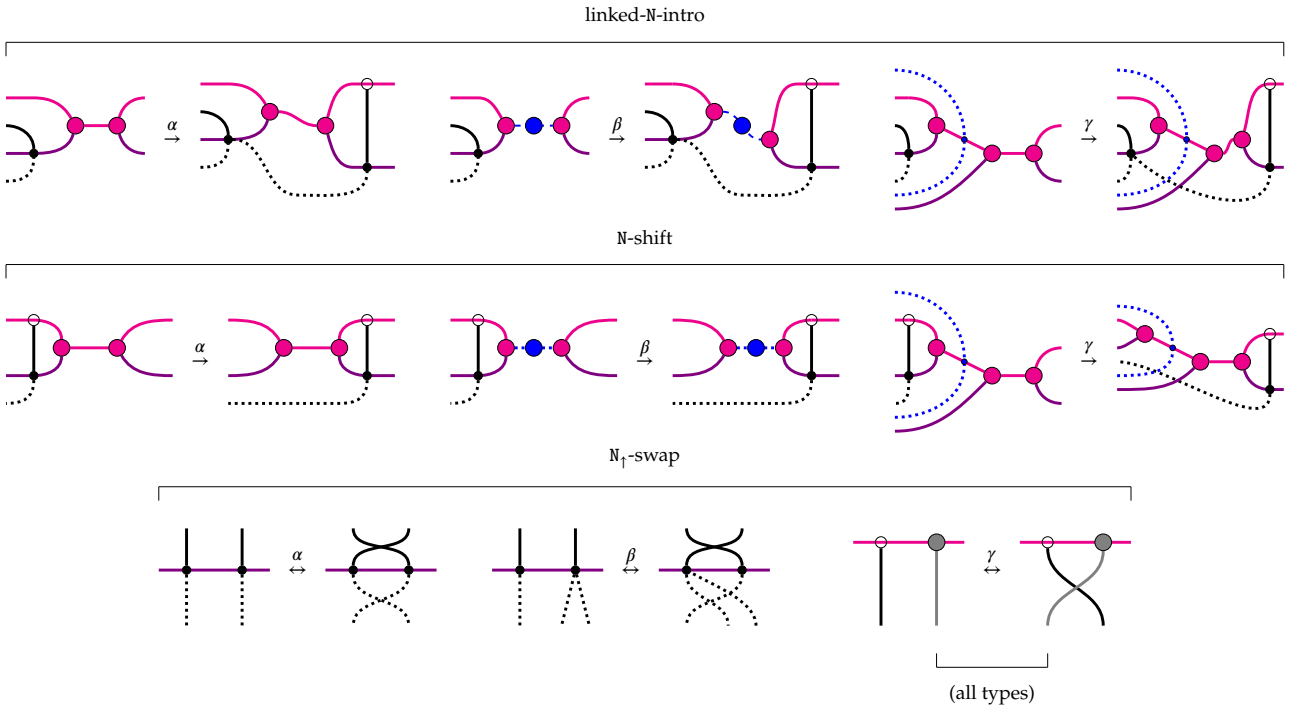


Figure 2.29: Reflexive coreference would violate of the processivity condition of string diagrams for symmetric monoidal categories. Not all symmetric monoidal categories possess the appropriate structure to interpret such reflexive pronouns, but there exist interpretative options. From left to right in roughly decreasing stringency, compact closed categories are the most direct solution. More weakly, traced symmetric monoidal categories also suffice. If there are no traces, so long as the noun wire possesses a monoid and comonoid, a convolution works. If all else fails, one can just specify a new gate. We will define coreference structure to exclude such reflexive coreference and revisit the issue as an extension.

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



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

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

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

Example 2.2.8 (`sober Alice sees Bob clumsily dance.  She laughs at him.`).

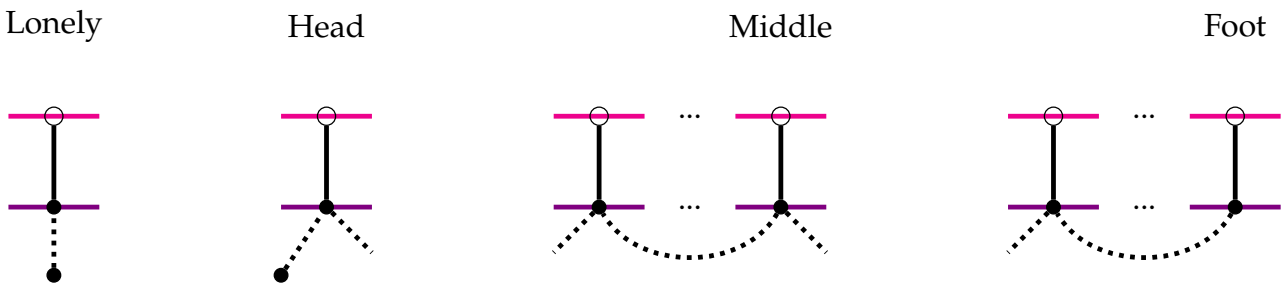


Figure 2.30: At this point, it is worth establishing some terminology about the kinds of unsaturated nouns we have in play. The kinds of nouns are distinguished by their tails. *Lonely* nouns have no coreferences, their tails connect to nothing. *Head* nouns have a forward coreference in text; they have two tails, one that connects to nothing and the other to a noun later in text. *Middle* nouns have a forward and backward coreference; they have two tails, one that connects to a noun in some preceding sentence, and one that connects forward to a noun in a succeeding sentence. *Foot* nouns only have a backward coreference; they have a single tail connecting to a noun in some preceding sentence.

2.2.5 *Text circuit theorem*

Definition 2.2.9 (Text Circuits)**.** *Text circuits* are made up of three ingredients:

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

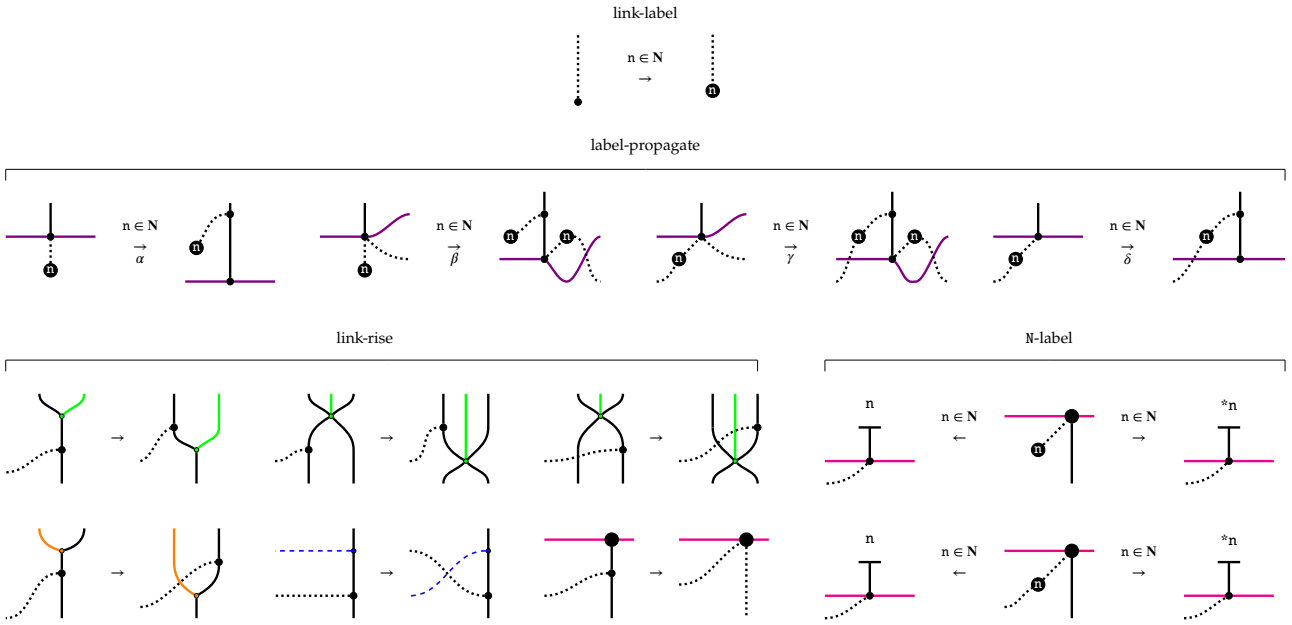


Figure 2.31: The $n \in \mathbf{N}$ notation indicates a family of rewrites (and generators) for each noun in the lexicon. Link-label assigns a noun to a diagrammatically linked collection of coreferent nouns, and link-propagation is a case analysis that copies a link label and distributes is across coreferent nouns. Link-rise is a case analysis to connect labels to the surface, and finally N-label allows a saturated noun to inherit the label of its coreference class, which may either be a noun n or a pronoun appropriate for the noun, notated *n

Figure 2.32: We start the derivation by setting up the sentence structure using S- and SCV-intro rules, and two instances of N-intro, one for Alice, and one for Bob. Observe how the N-intro for Bob occurs within the sub-sentence scoped over by the SCV-rule.

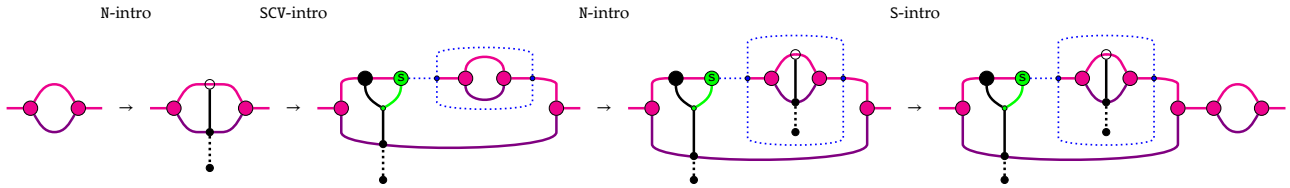


Figure 2.33: By homotopy, we can rearrange the previous diagram to obtain the source of the linked-N-intro rewrite in the dashed-box visual aid. Observe how we drag in the root of what is to be Alice's wire. Then we use the IV-intro in the second sentence, which sets up the surface structure `she laughs`, and the deep structure for bookkeeping that `she` refers to `Alice`.

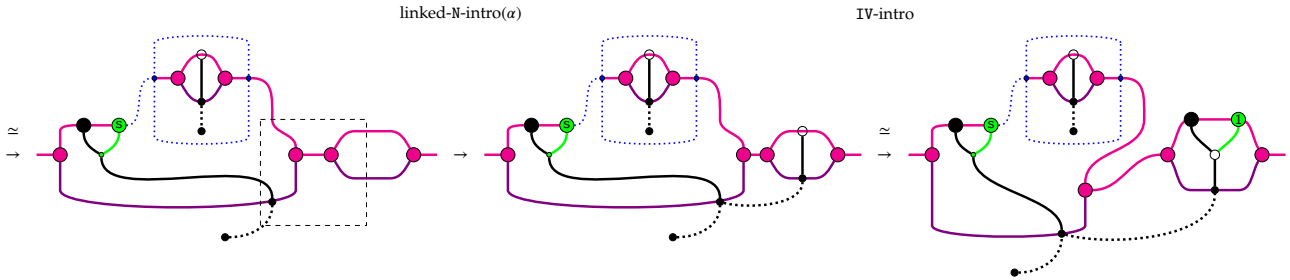


Figure 2.34: By homotopy again, we can do the same for Bob, this time setting up for the γ variant of linked-N-intro which handles the case when the spawning noun is within the scope of an SCV. Then by applying a series of $\mathtt{N}_{\uparrow}$-swaps, the unsaturated noun is placed to the right of the intransitive verb phrase.

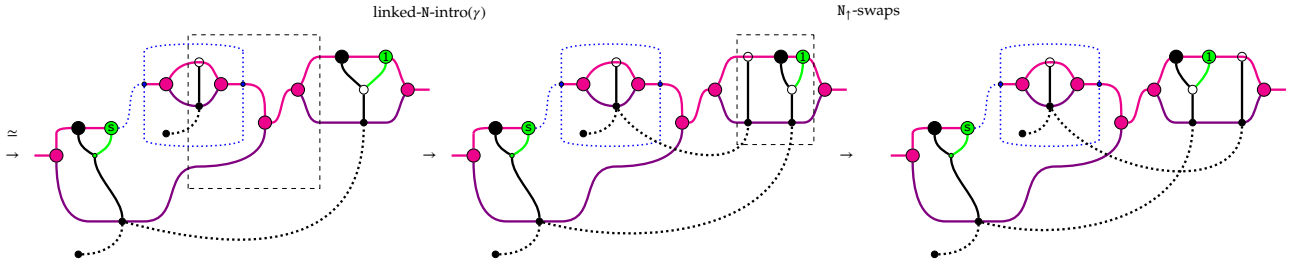
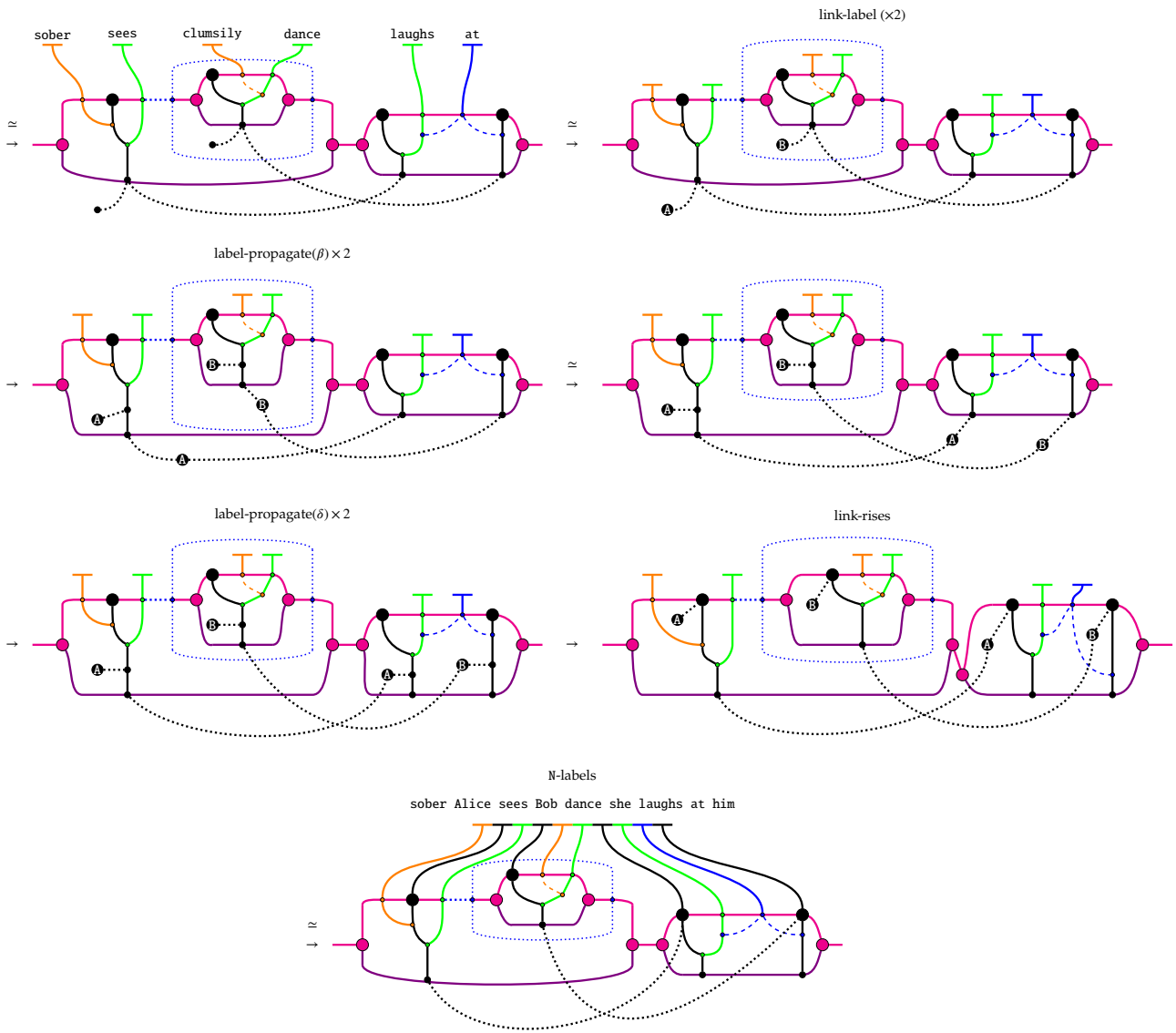


Figure 2.35: We've already done the surface derivation for the two sentences separately in Figures 2.25 and 2.26; since neither of those derivations touch the roots of noun-wires, we can emulate those derivations and skip ahead to the first diagram.



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

Definition 2.2.10 (Finished text diagram)**.** The circuit-growing grammar produces *text diagrams*. We call a text diagram *finished* if all surface nodes are labelled.

Proposition 2.2.11. Finished text diagrams yield text, up to interpreting distinct sentences as concatenated with punctuation `.`, `,`, contentless conjunctions or complementisers – such as `and`, or `that` respectively.

Proof. Sentence-wise grammaticality is gauged by Propositions 2.2.3 and 2.2.5. When multiple sentences occur within the scope of a `SCV` we might prefer the use of contentless complementisers and conjunctions, e.g. `Alice sees( Bob draws Charlie drinks ) Dennis dances` is grammatically preferable but meaningfully equivalent to `Alice sees that Bob draws and Charlie drinks , and Dennis dances .` For our purposes it makes no difference whether surface text has these decorations, as the deep structure of text diagrams encodes all the information we care to know. □

Lemma 2.2.12. The unsaturated noun kinds listed in Figure 2.30 are exhaustive, hence nouns that share a coreference are organised as a diagrammatic linked-list.

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

Lemma 2.2.13. No nouns within the same sentence are coreferentially linked.

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

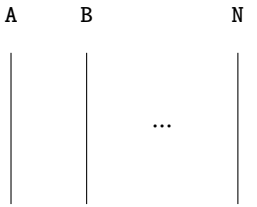


Figure 2.36: Nouns are represented by wires, each 'distinct' noun having its own wire.

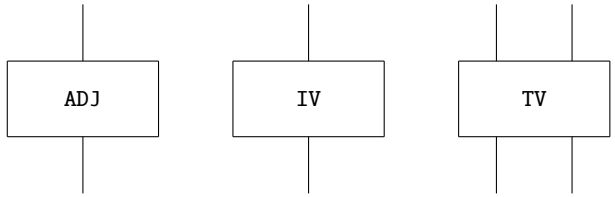


Figure 2.37: We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires. Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

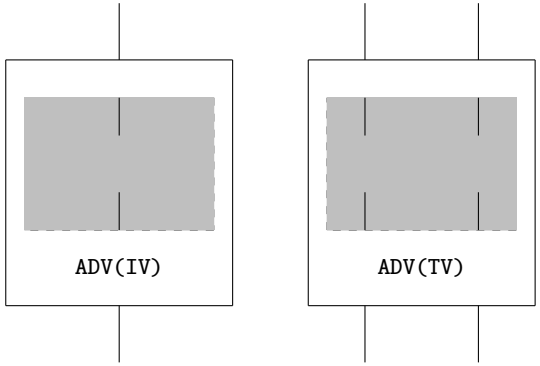


Figure 2.38: Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicating the shape of gate expected, and these should match the input- and output-wires of the box with the whole.

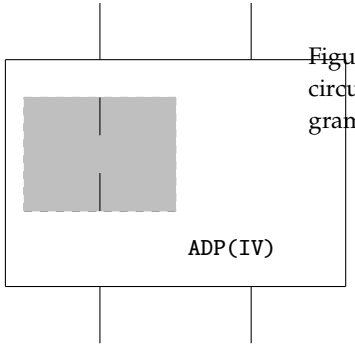


Figure 2.39: Similarly, adpositions also modify verbs, by moreover adding another noun-wire to the right.

Construction 2.2.14 (Text to circuit)**.**

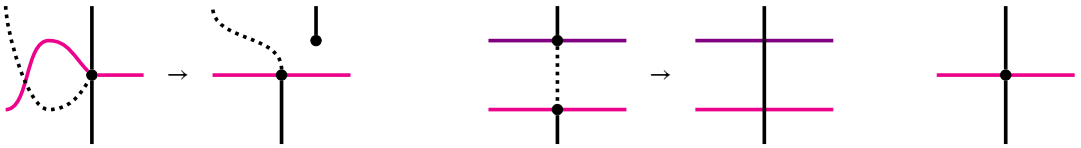


Figure 2.44: We turn finished text diagrams into text circuits by operating *in situ*, with extra rules outside the grammatical system that handle connecting noun wires.

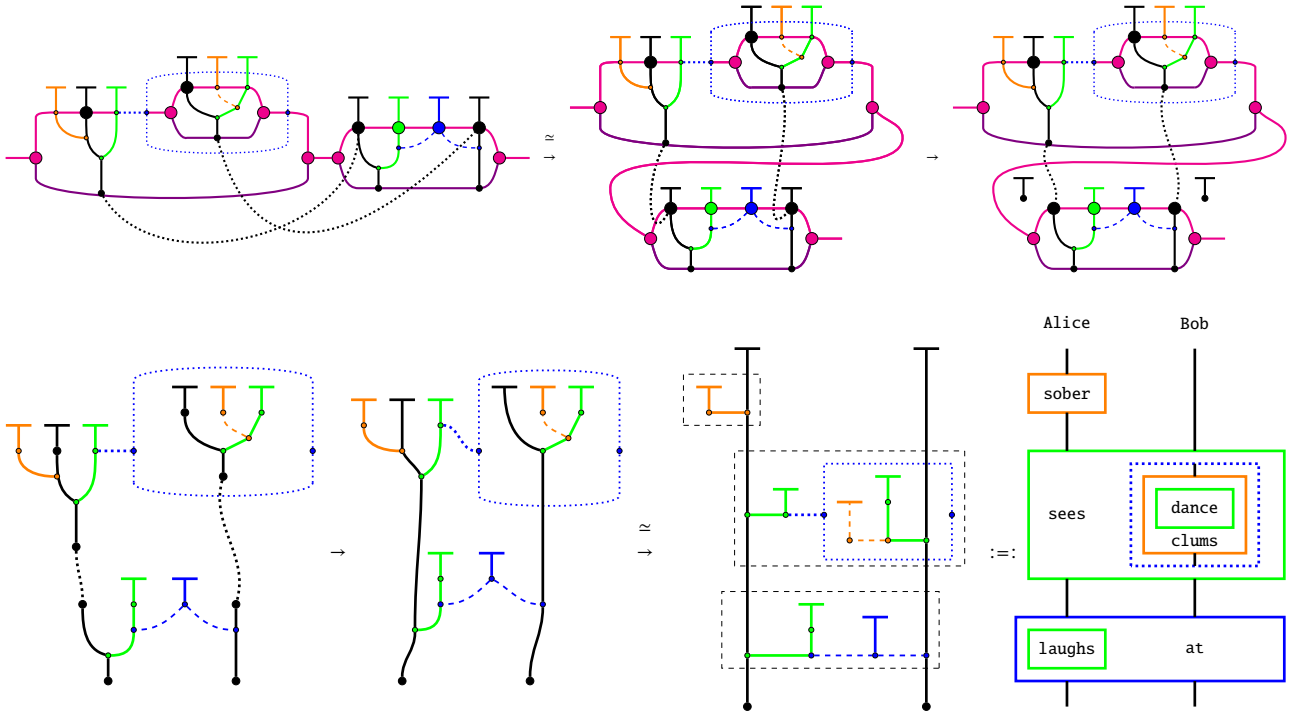


Figure 2.45: In the first step, by Lemmas 2.2.12 and 2.2.13, we can always rearrange a finished text diagram such that the noun wires are processive.

In the second step, use the first rewrite of Construction 2.2.14 to prepare the wires for connection.

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

In the fourth step, we apply the second and third rewrites of Construction 2.2.14 to connect the wires and eliminate nodules underneath labels. We can also straighten up the wires a bit and make them look proper.

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

Figure 2.46: For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit.

Proposition 2.2.15 (Finished text diagrams yield unique-up-to-processive-isotopy text circuits)**.** *Proof.* Every sentence corresponds to a gate up to notation, and we have a handle on sentences via Propositions 2.2.3 and 2.2.5. Lemmas 2.2.12 and 2.2.13 guarantee processivity. Uniqueness-up-to-processive-isotopy is inherited: text diagrams themselves are already specified up-to-connectivity, which is strictly more general than processive isotopy. Therefore, for any circuit C obtained from a text diagram T by Construction 2.2.14, T can be

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

The converse of Proposition 2.2.15 would be that any text circuit that can be formed by the composition of symmetric monoidal categories and of plugging gates into boxes yields a text diagram. This would mean that text circuit composition is acceptable as a generative grammar for text. Establishing this converse requires elaboration of some conventions.

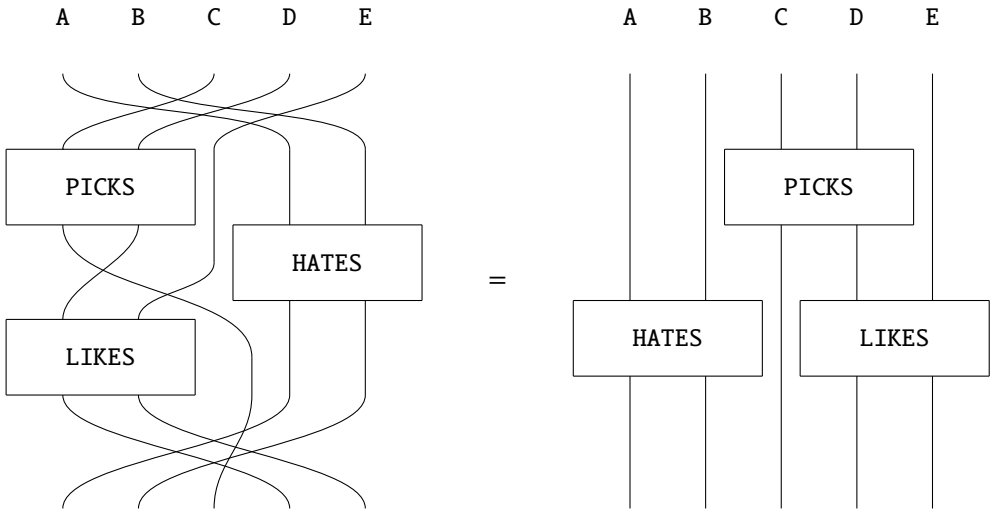


Figure 2.46:

Convention 2.2.17 (Wire twisting)**.**

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

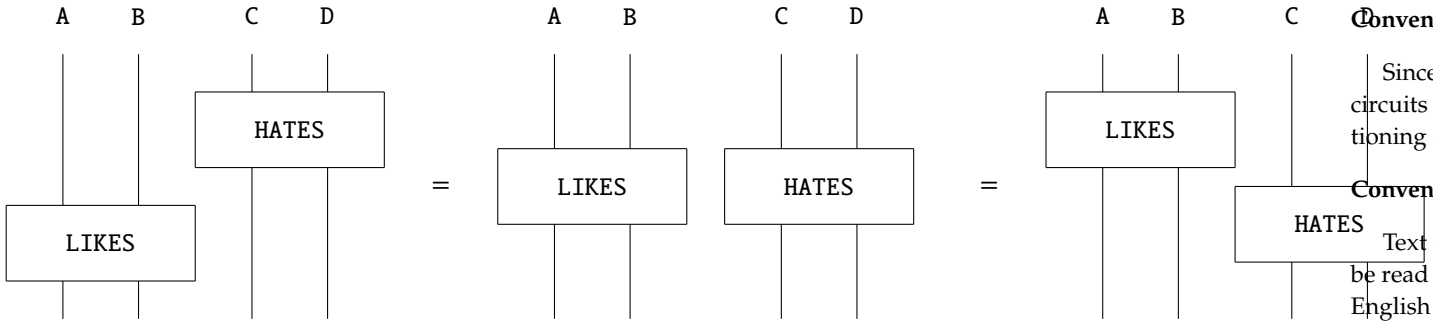


Figure 2.47:

Convention 2.2.18 (Sliding)**.**

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

Convention 2.2.19 (Reading text circuits)**.**

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

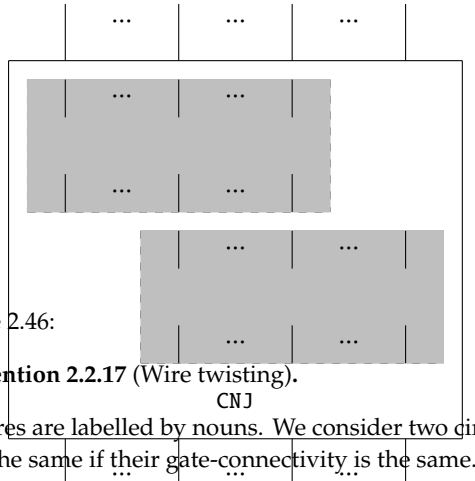


Figure 2.41: Conjunctions are boxes that take two circuits which might share labels on some wires.

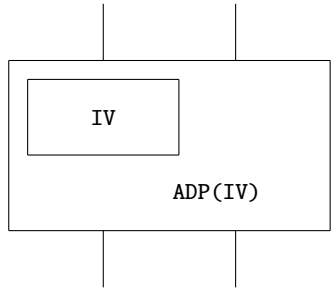


Figure 2.42: Of course filled up boxes are just gates.

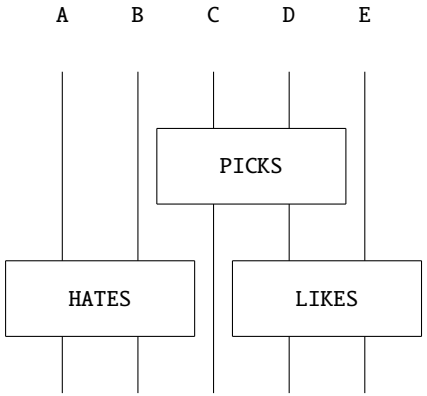
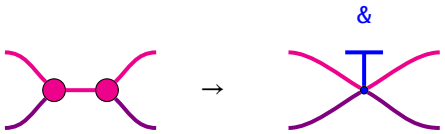


Figure 2.43: Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits.

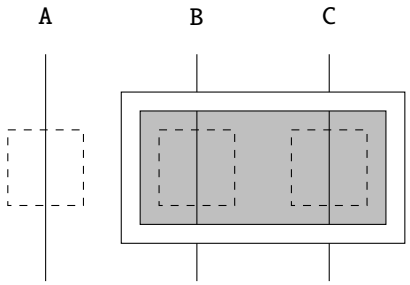
Remark 2.2.16. There are some oddities about our conventions that will make sense later when we consider semantics. For example, Convention 2.46 an acceptable thing to ask for syntactically but quite odd to think about at the semantic level, where we would like to think that distinct nouns manifest as different states on the same noun-wire-type. A semantic interpretation that makes use of this convention will become clearer in Section configspace . Similarly, Convention 2.2.22 wouldn't be true if we consider the order of text to reflect the chronological ordering of events, in which case there are implicit `...and then ...` conjunctions that distinguish ordered gates from parallel gates conjoined by an implicit `...while ...`; this particular complication is handled at the semantic level in Section statesactionmanner . The distinction in Convention 2.2.20 between typed and "untyped" higher-order processes will be given a suitable semantic interpretation in Section lassos .

Convention 2.2.20 (Arbitary vs. fixed holes)**.** Diagrammatically, adverbs and adpositions are depicted with no gap between the bounding box and their contents, whereas conjunctions and verbs with sentential complement are depicted with a gap; this is a visual indication that the former are type-sensitive, and the latter can take any circuit.

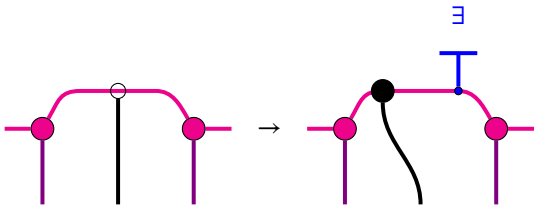
Convention 2.2.21 (Contentless conjunctions)**.** Conventions **??** and 2.47 require something else to allow them to work at the same time. The left text circuit may be read `C hates D. A likes B.`, and the right as `A likes B. C hates D.` The middle text circuit can be rewritten as either the left or right, which can be done if we're happy to make such choices. A choiceless approach requires something to distinguish the fact that the gates are parallel: a "contentless" conjunction, such as `and`, or `while`. In terms of text diagrams, we want a rewrites that introduces such contentless conjunctions and witnesses their associativity, like this:



Convention 2.2.22 (Lonely wires)**.** There's really only a single kind of text circuit we can draw that doesn't obviously correspond to a text diagram, and that's one where gates are missing.



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



Construction 2.2.23 (Circuit to text)**.** In the presence of additional rewrites from Conventions 2.2.21 and 2.2.22, every text circuit is obtainable from some text diagram, up to Conventions 2.46 and **??**.

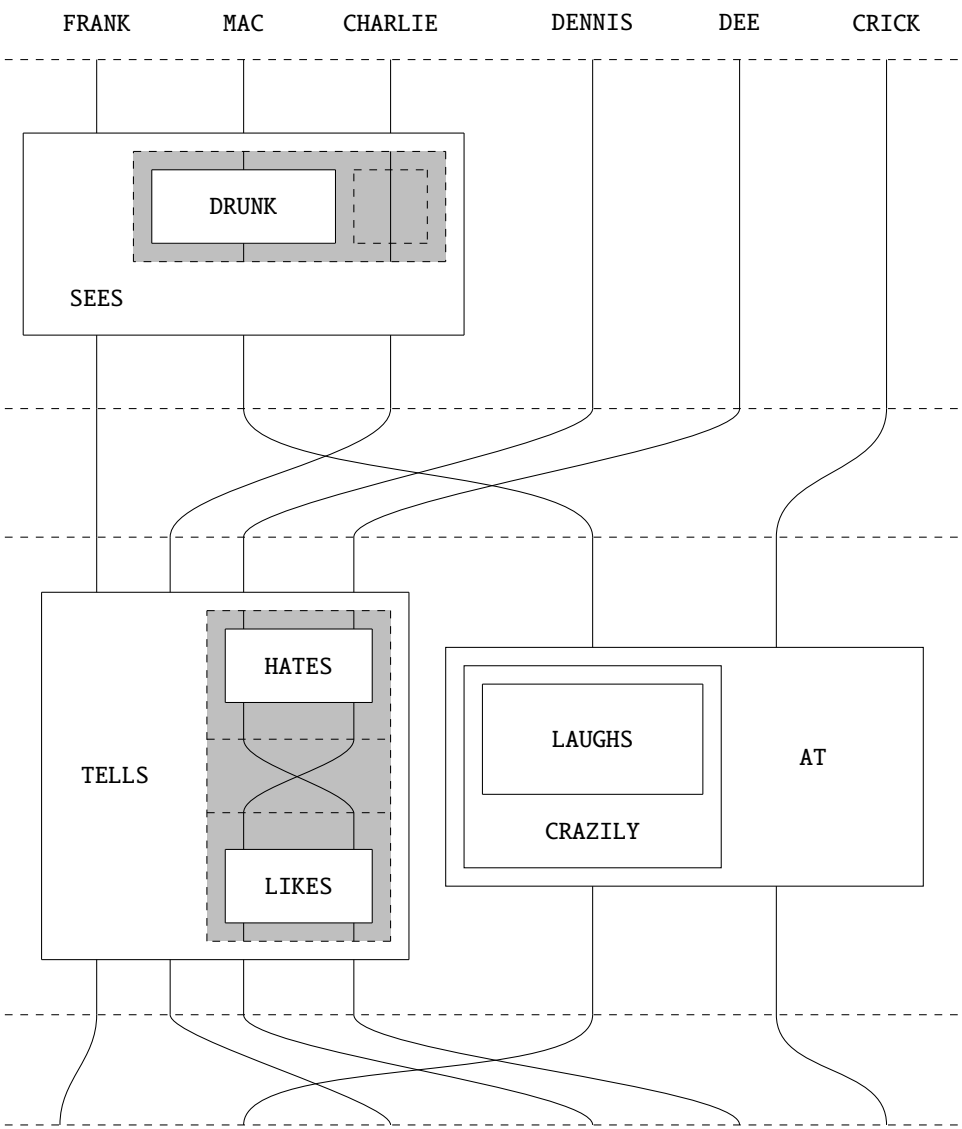


Figure 2.48: Starting with a circuit, we may use Convention 2.46 to arrange the circuit into alternating slices of twisting wires and (possibly tensored) circuits, and this arrangement recurses within boxes. Slices with multiple tensored gates will be treated using Convention 2.2.21. By convention 2.2.22, we decorate lonely wires with formal `exists` gates, as in the `Frank sees` box. Observe how verbs with sentential complement are depicted with grey gaps, whereas the adverb and adposition combination of `Mac crazily laughs at Cricket` is gapless, according to Convention 2.2.20.

TODO: insert text circuit theorem

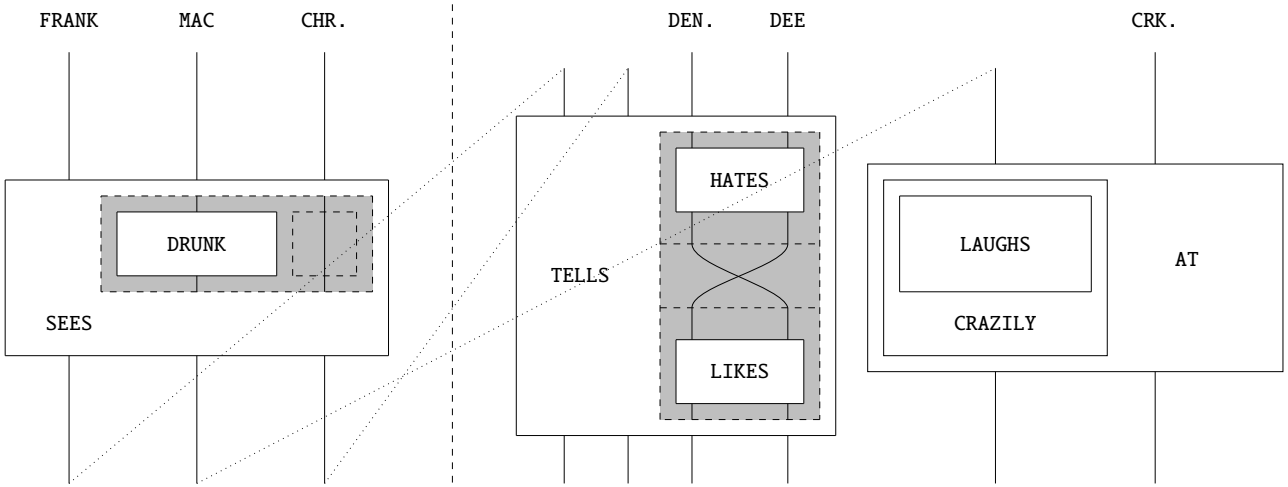


Figure 2.49: We then linearise the slices, representing top-to-bottom composition as left-to-right. Twist layers are eliminated, replaced instead by dotted connections indicating processive connectivity. The dashed vertical line distinguishes slices. This step of the procedure always behaves well, guaranteed by Proposition 2.2.12. Noun wires that do not participate in earlier slices can be shifted right until the slice they are introduced.

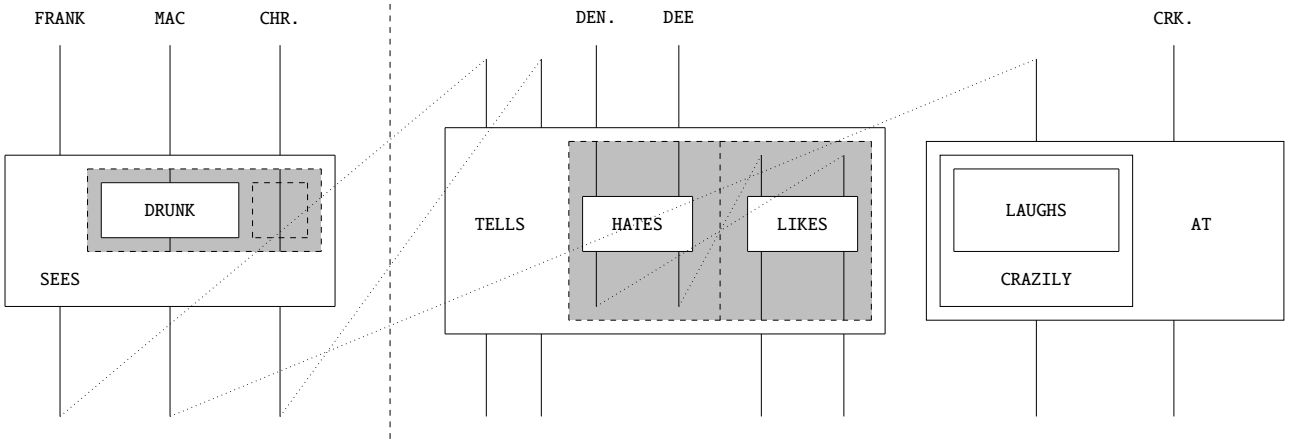


Figure 2.50: We recurse the linearisation procedure within boxes until there are no more sequentially composed gates. The linearisation procedure evidently terminates for finite text circuits. At this point, we have abstracted away connectivity data, and we are left with individual gates.

2.2.6 Extensions I: relative and reflexive pronouns

Relative pronouns

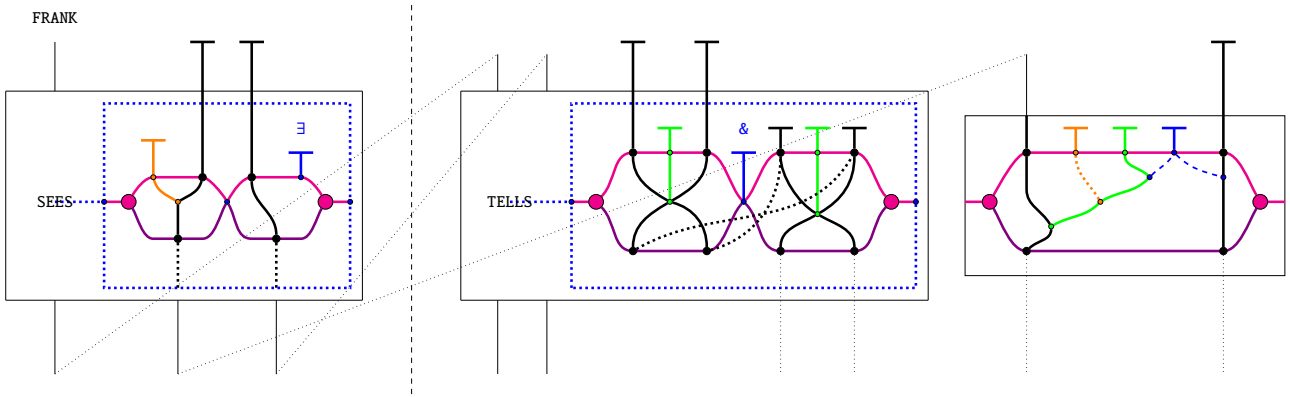


Figure 2.51: By Proposition 2.2.5, gates are equivalent to sentences up to notation, so we swap notations *in situ*. Conventions 2.2.21 and 2.2.22 handle the edge cases of parallel gates and lonely wires. Observe that the blue-dotted wiring in text diagrams delineates the contents of boxes that accept sentences.

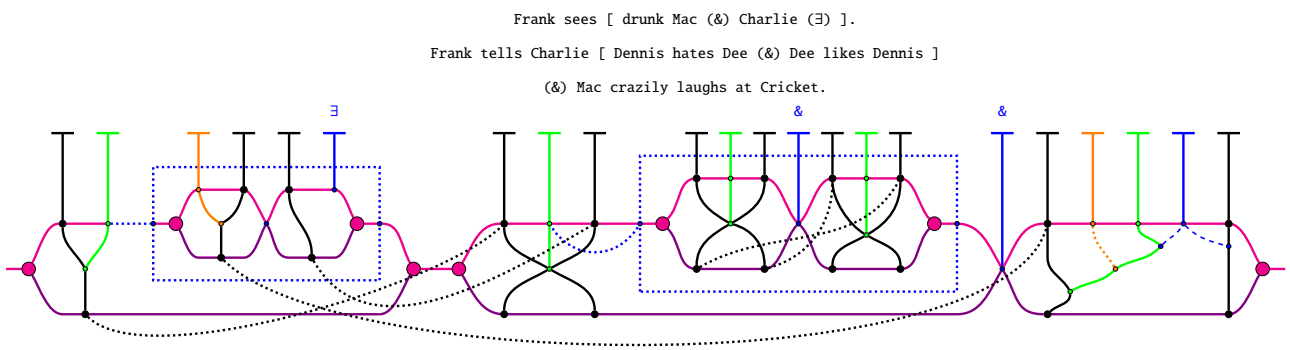
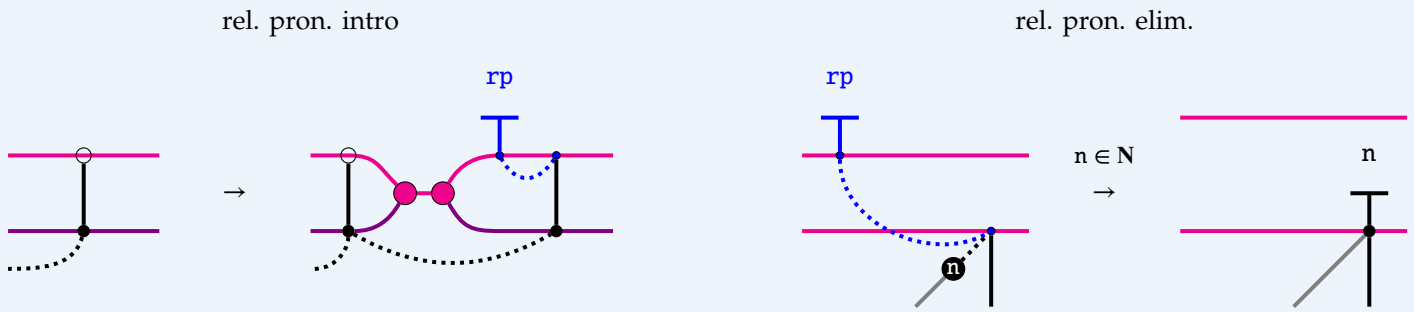


Figure 2.52: Recursing notation swaps outwards and connecting left-to-right slices as sentence-bubbles connect yields a text circuit, up to the inclusion of rewrites from Conventions 2.2.21 and 2.2.22: applying the reverse of those rewrites and the reverse of text-diagram rewrites yields a valid text-diagram derivation, by Propositions 2.2.5 and 2.2.12. We haven't formally included transitive verbs with sentential complement in our vocabulary, but it should be obvious at this point how they function with our existing machinery.

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

Reflexive pronouns are dealt with in a similar fashion as we have with relative pronouns, and various semantic interpretation options have been covered already in Figure 2.29.

Construction 2.2.24. We can extend our system to accommodate relative pronouns as follows. The introduction of a relative pronoun is considered to start a new sentence with a premade pronominal link. The relative pronoun connects to a new kind of surface noun, which for most intents and purposes behaves just like the nouns we have seen before, except for two caveats: it cannot be labelled (since the relative pronoun is already handling that), and it cannot leave the sentence it starts in by N-shift rules. We can eliminate relative pronouns by forcing the governed noun to be named, which is equivalent to dropping the relative pronoun and recovering distinct sentences.



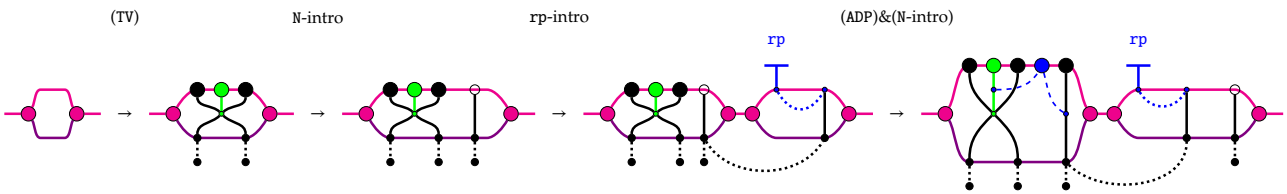


Figure 2.53:

Example 2.2.25 (Introducing relative pronouns)**.**

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

2.2.7 *Extensions II: grammar equations*

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

2.2.8 *Extensions III: Types and Nesting*

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

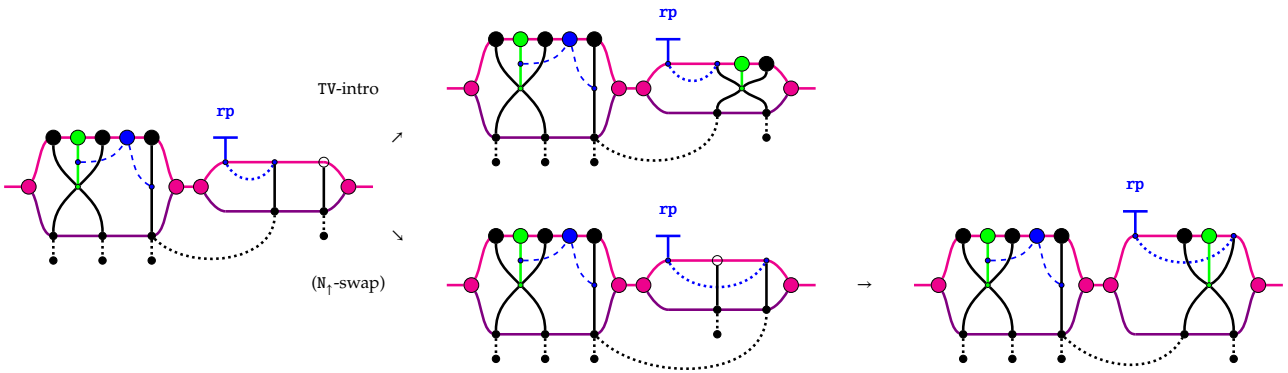


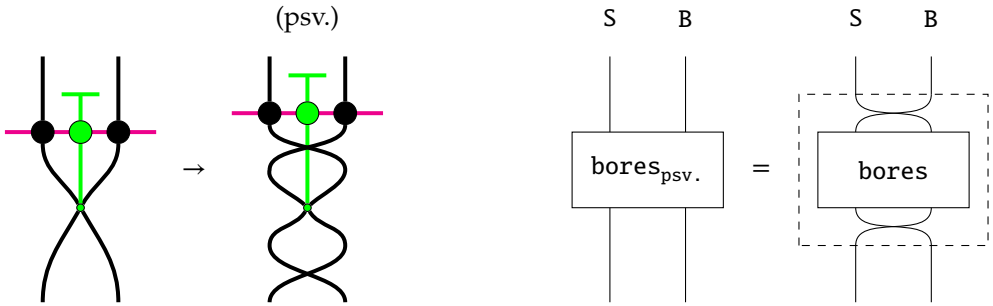
Figure 2.54: Now have a branching derivation. We may either directly generate a transitive verb treating the relative pronoun as a subject, or we may first perform an $\mathtt{N}_{\uparrow}$-swap first and then generate a transitive verb, treating the relative pronoun as an object. Now the ends of either branch can be labelled to recover our initial examples.

Figure 2.55:

Example 2.2.26 (Passive voice).

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

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



Intensifiers

Figure 2.56:

Example 2.2.27 (Copulas).

`Red car` = `Car is red`

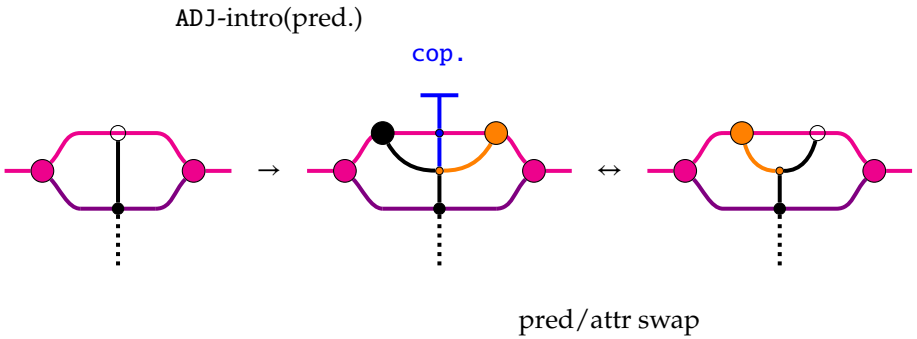
Modifiers such as adjectives and adverbs when they occur before their respective noun or verb are called *attributive*. When modifiers occur after their respective target, they are called *predicative*. In English, without the aid of `and`, only a single predicative modifier is permissible, e.g. `big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

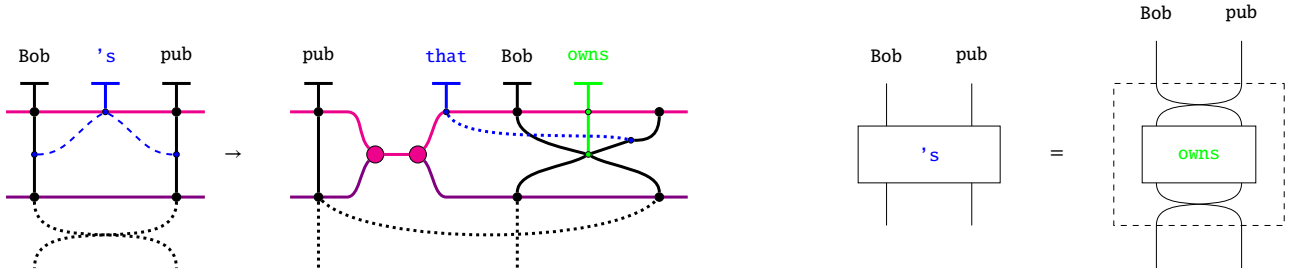
Figure 2.57:

Example 2.2.28 (Possessive pronouns).

`Bob's pub` = `Pub that Bob owns`

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





2.3 *Text circuits: details, demos, developments*

This section covers some practical developments, conventions, references for technical details of text circuits. The most striking demonstration to date is that circuits are defined over a large enough fragment of language to *leverage* several bAbi tasks CITE , which are a family of 20 general-reasoning text tasks – the italicised choice of wording will be elaborated shortly. Each family of tasks consists of tuples of text in simple sentences concluded by a question, along with an answer. It was initially believed that world models were required for the solution of these tasks, but they have been solved using transformer architectures CITE . 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

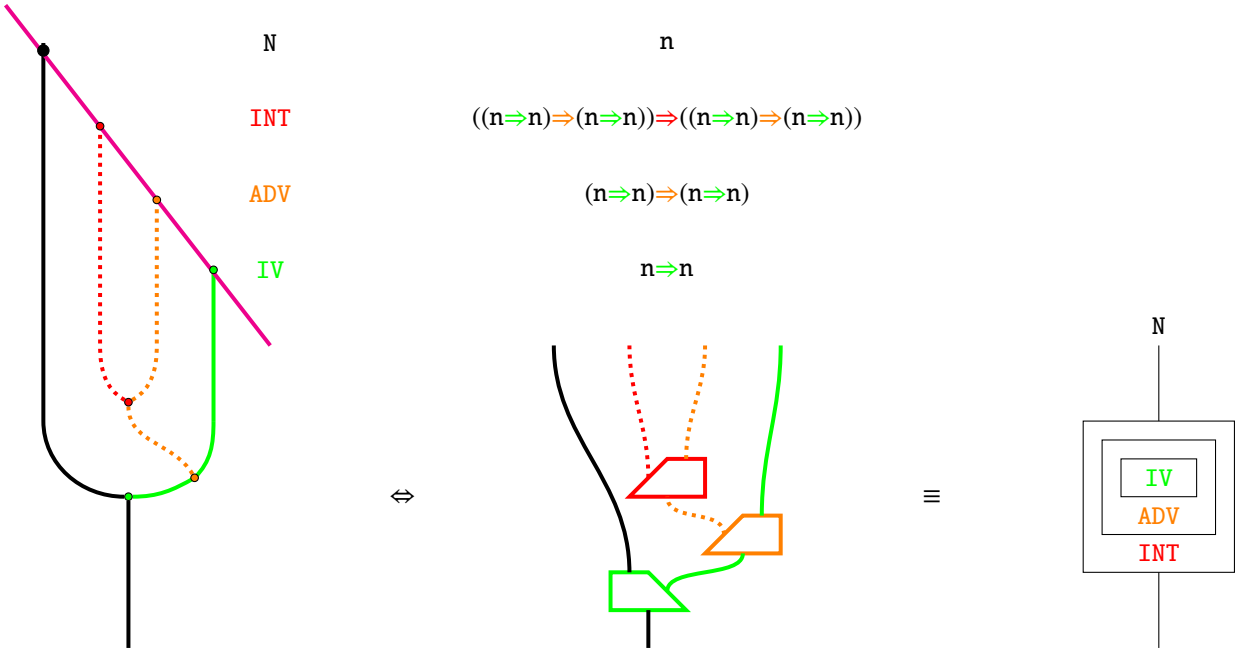


Figure 2.58:

Example 2.2.29 (Intensifers).

`Alice very quickly runs`

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

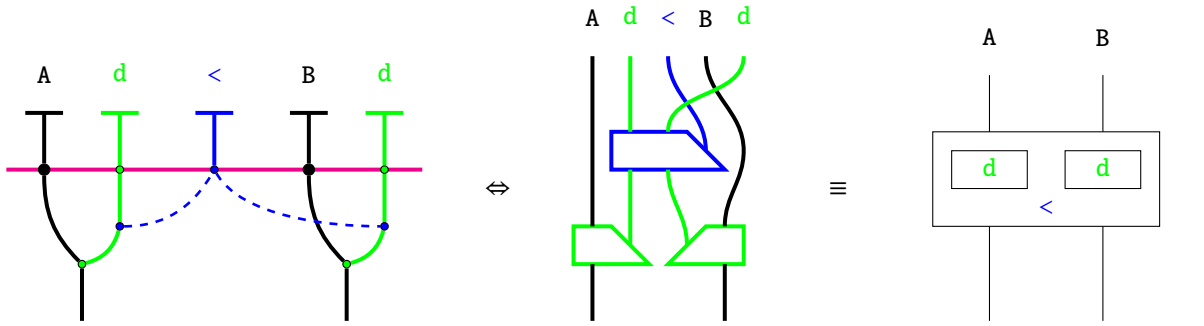


Figure 2.59:

Example 2.2.30 (Comparatives).

`Alice drinks less than Bob drinks`

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

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

theoretically, and naturally accommodate previously computed vector embeddings of words.

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

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

2.3.1 *Avenues I: syncategorematicity as distributivity*

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

Figure 2.60:

Example 2.3.1 (Syncategorematicity I).

Alice and Bob drink

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

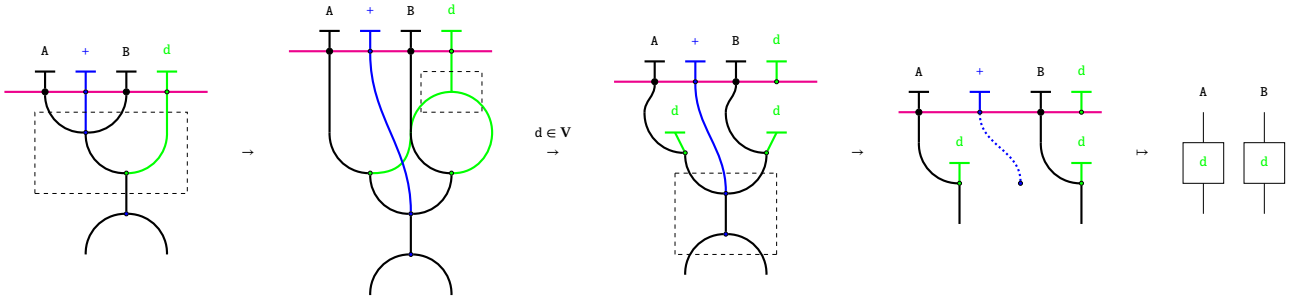
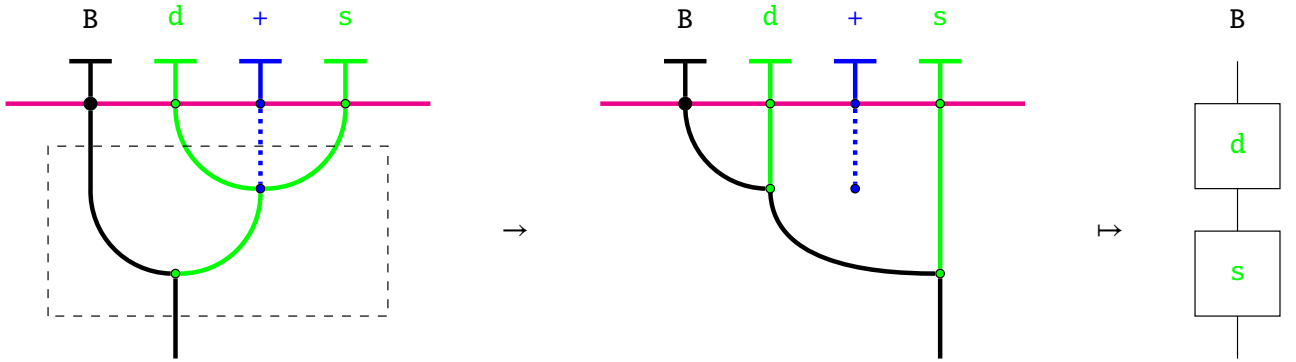


Figure 2.61:

Example 2.3.2 (Syncategorematicity II).

Bob drinks and smokes

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



2.3.2 *Avenues II: determiners and quantifiers in context*

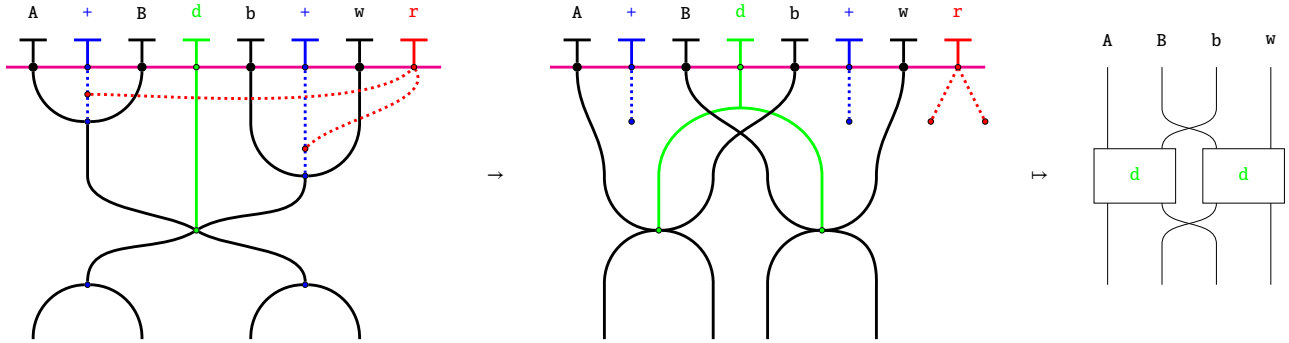
Extending the reach of text circuits to determiners, quantifiers, and conditionals appears to require a contextual diagram or process theory in which to evaluate and enforce constraints upon the purely syntactic content of text circuits. The broad strategy sketched here rests upon three tactics. First,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

Figure 2.62:

Example 2.3.3 (Coordination).

`Alice and Bob drink beer and wine respectively`

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



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

Figure 2.63:

Example 2.3.4 (Determiners I).

`Bob drinks the beer` (among drinks)

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

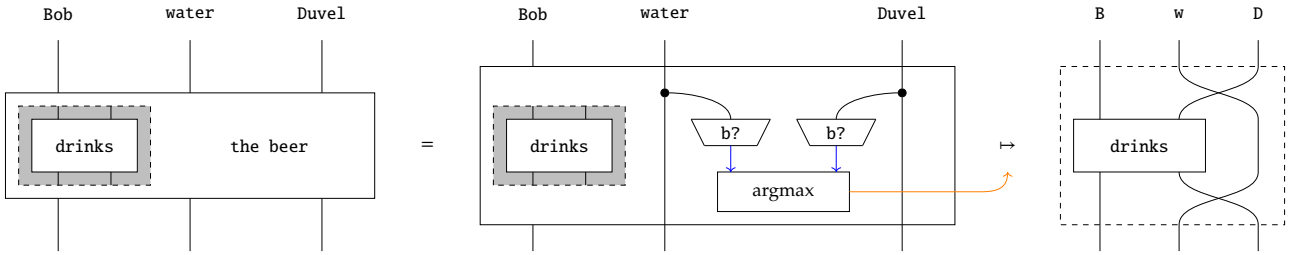


Figure 2.64:

Example 2.3.5 (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.

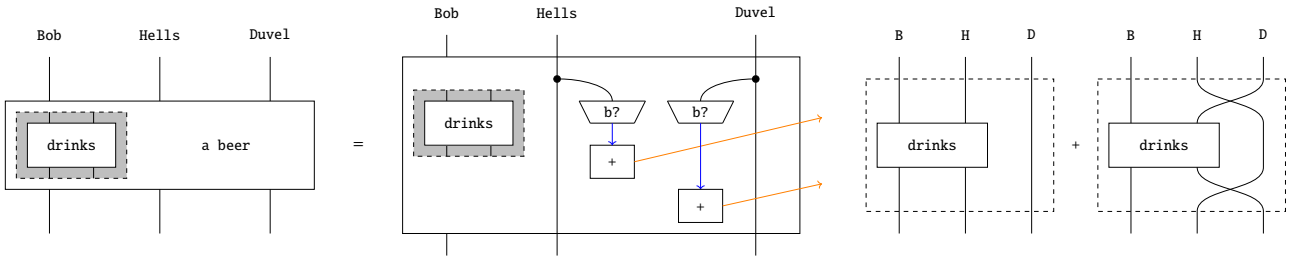


Figure 2.65:

Example 2.3.6 (Determiners III).

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

When there are no beers in context, the same statement takes on a dynamic reading: it constitutes the introduction of a beer into discourse. In terms of text circuits, this amounts to introducing a novel beer-state and beer-wire. Determining an appropriate setting to accommodate "arbitrary" vs. "concrete" beers (c.f. Fine's arbitrary objects CITE) requires further research and experimentation, but preliminarily it is known that density matrices are capable of modelling semantic entailment CITE , at the computational cost of adopting the kronecker product. This diagram doesn't typecheck, but note that `Bob drinks a beer` (in context) `because` (in context) `Bob drinks all the beers`. In a finitary context, drinking all the beers amounts to applying the distributivity of and.

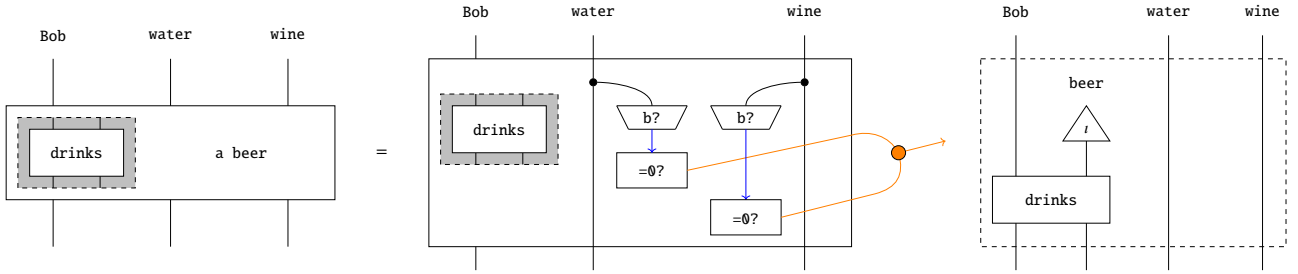


Figure 2.66:

Example 2.3.7 (Quantifiers I).

Bob drinks all the beers (in context) for evaluation of determiners treats circuits as syntactic objects to be manipulated.

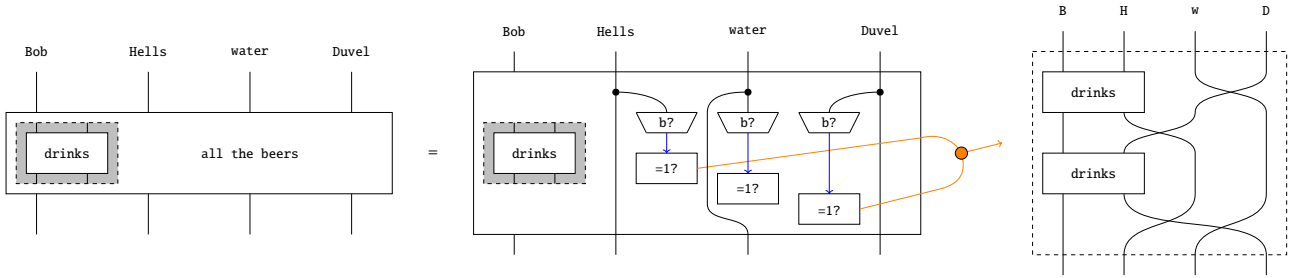
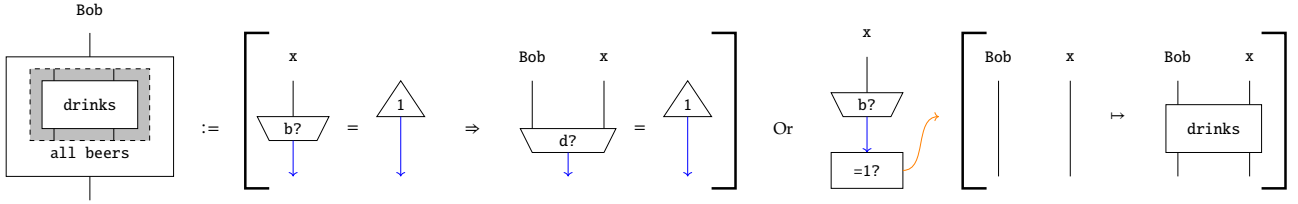


Figure 2.67:

Example 2.3.8 (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 test for the truth conditional appropriateness of measurements: whenever it is the case that something is a beer, it is the case that Bob drinks it. The second "inferential" approach 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 beer, which are in bijection with (multi-sorted) Lawvere Theories, which are equivalently coloured



2.4 A modern mathematician's companion to Montague's "Universal Grammar"

Summary: To do. "Montague semantics" means taking structure-respecting homomorphisms from grammar to meaning CITE . Montague considered grammars to be coloured operads; Montague's "algebras" are (multi-sorted) clones, which

operads, which are special cases of PROPs for symmetric monoidal categories. Hence text circuits share a mathematical lineage with many other mathematical conceptions of grammar, while also enjoying Montague semantics.

2.4.1 What did Montague consider grammar to be?

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

There is a natural division of Montague's approach into two structural components. According to Partee CITE , a formal semanticist, advocate, and torch-bearer for Montague, the chief interest of Montague's approach (as far as his contemporary linguists were concerned) lay in the following ideas:

1. Take truth conditions to be the essential data of semantics.
2. (a) Use lambdas to emulate the structure of syntax...
   (b) ...in a typed system of intensional predicate logic, such that composition is function application.

More precisely, Montague devised a higher-order intensional logic for the first point, and the notion of a structure-preserving map from syntax to semantics for the second. The truth-conditional perspective was important at the time for enabling semantic computation, but within formal semantics there arose other perspectives on the nature of formal semantics, such as inquisitive CITE and update semantics CITE . Today, the empirical evidence we have from vector-based methods in computational linguistics is that none of those conceptions of semantics are intrinsically interesting or canonical: certainly none are procedurally necessary for a broad conception of practicality. So let's nevermind points 1 and 2b.

I have split the second point to highlight the role of lambdas. This element was the crux of the Montagovian revolution: according to Janssen in a personal communication with Partee from 1994, lambdas were "...the feature that made compositionality possible at all." Using lambdas to make the semantic domain compositional then gave a target for the structure-preserving homomorphism from the syntactic domain. Today, we have more refined ways to grant structure to semantic domains using category-theoretic tools. So let's redact "lambdas" from 2a.

What remains that is of interest is the question of what Montague considered the structure of syntax to be. This is worth understanding, since we claim text circuits are a "structure of syntax", and that functorial

interpretation of text circuits in symmetric monoidal categories is Montagovian semantics in spirit if not in letter. So let's begin. In Section 1 of *Universal Grammar*, Montague's first paragraph establishes common notions of relation and function – the latter he calls *operation*, to distinguish the n-ary case from the unary case which he calls *function*. This is all done with ordinals indexing lists of elements of an arbitrary but fixed set A, which leads later on to nested indices and redundancy by repeated mention of A. We will try to avoid these issues going forward by eliding some data where there is no confusion, following common modern practice.

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

Definition 2.4.1 (Generating data of an Algebra)**.** Let A be the carrier set, and F_γ be a set of functions $A^k \to A$ for some $k \in \mathbb{N}$, indexed by $\gamma \in \Gamma$. Denoted $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$

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

Definition 2.4.2 (Identities)**.** A family of operations populated, for all $n, m \in \mathbb{N}$, $n \leq m$, by an m-ary operation $I_{n,m}$, defined on all m-tuples as

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

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

Definition 2.4.3 (Constants)**.** For all elements of the carrier $x \in A$, and all $m \in \mathbb{N}$, a constant operation $C_{x,m}$ defined on all m-tuples a as:

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

Definition 2.4.4 (Composition)**.** Given an n-ary operation G, and n instances of m-ary operations $H_{1 \leq i \leq n}$, define the composite $G(H_i)_{1 \leq i \leq n}$ to act on m-tuples a by:

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

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

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

Definition 2.4.5 (Polynomial Operations)**.** The polynomial operations over an algebra $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ are defined to be smallest class K containing all $F_{\gamma \in \Gamma}$, identities, constants, closed under composition.

Definition 2.4.6 (Homomorphism of Algebras)**.** h is a homomorphism from $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ *into* $\langle B, G_\gamma \rangle_{\gamma \in \Delta}$ iff

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

2.

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

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

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

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

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

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

However, we see aspects where Montague was born ahead of his time mathematically: it appears that his intent was to impose a system of types to constrain composition of operations, but the tools were not available for him. Montague addresses (†) obliquely, by defining $\mathtt{ME}_L$ to be the image in $\mathtt{PE}_L$ of R of just those expressions among A that are typed. Nothing appears to guard against (‡), which causes problems as Montague expresses structural constraints (in the modern view) in terms of constraints on the codomain of an interpreting functor (cf. Montague's notion of *generating* syntactic categories). One consquence, in conjunction with ($\star$), is that every multiply typed operation F induces a boolean algebra where the typings are the generators and the operations are elementwise in the inputs and output. Worse problems occur, as Montague's clone definition include all projectors, and when defined separately from the typing structure, these projectors may be typed in a way that permits operations that arbitrarily change types, which appears to defeat the purpose. We doubt these artefacts are intentional, so we will interpret Montague assuming his intent was a type-system with categorical semantics as we would use today.

2.4.2 *On the historical inevitability of text diagrams*

Definition 2.4.2 is equivalent to asking for all projections. Definitions 2.4.2 and 2.4.4 together characterise Montagovian algebras as (concrete) clones CITE , which then generalised to (abstract) clones CITE , which were then discovered to be in bijection with Lawvere theories CITE . By an evident extension of [Prop 3.51] to the typed case, a *disambiguated language* is a multi-sorted Lawvere theory without relations, where the sorts are generated from products of a pointed set $(\Delta, \delta_0 : 1 \to \Delta)$.

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

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

3
Continuous relations for semantics

We want to reason formally with and about pictorial iconic representations, of the sort one might draw to solve a problem in elementary geometry stated in words, involving topological concepts such as `touching` and `inside`. To do this in string diagrams, I introduce and investigate the category of continuous relations, **ContRel**.

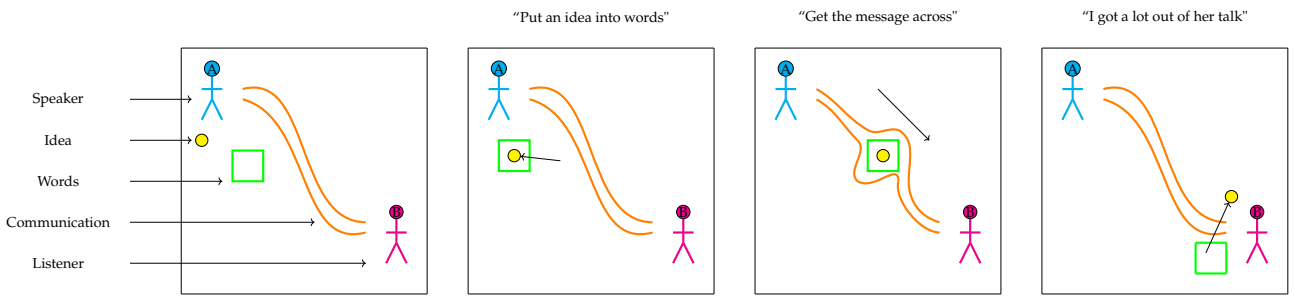


Figure 3.1: Sometimes it is very helpful to illustrate concepts using iconic representations in cartoons. For instance in the *conduit metaphor* CITE , `words` are considered *containers* for `ideas`, and `communication` is considered a *conduit* along which those containers are sent.

The aim of this chapter is to formally paint pictures with words. More verbosely, to formalise cartoon doodles like the one above in a symmetric monoidal category so that we can give semantics to text circuits in terms of graphical, iconic representations – cartoons, in short. To do so, we introduce the category **ContRel** of *continuous relations*, which are a naïve extension of the category **Top** of topological spaces and continuous functions towards continuous relations.

The main reason we prefer **ContRel** to either **Rel** or **Top** for our purposes is that we can diagrammatically characterise set-indexed collections of mutually disjoint open sets as *sticky-spiders*: a generalisation of spiders that interact with idempotents. We can then treat the indexing set as a collection of labels, and an indexed open set as a doodle. Notably, spiders don't exist in cartesian **Top** except for the one-point space, and the spatial structure of open sets doesn't exist in **Rel**.

$$placeholder : stickyspiderlaws$$

But there are all kinds of poorly behaved open sets even on the plane, so enter the next benefit: In **ContRel**, we can diagrammatically characterise the reals as a topological space up to homeomorphism, which gives us a diagrammatic handle on paths and homotopies, mathematical concepts that enable us to diagrammatically characterise when open sets are connected, how they might move and transform continuously in space, and when open sets are contained inside others.

$$realcharacterisation$$

And once we've formalised doodles we'll be able to treat ourselves to cartoons as formal semantics for language and nobody can stop us.

SIDENOTE FOR CATEGORY THEORISTS

The naïve approach I take is to observe that the preimages of functions are precisely relational converses when functions are viewed as relations, so the preimage-preserves-opens condition that defines continuous functions directly translates to the relational case. To the best of my knowledge, the study of **ContRel** is a novel contribution. I venture two potential reasons.

First, it is because and not despite of the naïvity of the construction. Usually, the relationship between **Rel** and **Set** is often understood in sophisticated general methods which are inappropriate in different ways. I have tried applying Kliesli machinery which generalises to "relationification" of arbitrary categories via appropriate analogs of the powerset monad to relate **Top** and **ContRel**, but it is not evident to me whether there is such a monad. The view of relations as spans of maps in the base category should work, since **Top** has pullbacks, but this makes calculation difficult and especially cumbersome when monoidal structure is involved. See Section ref for details.

Second, the relational nature of **ContRel** means that the category has poor exactness properties. Even if the sophisticated machinery mentioned in the first reason do manage to work, relational variants of **Top** are poor candidates for any kind of serious mathematics because they lack many limits and colimits. Since we take an entirely "monoidal" approach, we are able to find and make use of the rich structure of **ContRel** with a different toolkit.

ITINERARY OF THE CHAPTER: First we'll build some intuitions about what continuous relations are by example in Section ref . Before we can start reasoning diagrammatically, we ought to define the category **ContRel** and show it is symmetric monoidal, which will be the work in Section ref . Then we introduce sticky spiders and prove the following theorem:

Theorem 3.1.1.

$$placeholder : thmstatement$$

Finally, in Section ref , we build a vocabulary of topological concepts upon sticky spiders diagrammatically, where the point is to demonstrate sufficient expressivity to reason about whatever we want in principle. We start with the unit interval and isometries, through to rigid motions of shapes in configuration, connectedness and contractibility of shapes via homotopies, until we get to sketching some cognitively primitive relations like parthood, touching, and insideness.

3.2 *Continuous Relations by examples*

Definition 3.2.7 (Continuous Relation)**.** A continuous relation $R : (X, \tau) \to (Y, \sigma)$ is a relation $R : X \to Y$ such that

$$U \in \sigma \Rightarrow R^{\dagger}(U) \in \tau$$

where $\dagger$ denotes the relational converse.

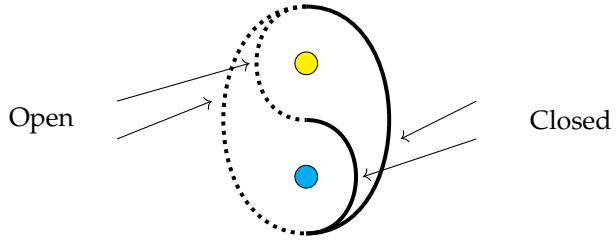
Let's consider three topological spaces and examine the continuous relations between them. This way we can build up intuitions, and prove some tool results in the process.

The **singleton space** consists of a single point which is both open and closed. We denote this space $\bullet$. Concretely, the underlying set and topology is

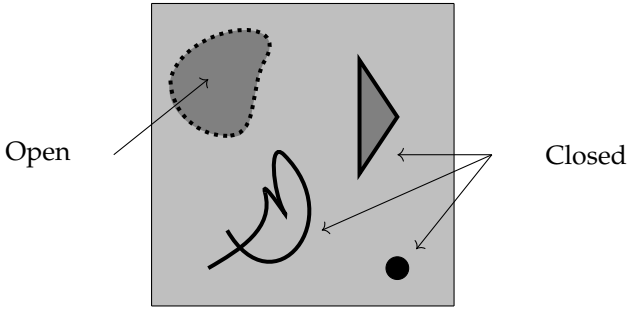
$$(\{\star\}\ ,\ \{\{\star\}, \varnothing\})$$

The **Sierpiński space** consists of two points, one of which (in yellow) is open, and the other (in cyan) is closed. We denote this space $\mathcal{S}$. Concretely, the underlying set and topology is:

$$\big(\{0,1\}\ ,\ \{\varnothing, \{1\}, \{0,1\}\}\big)$$



The **unit square** has $[0,1] \times [0,1]$ as its underlying set. Open sets are "blobs" painted with open balls. Points, lines, and bounded shapes are closed. We denote this space $\blacksquare$.



$\bullet \to \bullet$: There are two relations from the singleton to the singleton; the identity relation $\{(\bullet, \bullet)\}$, and the empty relation $\varnothing$. Both are topological.

$\bullet \to \mathcal{S}$: There are four relations from the singleton to the Sierpiński space, corresponding to the subsets of $\mathcal{S}$. All of them are topological.

$\mathcal{S} \to \bullet$: There four candidate relations from the Sierpiński space to the singleton, but as we see in Example **??**, not all of them are topological.

Now we need some abstraction. We cannot study the continuous relations between the singleton and the unit square case by case. We discover that continuous relations out of the singleton indicate arbitrary subsets, and that continuous relations into the singleton indicate arbitrary opens.

$\bullet \to \blacksquare$: Proposition **??** tells us that there are as many continuous relations from the singleton to the unit square as there are subsets of the unit square.

$\blacksquare \to \bullet$: Proposition **??** tells us that there are as many continuous relations from the unit square to the singleton as there are open sets of the unit square.

There are 16 candidate relations $\mathcal{S} \to \mathcal{S}$ to check. A case-by-case approach won't scale, so we could instead identify the building blocks of continuous relations with the same source and target space.

Given two continuous relations $R, S : X^\tau \to Y^\sigma$, how can we combine them?

Proposition 3.2.13. If $R, S : X^\tau \to Y^\sigma$ are continuous relations, so are $R \cap S$ and $R \cup S$.

Proof. Replace $\square$ with either $\cup$ or $\cap$. For any non-$\varnothing$ open $U \in \sigma$:

$$(R \square S)^\dagger(U) = R^\dagger(U) \square S^\dagger(U)$$

As R, S are continuous relations, $R^\dagger(U), S^\dagger(U) \in \tau$, so $R^\dagger(U) \square S^\dagger(U) = (R \square S)^\dagger(U) \in \tau$. Thus $R \square S$ is also a continuous relation. □

Corollary 3.2.14. Continuous relations $X^\tau \to Y^\sigma$ are closed under arbitrary union and finite intersection. Hence, continuous relations $X^\tau \to Y^\sigma$ form a topological space where each continuous relation is an open set on the base space $X \times Y$, where the full relation $X \to Y$ is "everything", and the empty relation is "nothing".

A topological basis for spaces of continuous relations

Reminder 3.2.15 (Topological Basis)**.** $\mathfrak{b} \subseteq \tau$ is a basis of the topology τ if every $U \in \tau$ is expressible as a union of elements of $\mathfrak{b}$. Every topology has a basis (itself). Minimal bases are not necessarily unique.

Having a tangible topological basis for continuous relations is good for intuition: we can think of breaking down or constructing complex relations to or from simpler parts. Luckily, there do exist nice topological bases for continuous relations!

Definition 3.2.16 (Partial Functions)**.** A **partial function** $X \to Y$ is a relation for which each $x \in X$ has at most a single element in its image. In particular, all functions are special cases of partial functions, as is the empty relation.

Lemma 3.2.17 (Partial functions are a $\cap$-ideal)**.** The intersection $f \cap R$ of a partial function $f : X \to Y$ with any other relation $R : X \to Y$ is again a partial function.

Proof. Consider an arbitrary $x \in X$. $R(x) \cap f(x) \subseteq f(x)$, so the image of x under $f \cap R$ contains at most one element, since $f(x)$ contains at most one element. □

Lemma 3.2.18 (Any single edge can be extended to a continuous partial function)**.** Given any $(x, y) \in X \times Y$, there exists a continuous partial function $X^\tau \to Y^\sigma$ that contains (x, y).

Proof. Let $\mathcal{N}(x)$ denote some open neighbourhood of x with respect to the topology τ. Then $\{(z, y) : z \in \mathcal{N}(x)\}$ is a continuous partial function that contains (x, y). □

Proposition 3.2.19. Continuous partial functions form a topological basis for the space $(X \times Y)^{(\tau \multimap \sigma)}$, where the opens are continuous relations $X^\tau \to Y^\sigma$.

Proof. We will show that every continuous relation $R : X^\tau \to Y^\sigma$ arises as a union of continuous partial functions. Denote the set of continuous partial functions $\mathfrak{f}$. We claim that:

$$R = \bigcup_{F \in \mathfrak{f}} (R \cap F)$$

The $\supseteq$ direction is evident, while the $\subseteq$ direction follows from Lemma 3.2.18. By Lemma 3.2.17, every $R \cap F$ term is a partial function, and by Corollary 3.2.14, continuous. □

$\mathcal{S} \to \mathcal{S}$: We can use Proposition 3.2.19 to write out the topological basis of continuous partial functions, from which we can take unions to find all the continuous relations, which we depict in Figure **??**.

$\mathcal{S} \to \blacksquare$: Now we use the colour convention of the points in $\mathcal{S}$ to "paint" continuous relations on the unit square "canvas", as in Figures **??** and **??**. So each continuous relation is a painting, and we can characterise the

paintings that correspond to continuous relations $\mathcal{S} \to \blacksquare$ in words as follows: Cyan only in points and lines, and either contained in or at the boundary of yellow or green. Have as much yellow and green as you like.

$\blacksquare \to \mathcal{S}$: The preimage of all of $\mathcal{S}$ must be an open set. So the painting cannot have stray lines or points outside of blobs. The preimage of yellow must be open, so the union of yellow and green in the painting cannot have stray lines or points outside of blobs. Point or line gaps within blobs are ok. Each connected blob can contain any colours in any shapes, subject to the constraint that if cyan appears anywhere, then either yellow or green must occur somewhere. Open blobs with no lines or points outside. Yellow and green considered alone is a painting made of blobs with no stray lines or points. If cyan appears anywhere, then either yellow or green have to appear somewhere.

One more example for fun: $[0,1] \to \blacksquare$: We know how continuous functions from the unit line into the unit square look.

Then what are the partial continuous functions? If we understand these, we can obtain all continuous relations by arbitrary unions of the basis. Observe that the restriction of any continuous function to an open set in the source is a continuous partial function. The open sets of $[0,1]$ are collections of open intervals, each of which is homeomorphic to $(0,1)$, which is close enough to $[0,1]$.

3.3 *The category* ***ContRel***

Definition 3.3.9 (ContRel). The (purported) category **ContRel** has topological spaces for objects and continuous relations for morphisms.

Proposition 3.3.10 (ContRel is a category). continuous relations form a category **ContRel**.

Proof. Identities: Identity relations, which are always continuous since the preimage of an open U is itself.

Composition: The normal composition of relations. We verify that the composite $X^\tau \xrightarrow{R} Y^\sigma \xrightarrow{S} Z^\theta$ of continuous relations is again continuous as follows:

$$U \in \theta \implies S^\dagger(U) \in \sigma \implies R^\dagger \circ S^\dagger(U) = (S \circ R)^\dagger \in \tau$$

Associativity of composition: Inherited from **Rel**. □

3.3.1 *Symmetric Monoidal structure*

Proposition 3.3.11. $(\mathbf{ContRel}, \bullet, X^\tau \otimes Y^\sigma := (X \times Y)^{(\tau \times \sigma)})$ is a symmetric monoidal category.

Tensor Unit: The one-point space $\bullet$. Explicitly, $\{\star\}$ with topology $\{\emptyset, \{\star\}\}$.

Tensor Product: For objects, $X^\tau \otimes Y^\sigma$ has base set $X \times Y$ equipped with the product topology $\tau \times \sigma$. For morphisms, $R \otimes S$ the product of relations. We show that the tensor of continuous relations is again a continuous relation. Take continuous relations $R : X^\tau \to Y^\sigma$, $S : A^\alpha \to B^\beta$, and let U be open in the product topology $(\sigma \times \beta)$. We need to prove that $(R \times S)^\dagger(U) \in (\tau \times \alpha)$. We may express U as $\bigcup_{i \in I} y_i \times b_i$, where the y_i and b_i are in the bases $\mathfrak{b}_\sigma$ and $\mathfrak{b}_\beta$ respectively. Since for any relations we have that $R(A \cup B) = R(A) \cup R(B)$ and $(R \times S)^\dagger = R^\dagger \times S^\dagger$:

$$\begin{aligned}
&(R \times S)^\dagger\Big(\bigcup_{i \in I} y_i \times b_i\Big) \\
&= \bigcup_{i \in I} (R \times S)^\dagger (y_i \times b_i) \\
&= \bigcup_{i \in I} (R^\dagger \times S^\dagger)(y_i \times b_i)
\end{aligned}$$

Since each y_i is open and R is continuous, $R^\dagger(y_i) \in \tau$. Symmetrically, $S^\dagger(b_i) \in \alpha$. So each $(R^\dagger \times S^\dagger)(y_i \times b_i) \in (\tau \times \alpha)$. Topologies are closed under arbitrary union, so we are done.

The natural isomorphisms are inherited from **Rel**. We will be explicit with the unitor, but for the rest, we will only check that the usual isomorphisms from **Rel** are continuous in **ContRel**. To avoid bracket-glut, we will vertically stack some tensored expressions.

Unitors: The left unitors are defined as the relations $\lambda_{X^\tau} : \bullet \times X^\tau \to X^\tau := \{(\begin{pmatrix}\star\\ x\end{pmatrix}, x) \mid x \in X\}$, and we reverse the pairs to obtain the inverse $\lambda^{-1}_{X^\tau}$. These relations are continuous since the product topology of τ with the singleton is homeomorphic to τ: $U \in \tau \iff (\bullet, U) \in (\bullet \times \tau)$. These relations are evidently inverses that compose to the identity. The construction is symmetric for the right unitors ρ_{X^τ}.

Associators: The associators $\alpha_{X^\tau Y^\sigma Z^\rho} : ((X \times Y) \times Z)^{((\tau\times\sigma)\times\rho)} \to (X \times (Y \times Z))^{(\tau\times(\sigma\times\rho))}$ are inherited from **Rel**. They are:

$$\alpha_{X^\tau Y^\sigma Z^\rho} := \{\big((\begin{pmatrix}x\\ y\end{pmatrix}, z)\,,\, (x, \begin{pmatrix}y\\ z\end{pmatrix})\big) \quad | \quad x \in X\,,\, y \in Y\,,\, z \in Z\}$$

To check the continuity of the associator, observe that product topologies are isomorphic in **Top** up to bracketing, and these isomorphisms are inherited by **ContRel**. The inverse of the associator has the pairs of the relation reversed and is evidently an inverse that composes to the identity.

Braids: The braidings $\theta_{X^\tau Y^\sigma} : (X \times Y)^{\tau\times\sigma} \to (Y \times X)^{\sigma\times\tau}$ are defined:

$$\{(\begin{pmatrix}x\\ y\end{pmatrix}\,,\, \begin{pmatrix}y\\ x\end{pmatrix}) \quad | \quad x \in X\,,\, y \in Y\}$$

The braidings inherit continuity from the isomorphisms between $X^\tau \times Y^\sigma$ and $Y^\sigma \times X^\tau$ in **Top**. They inherit everything else from **Rel**

Coherences: Since we have verified all of the natural isomorphisms are continuous, it suffices to say that the coherences [] are inherited from the symmetric monoidal structure of **Rel** up to marking objects with topologies.

3.3.2 Rig category structure

Definition 3.3.12 (Biproducts and zero objects)**.** A *biproduct* is simultaneously a categorical product and coproduct. A *zero object* is both an initial and a terminal object. **Rel** has biproducts (the coproduct of sets equipped with reversible injections) and a zero object (the empty set).

Proposition 3.3.13. ContRel has a zero object.

Proof. As in **Rel**, there is a unique relation from every object to and from the empty set with the empty topology. □

Proposition 3.3.14. **ContRel** has biproducts.

Proof. The biproduct of topologies X^τ and Y^σ is their direct sum topology $(X \sqcup Y)^{(\tau+\sigma)}$ – the coarsest topology that contains the disjoint union $\tau \sqcup \sigma$. As in **Rel**, the (in/pro)jections are partial identities, which are continuous by construction. To verify that it is a coproduct, given continuous relations $R : X^\tau \to Z^\rho$ and $S : Y^\sigma \to Z^\rho$, where the disjoint union $X \sqcup Y$ of sets is $\{x_1 \mid x \in X\} \cup \{y_2 \mid y \in Y\}$, we observe that $R + S := \{(x_1, z) \mid (x, z) \in R\} \cup \{(y_2, z) \mid y \in S\}$ is continuous and commutes with the injections as required. The argument that it is a product is symmetric. □

Remark 3.3.15. Biproducts yield another symmetric monoidal structure which the $\times$ monoidal product distributes over appropriately to yield a rig category. Throughout the chapter we will use $\cup$, but we could have also "diagrammatised" $\cup$ by treating it as a monoid internal to **ContRel** viewed as a symmetric monoidal category with respect to the biproduct. There are at least two diagrammatic formalisms for rig categories that we could have used, CITE and CITE , but it gets too visually complicated. especially when we sometimes take unions over arbitrary indexing sets, which is alright in topology but not depictable as a finite diagram in the $\oplus$-structure. A neat fact that follows is that a topological space is compact precisely when any arbitrarily indexed $\cup$ of tests in the $\times$-structure is *depictable* in the $\oplus$-structure of either diagrammatic calculus for rig categories. **FdHilb** also has a monoidal product notated $\otimes$ that distributes over the monoidal structure given by biproducts $\oplus$. In contrast, we have used $\times$ – the cartesian product notation – for the monoidal product of **ContRel** since that is closer to what is familiar for sets.

3.3.3 *Monoidal (co!)closure*

Definition 3.3.16 (Closure type)**.** Recall by Proposition 3.2.13 that continuous relations $X^\tau \to Y^\sigma$ form a topological space. Denote this space $(X \times Y)^{(\tau \multimap \sigma)}$

The permissible continuous relations $X^\tau \to Y^\sigma$ are tests on $(X \times Y)^{(\tau \multimap \sigma)}$. **ContRel** is not monoidal closed, because taking a closure type as an input to the evaluator would permit arbitrary subsets of $X \times Y$ as arguments. So what we seek instead is a coevaluation, where the closure type is an output. This is not as straightforward as it is in strongly compact closed **Rel**, where we may use cups and caps for process-state duality (CJ-isomorphism), because in **ContRel** we have cups *but no caps*. However, we may exploit the fact that discrete topologies behave like plain sets See Lemma 3.3.23) and the observation that we may coarsen discrete topologies into target topologies, which is essentially enough to recover monoidal coclosed structure.

Proposition 3.3.17. For any X^τ and Y^σ, $\tau \times \sigma \subseteq \tau \multimap \sigma$; the product topology is coarser than the corresponding closure topology.

Proof. Let $\mathfrak{b}_\tau, \mathfrak{b}_\sigma$ be bases for τ and σ respectively, then $\tau \times \sigma$ has basis $\mathfrak{b}_\tau \times \mathfrak{b}_\sigma$. An arbitrarily element ($t \in \tau, s \in \sigma$) of this product basis can be viewed as a topological relation $t \times s \subseteq X \times Y$. Every open of $\tau \times \sigma$ is a union of such basis elements, and topological relations are closed under arbitrary union, so we have the (evidently injective) correspondence:

$$\tau \times \sigma \ni \bigcup_{i \in I} (t_i \times s_i) \mapsto \bigcup_{i \in I} (t_i \times s_i) \in \tau \multimap \sigma$$

□

Example 3.3.18 ($\tau \multimap \sigma \not\subseteq \tau \times \sigma$)**.** Recalling Proposition **??**, let $\tau = \{\emptyset, \{\star\}\}$ on the singleton, and σ be an arbitrary nondiscrete topology on base space Y. $(\{\star\} \times Y)^{(\tau \times \sigma)}$ is isomorphic to Y^σ, but $(\{\star\} \times Y)^{(\tau \multimap \sigma)}$ is the isomorphic to the discrete topology $Y^\bullet$. For a more concrete example, consider the Sierpiński space $\mathcal{S}$ again, along with the topological relation $\{(0,0),(1,0),(1,1)\} \subset \mathcal{S} \times \mathcal{S}$; due to the presence of $(0,0)$, this topological relation cannot be formed by a union of basis elements of the product topology, which are:

$$\{1\} \times \{1\} = \{(1,1)\}$$

$$\{1\} \times \{0,1\} = \{(1,0),(1,1)\}$$

$$\{0,1\} \times \{1\} = \{(1,1),(0,1)\}$$

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(0,1)\}$$

Definition 3.3.19 (Coarsening)**.** Where $\tau \supseteq \rho$ are topologies on X, the identity-on-elements relation $X^\tau \to X^\rho$ is continuous; the relational converse of the identity is the identity, which witnesses opens of ρ as opens of τ. but the converse is not unless $\tau = \rho$. We denote these *coarsening* relations as:

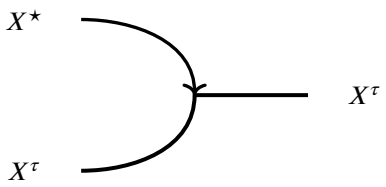
$$X^\tau \quad \longrightarrow\!\!\!\!\!\!\circ \quad X^\rho$$

Definition 3.3.20 (Pseudo-compare)**.** For any X^τ, the relation $X^\star \times X^\tau \to X^\tau$ defined on objects as

$$\{\begin{pmatrix} x \\ x \end{pmatrix}, x \mid x \in X\}$$

is continuous; the relational converse is the copy map, which sends opens U in τ to $U \times U \in \tau \times \tau \subseteq X \times X$,

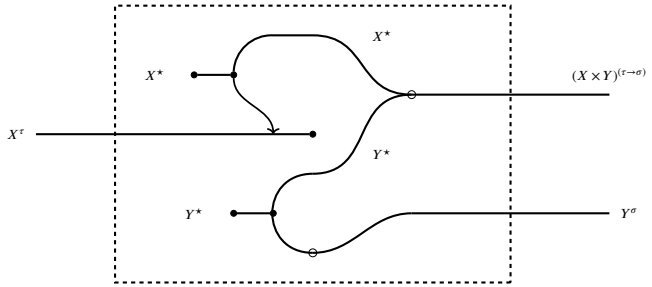
which are opens in $X^\star \times X^\tau$. We denote these *pseudo-compare* maps:



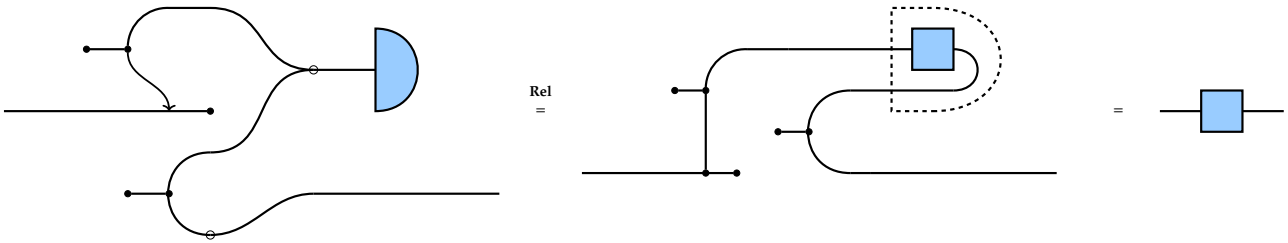
Proposition 3.3.21. Where $\star_X$ and $\star_Y$ are discrete topologies on X and Y, $\star_X \times \star_Y = \star_X \multimap \star_Y$.

Proof. Relations between discrete topologies are just arbitrary relations, and relations are monoidal closed with $X \multimap Y \simeq X \times Y$. □

Proposition 3.3.22 (**ContRel** is monoidal coclosed)**.** The coevaluation map is:



Proof. The string diagram itself demonstrates that the coevaluation is continuous, since we have demonstrated that **ContRel** is symmetric monoidal, and we know that copy (Prop. **??**), everything (Prop. **??**), coarsenings (Defn. 3.3.19) and pseudo-compares (Defn. 3.3.20) are continuous. What remains to be demonstrated is that the coevaluation behaves like one; i.e. that plugging in a continuous relation expressed as a test into the closure type of the coevaluator recovers that continuous relation. This can be shown diagrammatically by equating the underlying relations on sets in **Rel**.



The first equation is obtained by a few steps. Forgetting topology, we turn all coarsenings into identities in

Rel, and in particular, proposition 3.3.21 deals with the 2-1 coarsening. The expression inside the closure test is obtained by CJ-isomorphism using strong compact closure in **Rel**. The pseudo-compares in **ContRel** becomes an honest compare in **Rel**. The second equation then follows by frobenius. □

That's all we need for the diagrams. The remainder of this section are endnotes for category theorists addressing the question of how **ContRel** relates to **Rel** and **Top**, and some conceptual motivations for topological relations. If none of that interests you, ignore the main body: the margins carry on with diagrammatic facts about **ContRel**.

3.3.4 *Category-theoretic endnotes*

ContRel and Rel are related by a free-forgetful adjunction

We provide free-forgetful adjunctions relating **ContRel** to **Rel** by "forgetting topology" and sending sets to "free" discrete topologies. We exhibit a free-forgetful adjunction between **Rel** and **ContRel**.

Lemma 3.3.23 (Any relation R between discrete topologies is continuous)**.** *Proof.* All subsets in a discrete topologies are open. □

Definition 3.3.24 (L: **Rel** → **ContRel**)**.** We define the action of the functor L:

On objects $L(X) := X^\star$, (X with the discrete topology)

On morphisms $L(X \overset{R}{\to} Y) := X^\star \overset{R}{\to} Y^\star$, the existence of which in **ContRel** is provided by Lemma 3.3.23.

Evidently identities and associativity of composition are preserved.

Definition 3.3.25 (R: **ContRel** → **Rel**)**.** We define the action of the functor R as forgetting the topological structure.

On objects $R(X^\tau) := X$

On morphisms $R(X^\tau \overset{S}{\to} Y^\sigma) := X \overset{S}{\to} Y$

Evidently identities and associativity of composition are preserved.

Lemma 3.3.26 ($RL = 1_{\mathbf{Rel}}$)**.** The composite RL (first L, then R) is precisely equal to the identity functor on **Rel**.

Proof. On objects, $FU(X) = F(X^\star) = X$. On morphisms, $FU(X \overset{R}{\to} Y) = F(X^\star \overset{R}{\to} Y^\star) = X \overset{R}{\to} Y$ □

Reminder 3.3.27 (Coarser and finer)**.** Given a set of points X with two topologies X^τ and X^σ, if $\tau \subset \sigma$, we say that τ *is coarser than* σ, or σ *is finer than* τ.

Lemma 3.3.28 (Coarsening is a continuous relation)**.** Let X^σ be coarser than X^τ. The identity relation on underlying points $X^\tau \overset{1_X}{\to} X^\sigma$ is then a continuous relation.

Proof. The preimage of the identity of any open set $U \in \sigma, U \subseteq X$ is again U. By definition of coarseness, $U \in \tau$. □

Proposition 3.3.29 ($L \dashv R$)**.** *Proof.* We verify the triangular identities governing the unit and counit of the adjunction, which we first provide. By Lemma 3.3.26, we take the natural transformation $1_{\mathbf{Rel}} \Rightarrow RL$ we take to be the identity morphism:

$$\eta_X := 1_X$$

The counit natural transformation $LR \Rightarrow 1_{\mathbf{ContRel}}$ we define to be a coarsening, the existence of which in **ContRel** is granted by Lemma 3.3.28.

$$\epsilon_{X^\tau} : X^\star \to X^\tau := \{(x, x) : x \in X\}$$

First we evaluate $L \overset{L\eta}{\to} LRL \overset{\epsilon L}{\to} L$ at an arbitrary object (set) $X \in \mathbf{Rel}$. $L(X) = X^\star = LRL(X)$, where the latter equality holds because LR is precisely the identity functor on **Rel**. For the first leg from the left, $L(\eta_X) = L(1_X) = X^\star \overset{1_X}{\to} X^\star = 1_{X^\star}$. For the second, $\epsilon_{L(X)} = \epsilon_{X^\star} = X^\star \overset{1_X}{\to} X^\star = 1_{X^\star}$. So we have that $L\eta; \epsilon L = L$ as required.

Now we evaluate $R \overset{\eta R}{\to} RLR \overset{R\epsilon}{\to} R$ at an arbitrary object (topological space) $X^\tau \in \mathbf{ContRel}$. $R(X^\tau) = X = RLR(X^\tau)$, where the latter equality again holds because $LR = 1_{\mathbf{Rel}}$. For the first leg from the left, $\eta_{R(X^\tau)} = \eta_X = 1_X$. For the second, $R(\epsilon_{X^\tau}) = R(X^\star \overset{1_X}{\to} X^\tau) = X \overset{1_X}{\to} X = 1_X$. So $\eta R; R\epsilon = R$, as required. □

The usual forgetful functor from **ContRel** to **Loc** has no left adjoint. Just as the forgetful functor from **ContRel** to **Rel** "forgets topology while keeping the points", we might consider a forgetful functor to **Loc** that "forgets points while remembering topology". But we show that there is no such functor that forms a free-forgetful adjunction.

Reminder 3.3.30 (The category **Loc**)**.** CITE A *frame* is a poset with all joins and finite meets satisfying the infinite distributive law:

$$x \wedge (\bigvee_i y_i) = \bigvee_i (x \wedge y_i)$$

A *frame homomorphism* $\phi : A \to B$ is a function between frames that preserves finite meets and arbitrary joins, i.e.:

$$\phi(x \wedge_A y) = \phi(x) \wedge_B \phi(y) \qquad \phi(x \vee_A y) = \phi(x) \vee_B \phi(y)$$

The category **Frm** has frames as objects and frame homomorphisms as morphisms. The category **Loc** is defined to be $\mathbf{Frm}^{\mathrm{op}}$.

Remark 3.3.31. Here are informal intuitions to ease the definition. The lattice of open sets of a given topology ordered by inclusion forms a frame – observe the analogy "arbitrary unions" : "all joins" :: "finite intersections" : "finite meets". Closure under arbitrary joins guarantees a maximal element corresponding to the open set that is the whole space. So frames are a setting to speak of topological structure alone, without referring

to a set of underlying points, hence, pointless topology. Observe that in the definition of continuous functions, open sets in the *codomain* must correspond (uniquely) to open sets in the *domain* – so every continuous function induces a frame homomorphism going in the opposite direction that the function does between spaces, hence, to obtain the category **Loc** such that directions align, we reverse the arrows of **Frm**. Observe that continuous relations induce frame homomorphisms in the same way. These observations give us insight into how to construct the free and forgetful functors.

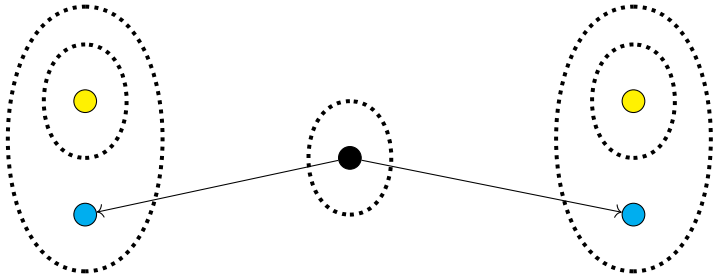
Definition 3.3.32 (U : **ContRel** → **Loc**)**.** On objects, U sends a topology X^τ to the frame of opens in τ, which we denote $\hat{\tau}$.

On morphisms $R : X^\tau \to Y^\sigma$, the corresponding partial frame morphism $\hat{\tau} \leftarrow \hat{\sigma}$ (notice the direction reversal for **Loc**), we define to be $\{(U_{\in\sigma}, R^\dagger(U)_{\in\tau}) \mid U \in \sigma\}$. We ascertain that this is (1) a function that is (2) a frame homomorphism. For (1), since the relational converse picks out precisely one subset given any subset as input, these pairs do define a function. For (2), we observe that the relational converse (as all relations) preserve arbitrary unions and intersections, i.e. $R^\dagger(\bigcap_i U_i) = \bigcap_i R^\dagger(U_i)$ and $R^\dagger(\bigcup_i U_i) = \bigcup_i R^\dagger(U_i)$, so we do have a frame homomorphism. Associativity follows easily.

Proposition 3.3.33 (U has no left adjoint)**.** *Proof.* Seeking contradiction, if U were a right adjoint, it would preserve limits. The terminal object in **Loc** is the two-element lattice $\bot < \top$, where the unique frame homomorphism to any $\mathcal{L}$ sends $\top$ to the top element of $\mathcal{L}$ and $\bot$ to the bottom element. In **ContRel**, the empty topology $\mathbf{0} = (\varnothing, \{\varnothing\})$ is terminal (and initial). However, $U\mathbf{0}$ is the singleton lattice, not $\bot < \top$ (which is the image under U of the singleton topology). □

This is a rather frustrating result, because U does turn continuous relations into backwards frame homomorphisms on lattices of opens; see Proposition 3.2.13, and note that in the frame of opens associated with a topology, the empty set becomes the bottom element. The obstacle is the fact that the empty topology is both initial and terminal in **ContRel**. We may be tempted to try treating U as a right adjoint going to **Frm** instead, but then the monad induced by the injunction on **Loc** would trivialise: left adjoints preserve colimits, so any putative left adjoint F must send $\bot < \top$ (initial in **Frm** by duality) to the empty toplogy, and the empty topology as terminal object must be sent to the terminal singleton frame, which implies that the monad UF on **Frm** sends everything to the singleton lattice.

Why not Span(**Top**)? One common generalisation of relations is to take spans of monics in the base category []. This actually produces a different category than the one we have defined. Below is an example of a span of monic continuous functions from **Top** that corresponds to a relation that doesn't live in **ContRel**. It is the span with the singleton as apex, with maps from the singleton to the closed points of a two Sierpiński spaces.



Why not a Kleisli construction on **Top**? Another way to view the category **Rel** is as the Kleisli category $K_{\mathcal{P}}$ of the powerset monad on **Set**; that is, every relation $A \to B$ can be viewed as a function $A \to \mathcal{P}B$, and composition works by exploiting the monad multiplication: $A \overset{f}{\to} \mathcal{P}B \overset{\mathcal{P}g}{\to} \mathcal{P}\mathcal{P}C \overset{\mu_{\mathcal{P}C}}{\to} \mathcal{P}C$. So it is reasonable to investigate whether there is a monad T on **Top** such that K_T is equivalent to **ContRel**. We observe that the usual free-forgetful adjunction between **Set** and **Top** sends the former to a full subcategory (of continuous functions between discrete topologies) of the latter, so a reasonable coherence condition we might ask for the putative monad T to satisfy is that it is related to $\mathcal{P}$ via the free-forgetful adjunction. This amounts to asking for the following commutative diagram (in addition to the usual ones stipulating that T and $\mathcal{P}$ are monadic):

$$\begin{array}{ccc} \mathbf{Top} & \xrightarrow{\ T\ } & \mathbf{Top} \\ \dashv\downarrow\uparrow & & \dashv\downarrow\uparrow \\ \mathbf{Set} & \xrightarrow{\ \mathcal{P}\ } & \mathbf{Set} \end{array}$$

This condition would be nice to have because it witnesses $K_{\mathcal{P}}$ as precisely K_T restricted to the discrete topologies, so that T really behaves as a conservative generalisation of the notion of relations to accommodate topologies. As a consequence of this condition, we may observe that discrete topologies $X^\star$ must be sent to discrete topologies on their powerset $\mathcal{P}X^\star$. In particular, this means the singleton topology is sent to the the discrete topology on a two-element set; $T\bullet = \mathbf{2}$. This sinks us. We know from Proposition **??** that the continuous relations $X^\tau \to \bullet$ are precisely the open sets of τ, which correspond to continuous functions into Sierpiński space $X^\tau \to \mathbb{S}$, and $\mathbb{S} \neq \mathbf{2}$.

Where is the topology coming from?

It is category-theoretically natural to ask whether **ContRel** is "giving topology to relations" or "powering up topologies with relations", but we have explored those techniques and it doesn't seem to be that. It is possible that the failure of these regular avenues may explain why I had such difficulty finding **ContRel** in the literature. However, we do have a free-forgetful adjunction between **ContRel** and **Rel**, and if we focus

on this, it *is* possible to crack the nut of where topology is coming from with enough machinery; here is one such sketch. Observe that the forgetful functor looks like it could be a kind of fibration, where the elements of the fibre over any set A in **Rel** correspond to all possible topologies on A. Moreover, these topologies may be partially ordered by coarseness-fineness to form a frame (though a considering it a preorder will suffice.) The fibre over a relation $R : A \to B$ is all pairs of topologies τ, σ such that R is continuous between A^τ and B^σ. The crucial observation is that if R is continuous between τ and σ, then R will be continuous for any finer topology in the domain, $\tau \leq \tau'$, and any coarser topology in the codomain $\sigma' \leq \sigma$; that is, the fibre over R displays a boolean-valued profunctor between preorders. So **ContRel** can be viewed as the display category induced by a functor **Rel** $\to \mathcal{C}$, where $\mathcal{C}$ is a category with preorders for objects and boolean-enriched profunctors as morphisms, and the functor encodes topological data by sending sets in **Rel** to preorders of all possible topologies, and relations to profunctors. I have deliberately left this as a sketch because it doesn't seem worth it to view something so simple in such a complex way (I accept the charges of hypocrisy having just used weak n-categories to present TAGs.)

3.4 Populating space with shapes using sticky spiders

3.4.1 When does an object have a spider (or something close to one)?

Example 3.4.1 (The copy-compare spiders of **Rel** are not always continuous)**.** The compare map for the Sierpiński space is not continuous, because the preimage of $\{0,1\}$ is $\{(0,0),(1,1)\}$, which is not open in the product space of S with itself.

Reminder 3.4.2 (copy-compare spiders of **Rel**)**.** For a set X, the *copy* map $X \to X \times X$ is defined:

$$\{(x,(x,x)) : x \in X\}$$

the *compare* map $X \times X \to X$ is defined:

$$\{((x,x),x) : x \in X\}$$

These two maps are the (co)multiplications of special frobenius algebras. The (co)units are *delete*:

$$\{(x,\star) : x \in X\}$$

and *everything*:

$$\{(\star,x) : x \in X\}$$

We can use split idempotents to transform copy-spiders from discrete topologies to sticky-spiders on other spaces.

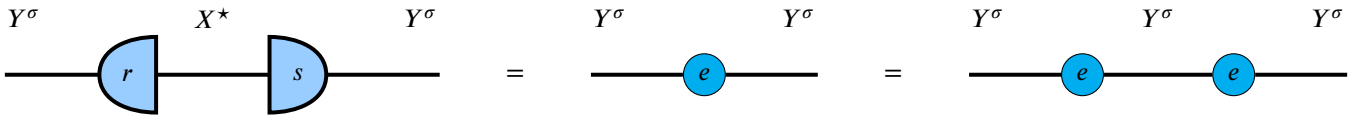
Reminder 3.4.5 (Split idempotents)**.** An **idempotent** in a category is a map $e : A \to A$ such that

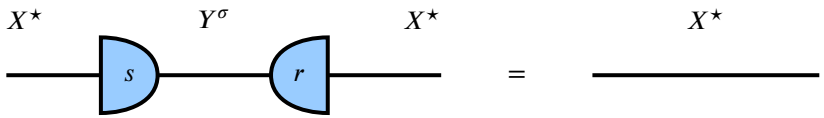
$$A \overset{e}{\to} A \overset{e}{\to} A = A \overset{e}{\to} A$$

A **split idempotent** is an idempotent $e : A \to A$ along with a **retract** $r : A \to B$ and a **section** $s : B \to A$ such that:

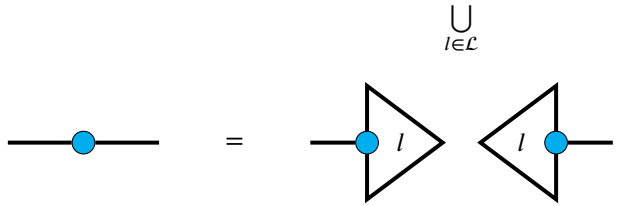
$$A \overset{e}{\to} A = A \overset{r}{\to} B \overset{s}{\to} A$$
$$B \overset{s}{\to} A \overset{r}{\to} B = B \overset{id}{\to} B$$

We can graphically express the behaviour of a split idempotent e as follows, where the semicircles for the section and retract r, s form a visual pun.

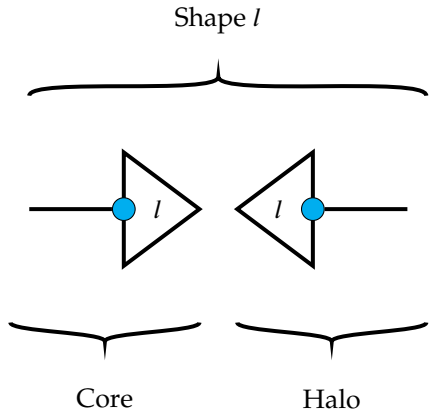

Y^σ
$X^\star$
Y^σ
r
s
=
Y^σ
Y^σ
e
=
Y^σ
Y^σ
Y^σ
e
e


$X^\star$
Y^σ
$X^\star$
s
r
=
$X^\star$

Definition 3.4.22 (Labels, shapes, cores, halos)**.** Recall by Proposition **??** that we can express the idempotent as a union of continuous relations formed of a state and test, for some indexing set of *labels* $\mathcal{L}$.



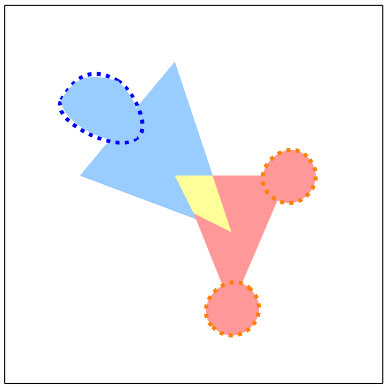
A *shape* is a component of this union corresponding to some arbitary $l \in L$. So we refer to a sticky spider as a labelled collection of shapes. The state of a shape is the *halo* of the shape. The halos are precisely the copiables of the sticky spider. The test of a shape is the *core*. The cores are precisely the cocopiables of the sticky spider.



Proposition 3.4.23 (Core exclusion: Distinct cores cannot overlap)**.** *Proof.* A direct consequence of Lemma **??**. □

Proposition 3.4.24 (Core-halo exclusion: Each core only overlaps with its corresponding halo)**.** *Proof.* Seeking contradiction, if a core overlapped with multiple halos, Lemma **??** would be violated. □

Proposition 3.4.25 (Halo non-exclusion: halos may overlap)**.** *Proof.* By example:



The two shapes are colour coded cyan and magenta. The halos are two triangles which overlap at a yellow region, and partially overlap with their blobby cores. The cores are outlined in dotted blue and orange respectively. Observe that cores and halos do not have to be simply connected; in this example the core of the magenta shape has two connected components. Viewing these sticky spiders as a process, any shape that overlaps with the magenta core will be deleted and replaced by the magenta triangle, and similarly with the cyan cores and triangle. Any shape that overlaps with both the magenta and cyan cores will be deleted and replaced by the union of the triangles. Any shape that overlaps with neither core will be deleted and not replaced. □

Example 3.4.26 (Analog of quantum venn diagram paradox)**.**

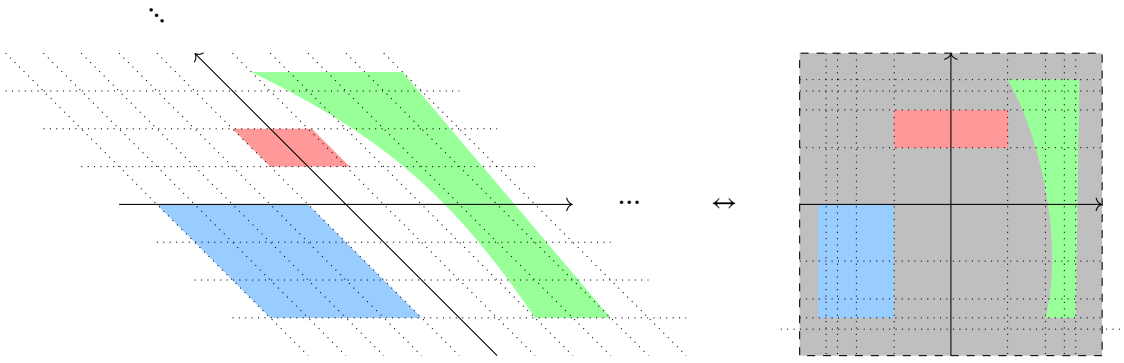
Proposition 3.4.27 (Set-indexed collection of open sets)**.**

3.5 *Topological concepts in flatland via* ***ContRel***

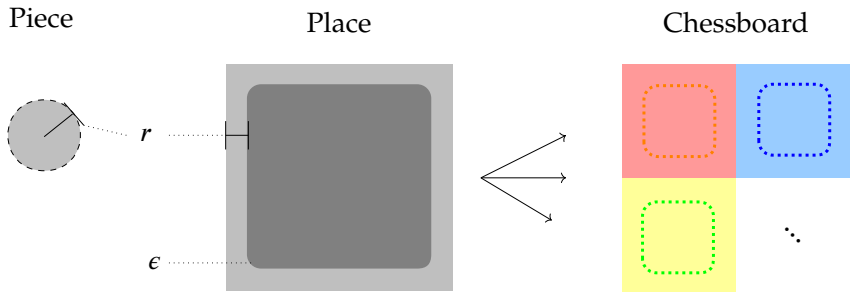
The goal of this section is to demonstrate the use of sticky spiders as formal semantics for the kinds of schematic doodles or cartoons we would like to draw. Throughout we consider sticky spiders on $\mathbb{R}^2$. In Section 3.5.1, we introduce how sticky spiders may be viewed as labelled collections of shapes. In service of defining *configuration spaces* of shapes up to rigid displacement, we diagrammatically characterise the topological subgroup of isometries of $\mathbb{R}^2$ by building up in Sections 3.5.2 and 3.5.3 the diagrammatic presentations of the unit interval, metrics, and topological groups. To further isolate rigid displacements that arise from continuous sliding motion of shapes in the plane (thus excluding displacements that result in mirror-images), in Sections 3.5.4 and 3.5.5 we diagrammatically characterise an analogue of homotopy in the relational setting. Finally, in Sections 3.5.6 and 3.5.7 we build up a stock of topological concepts and study by examples how implementing these concepts within text circuits explains some idiosyncrasies of the theory: namely why noun wires are labelled by their noun, why adjective gates ought to commute, and why verb gates do not.

3.5.1 *Shapes and places*

Remark 3.5.1. When we draw on a finite canvas representing all of euclidean space, properly there should be a fishbowl effect that relatively magnifies shapes close to the origin and shrinks those at the periphery, but that is only an artefact of representing all of euclidean space on a finite canvas. Since all the usual metrics are still really there, going forward we will ignore this fishbowl effect and just doodle shapes as we see fit.

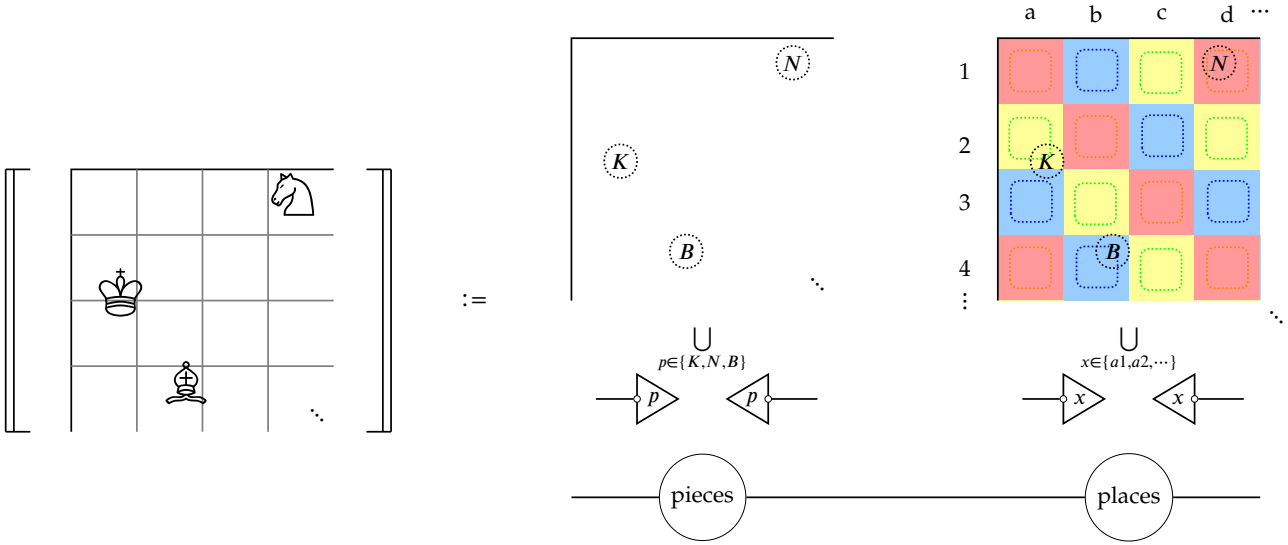


Example 3.5.2 (Where is a piece on a chessboard?)**.** How is it that we quotient away the continuous structure of positions on a chessboard to locate pieces among a discrete set of squares? Evidently shifting a piece a little off the centre of a square doesn't change the state of the game, and this resistance to small perturbations suggests that a topological model is appropriate. We construct two spiders, one for pieces, and one for places on the chessboard. For the spider that represents the position of pieces, we open balls of some radius r, and we consider the places spider to consist of square halos (which tile the chessboard), containing a core inset by the same radius r; in this way, any piece can only overlap at most one square. As a technical aside, to keep the core of the tiles open, we can choose an arbitrarily sharp curvature ϵ at the corners.



Now we observe that the calculation of positions corresponds to composing sticky spiders. We take the initial state to be the sticky spider that assigns a ball of radius r on the board for each piece. We can then obtain the set of positions of each piece by composing with the places spider. The composite (pieces;places) will send the king to a2, the bishop to b4, and the knight to d1, i.e. $\langle K| \mapsto \langle a2|$, $\langle B| \mapsto \langle b4|$ and $\langle N| \mapsto \langle d1|$. In other words, we have obtained a process that models how we pass from continuous states-of-affairs on a physical

chessboard to an abstract and discrete game-state.



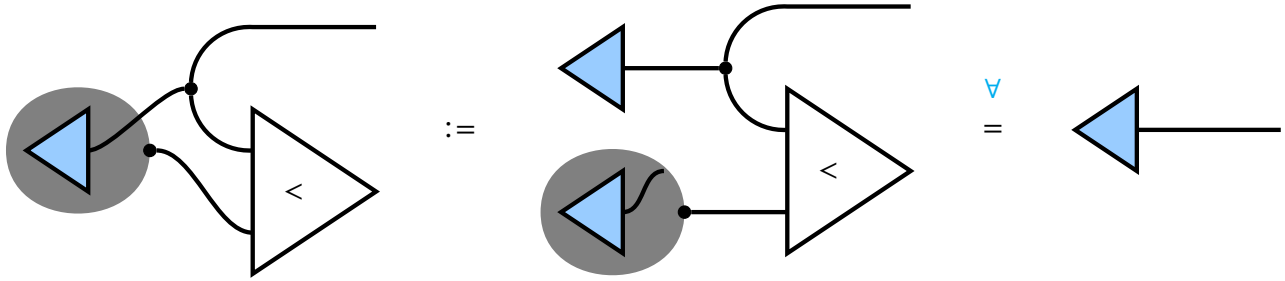
3.5.2 *The unit interval*

To begin modelling more complex concepts, we first need to extend our topological tools. If we have the unit interval, we can begin to define what it would mean for spaces to be connected (by drawing lines between points in those spaces), and we can also move towards defining motion as movement along a line. There are many spaces homeomorphic to the real line. How do we know when we have one of them? The following theorem provides an answer:

Theorem 3.5.3 (Friedman)**.** Let $\big((X, \tau), <\big)$ be a topological space with a total order. If there exists a continuous map $f : X \times X \to X$ such that $\forall a, b_{\in X} : a < f(a, b) < b$, then X is homeomorphic to $\mathbb{R}$. CITE

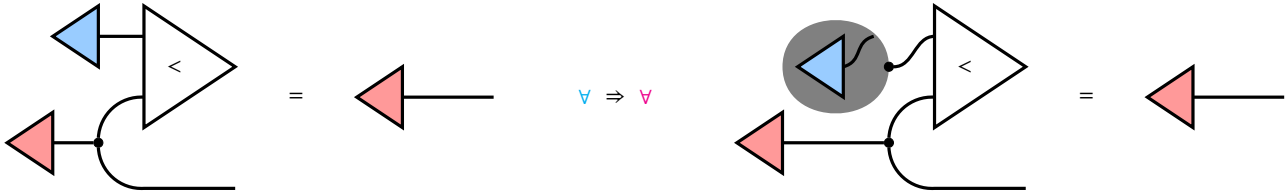
Using the technology in the margins, we can define:

Upper Bound



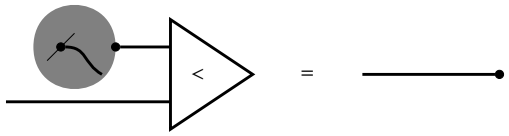
And we can add in further equations governing the upper bound endocombinator to turn it into a supremum, where the lower endpoint is obtained as the supremum of the empty set, and the upper endpoint is the supremum of the whole set.

Supremum

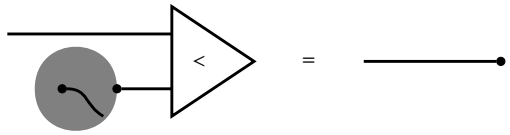


Now we can define endpoints purely graphically:

Lower endpoint



Upper endpoint

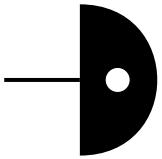


Going forward, we will denote the unit interval using a thick dotted wire.

Simply connected spaces

Once we have a unit interval, we can define the usual topological notion of a simply connected space: one where any two points can be connected by a continuous line without leaving the space.

Simple connectivity is a useful enough concept that we will notate simply connected open sets as follows, where the hole is a reminder that simply connected spaces might still have holes in them.



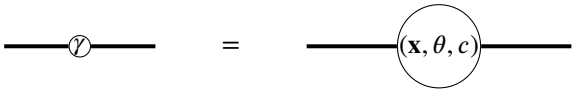
3.5.3 *Displacing shapes*

Static shapes in space are nice, but moving them around would be nicer. So we have to define a stock of concepts to express rigid motion. Rigidity however is a difficult concept to express in topological spaces up to homeomorphism – everyone is well aware of the popular gloss of topology in terms of coffee cups being homeomorphic to donuts. To obtain rigid transformations as we have in Euclidean space, we need to define metrics, and in order to do that, we need addition.

Rigid displacements

Now we return to our sticky spiders. From now we consider sticky spiders on the open unit square, so that we can speak of shapes on a canvas. Now we will try to displace the shapes of a sticky spider. We know the planar isometries of Euclidean space can be expressed as a translation, rotation, and a bit to indicate the chirality of the shape – as mirror reflections are also an isometry.

Isometries of $\mathbf{R}^2$



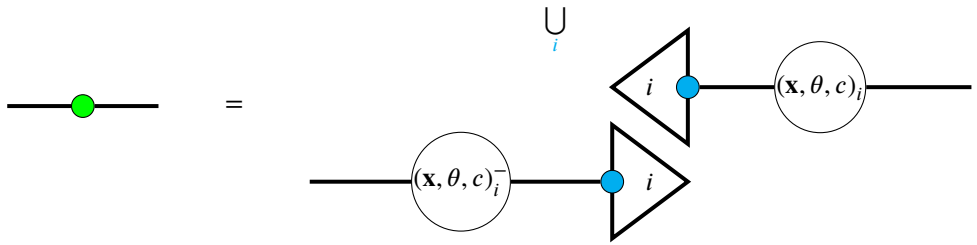
$$\mathbf{x} \in \mathbf{R}^2$$

$$\theta \in S^1 \simeq [0, 2\pi)$$

$$c \in \{-1, 1\}$$

With this in mind, we have the following condition relating different spiders, telling us when one is the same as the other up to rigidly displacing shapes.

Rigid displacement



Chirality leaves us with a wrinkle: in flatland, we do not expect shapes to suddenly flip over. We would like to express just those rigid transformations that leave the chirality of the shape intact, because really we want to only be able to slide the shapes around the canvas, not leave the canvas to flip over. So we go on to define rigid continuous motion in flatland.

3.5.4 *Moving shapes*

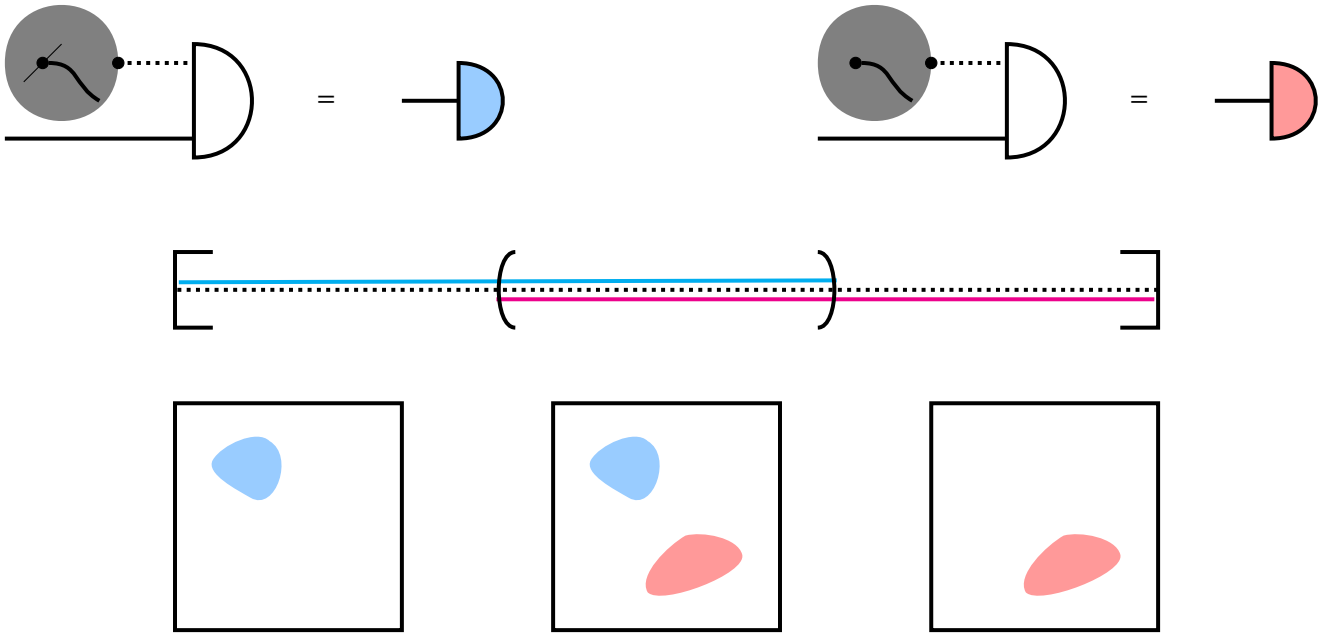
If we want continuous transformations in the plane from the configuration of shapes in one spider to end at the configuration of shapes in another, we ought to define an analogue of *homotopy*: the continuous deformation of one map to another. However, we will have to massage the definition a little to work in our setting of continuous relations.

Homotopy in **ContRel**

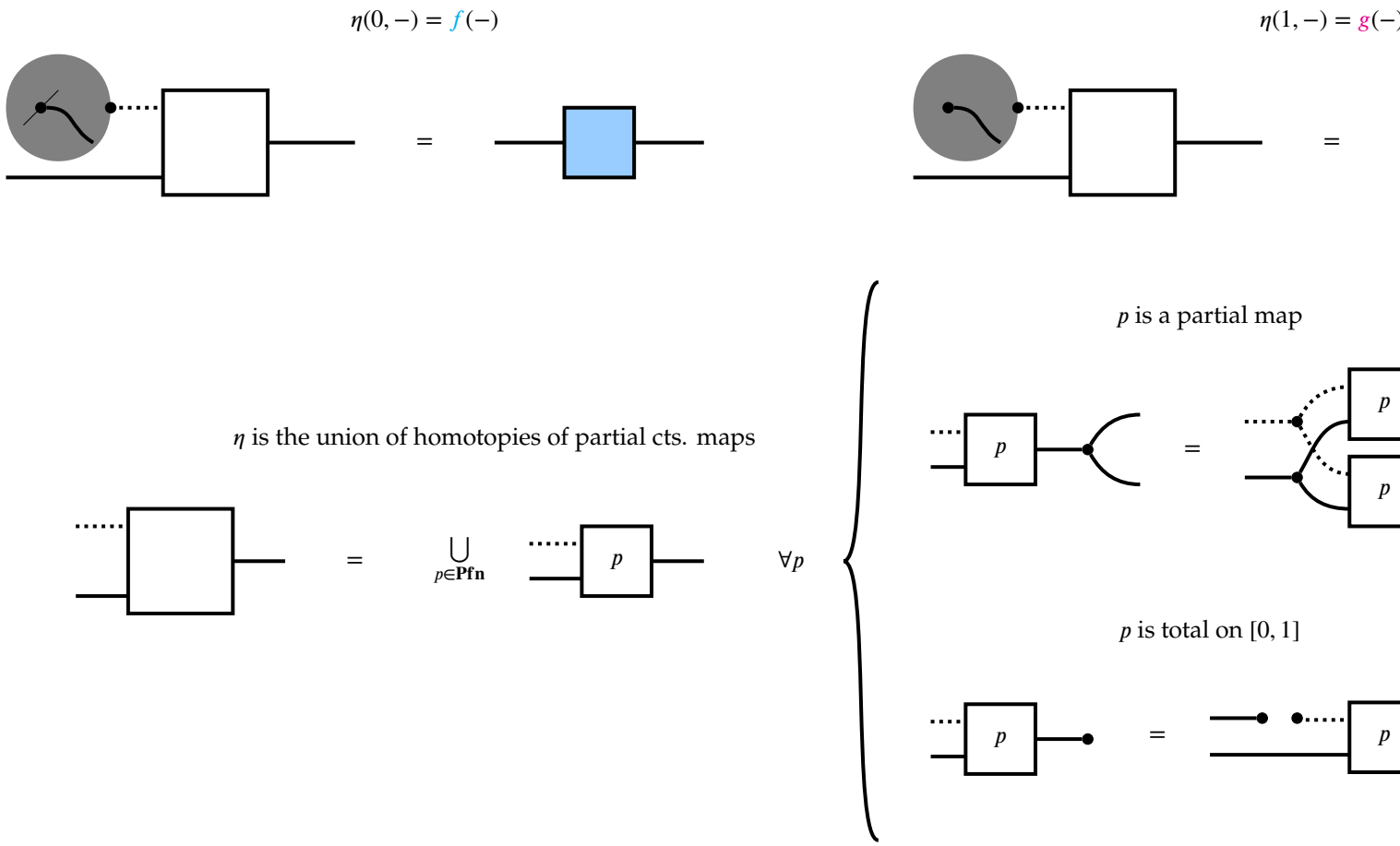
Usually, when we are restricted to speaking of topological spaces and continuous functions, a homotopy is defined:

Definition 3.5.12 (Homotopy in **Top**)**.** where f and g are continuous maps $A \to B$, a *homotopy* $\eta : f \Rightarrow g$ is a continuous function $\eta : [0,1] \times A \to B$ such that $\eta(0,-) = f(-)$ and $\eta(1,-) = g(1,-)$.

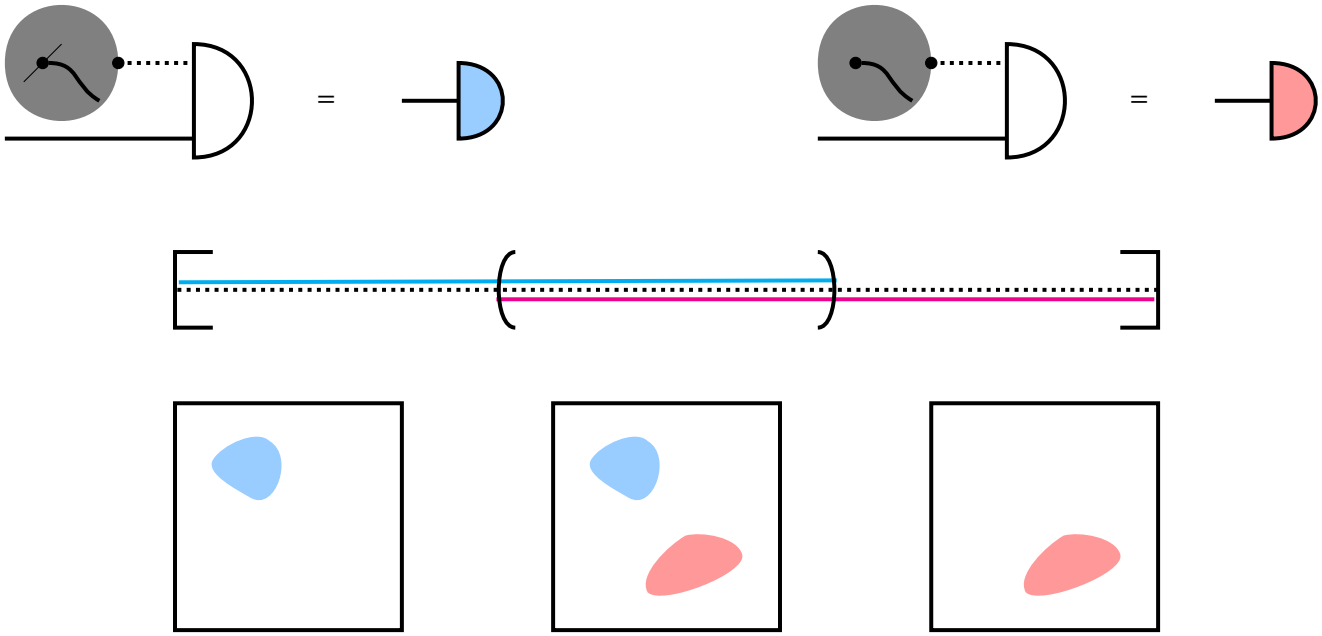
In other words, a homotopy is like a short film where at the beginning there is an f, which continuously deforms to end the film being a g. Directly replacing "function" with "relation" in the above definition does not quite do what we want, because we would be able to define the following "homotopy" between open sets.



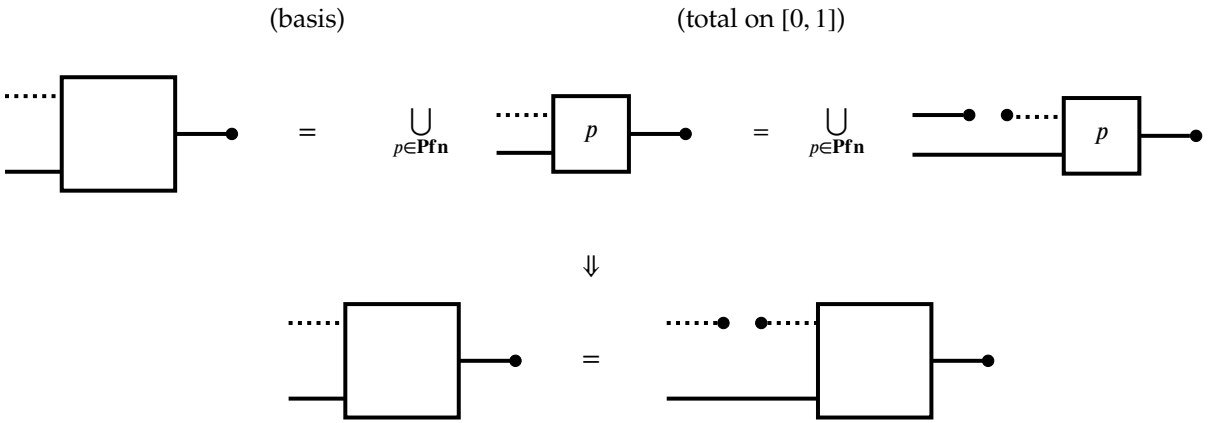
What is happening in the above film is that we have our starting open set, which stays constant for a while. Then suddenly the ending open set appears, the starting open disappears, and we are left with our ending; while *technically* there was no discontinuous jump, this isn't the notion of sliding we want. The exemplified issue is that we can patch together (by union of continuous relations) vignettes of continuous relations that are not individually total on $[0, 1]$. We can patch this problem by asking for homotopies in **ContRel** to satisfy the additional condition that they are expressible as a union of continuous partial maps that are total on the unit interval.



Observe that the second condition asking for decomposition in terms of partial comes for free by Proposition 3.2.19; the constraint of the definition is provided by the first condition, which is a stronger condition than just asking that the original continuous relation be total on I:

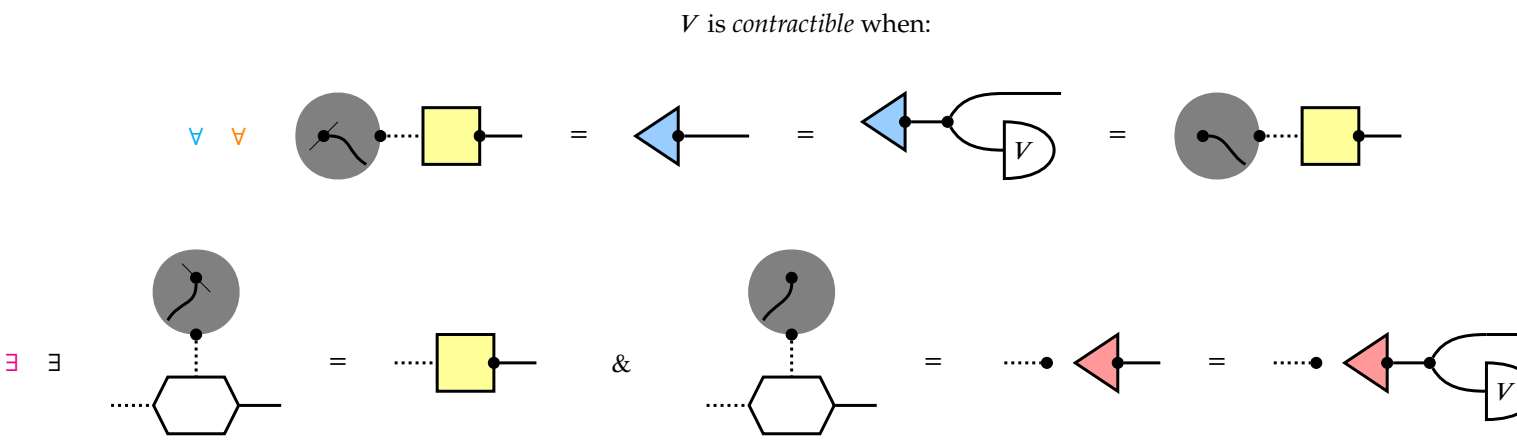


This definition is "natural" in light of Proposition 3.2.19, that the partial continuous functions $A \to B$ form a basis for **ContRel**(A, B): we are just asking that homotopies between partial continuous functions – which can be viewed as regular homotopies with domain restricted to the subspace topology induced by an open set – form a basis for homotopies between continuous relations.

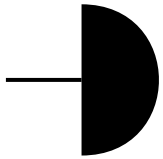


Contractible spaces

With homotopies in hand, we can define a stronger notion of connected shapes with no holes, which are usually called *contractible*. The reason for the terminology reflects the method by which we can guarantee a shape in flatland has no holes: when any loop in the shape is *contractible* to a point.



Contractible open sets are worth their own notation too; a solid black effect, this time with no hole.



3.5.5 *Rigid motion*

Configuration spaces

3.5.6 *Modelling linguistic topological concepts*

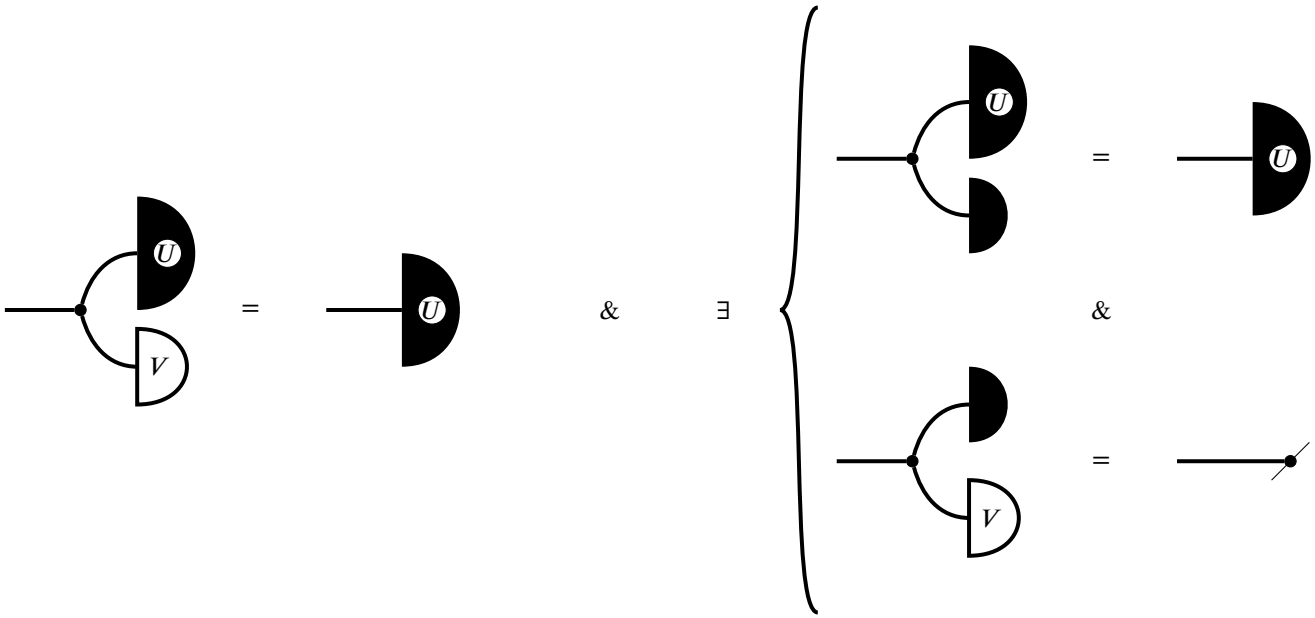
By "linguistic", I mean to refer to the kinds of concepts we use in everyday language. These are concepts that even young children have an intuitive grasp of [], but their formal definitions are difficult to pin down. One such relation modelled here – touching – is in fact a *semantic prime* []: a word that is present in essentially all natural languages that is conceptually primitive, in the sense that it resists definition in simpler terms. It is among the ranks of concepts like *wanting* or *living*, words that are understood by the experience of being

human, rather than by school. As such, I make no claim that these definitions are "correct" or "canonical", just that they are good enough to build upon moving forward.

Parthood

Let's say that a "part" refers to an entire simply connected component. Simply connected is already a concept in our toolkit. A shape U is disjoint from another shape V intuitively when we can cover U in a blob with no holes such that the blob has no overlap with V. So, U is a part of V when it is simply connect, wholly contained in V, and there exists a contractible open that is disjoint from V that covers U. Diagrammatically, this is:

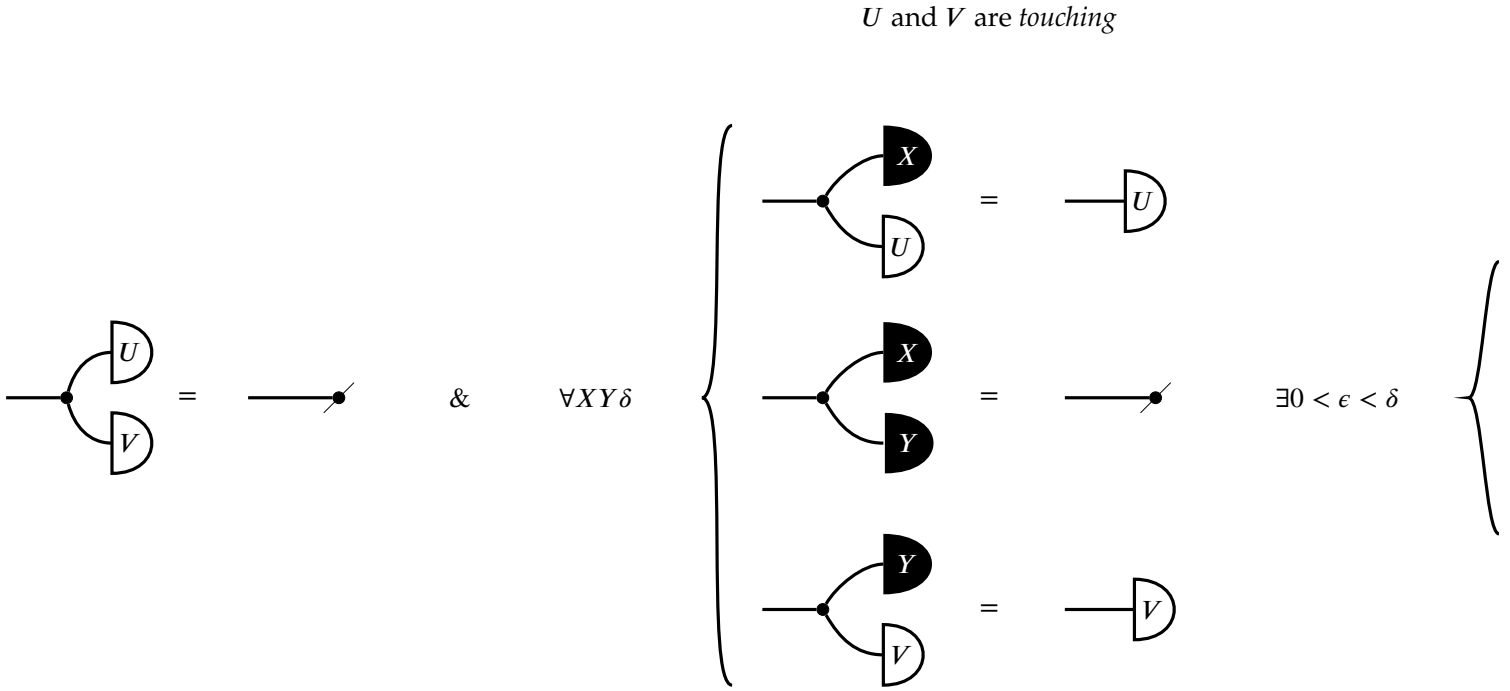
U is *a part of* V



Touching

Let's distinguish touching from overlap. Two shapes are "touching" intuitively when they are as close as they can be to each other, somewhere; any closer and they would overlap. Let's assume that we can restrict our attention to the parts of the shape that are touching, and that we can fill in the holes of these parts. At the point of touching, there is an infinitesimal gap – just as when we touch things in meatspace, there is a very small gap between us and the object due to the repulsive electromagnetic force between atoms. To deal with

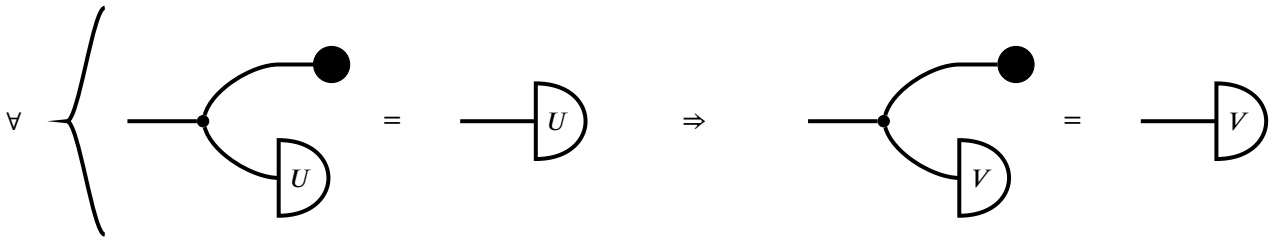
infinitesimals we borrow the $\epsilon - \delta$ trick from mathematical analysis; for any arbitrarily small δ, we can pick an even smaller ball of radius ϵ such that if we stick the ball in the gap, the ball forms a bridge that overlaps the two filled-in shapes, which allows us to draw a continuous line between them. Diagrammatically, this is:



Within

If U surrounds V, or equivalently, if V is within U, then we are saying that leaving V in almost any direction, we will see some of U before we go off to infinity. We can once again use open balls for this purpose, which correspond to possible places you can get to from a starting point $\mathbf{x}$ within a distance ϵ. In prose, we are asking that any open ball that contains all of U must also contain all of V.

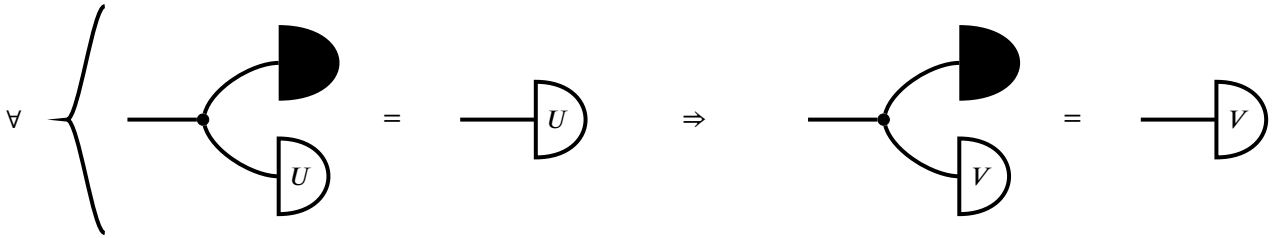
V is *within U*, or *U surrounds V*



Containers and enclosure

There is a strong version of within-ness, which we will call enclosure. As in when we are in elevators and the door is shut, nothing gets in or out of the container. Intuitively, there is a hole in the container surrounded on all sides, and the contained shape lives within the hole. To give a real-world example, honey lives within a honeycomb cell in a beehive, but whether the honey is enclosed in the cell depends on whether it is sealed off from air with beeswax. So in prose we are asking that any way we fill in the holes of the container with a blob, that blob must cover the contained shape. Diagrammatically, this amounts to levelling up from open balls in our previous definition to contractible sets:

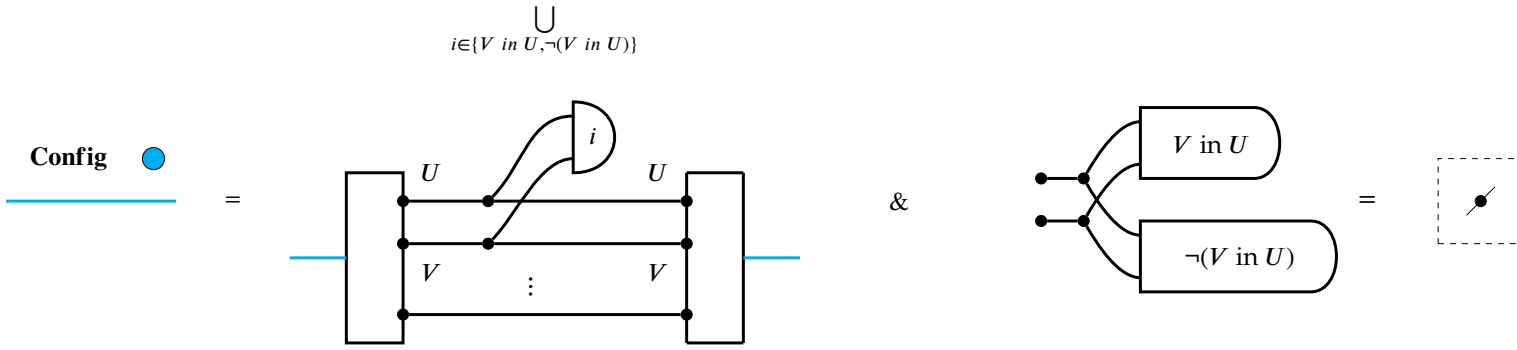
U encloses V



Trapped

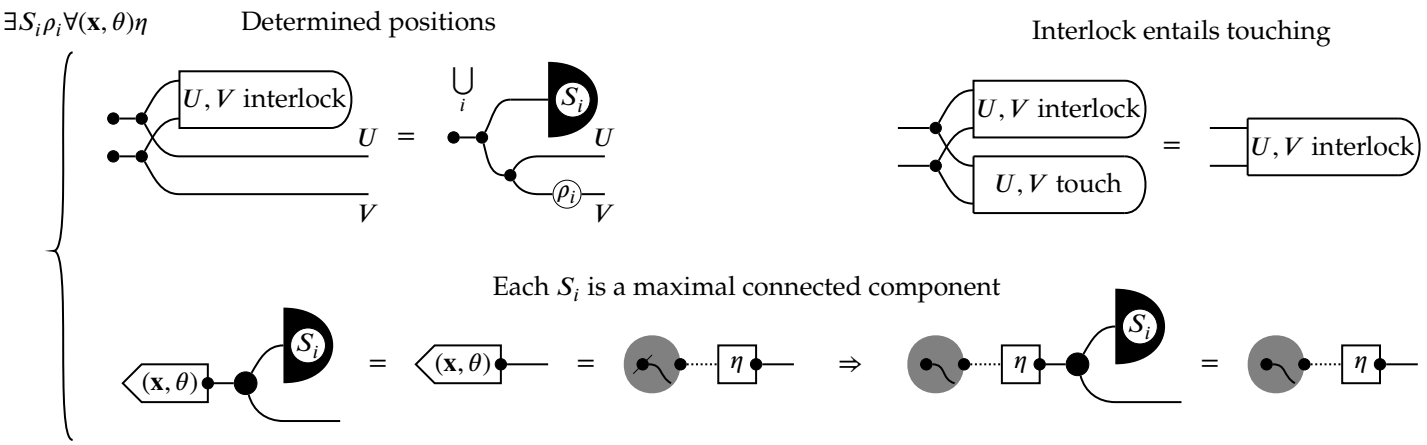
There is an intermediate notion between within-ness and enclosure; for instance, standing in the stonehenge you are surrounded by the pillars, but you can always walk away, whereas if the pillars are very close, such as the bars of a jail cell, a human would not be able to leave the trap while still being able to see the outside. The difficulty here is that relative sizes come into play: small animals would still consider it a case of mere within-ness, because they can still walk away between the bars. So we would like to say that no matter

how the pair of objects move rigidly, being trapped means that the trapped V stays within U. In other words, that in configuration space, if we forget about all other shapes, we can partition our space of configurations by two concepts, whether V is within U or not, and moreover that these two components is disjoint – i.e. not simply connected – so there is no rigid motion that can allow V to escape from being within U if V starts off trapped inside in U.



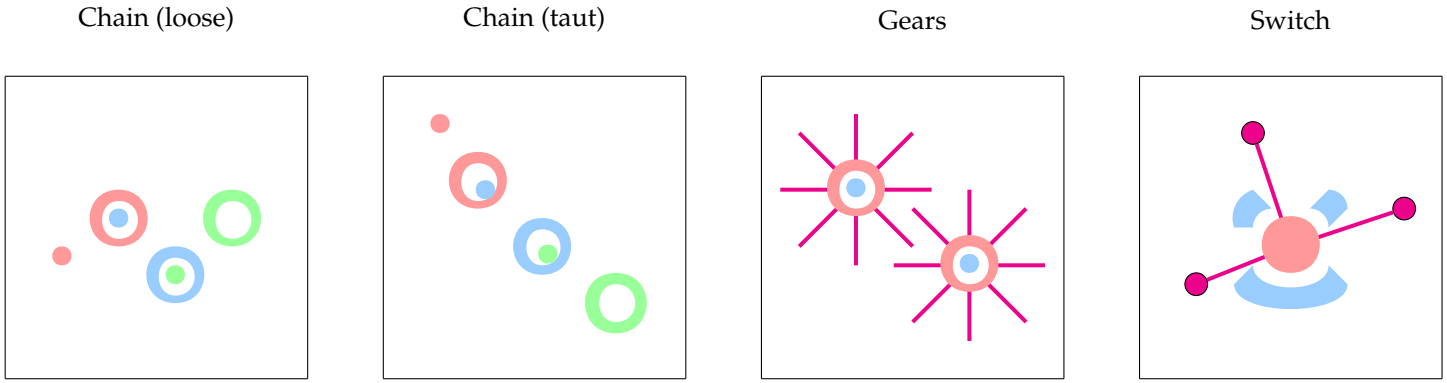
Interlocked

Two shapes might be tightly interlocked without being inside one another. Some potentially familiar examples are plastic models of molecular structure that we encounter in school, metal lids in cold weather that are too tightly hugging the glass jar, or stubborn Lego pieces that refuse to come apart. The commonality of all these cases is that the two shapes must move together as one, unless deformed or broken. In other words, when two shapes are interlocked, knowing the position in space of one shape determines the position of the other, and this determination is a fixed isometry of space. So we only need to specify a range of positions S for the entire subconfiguration of interlocked shapes U and V, and we may obtain their respective positions by a fixed rigid motion ρ. Since objects may interlock in multiple ways, we may have a sum of these expressions. We additionally observe that interlocking shapes should also be touching, which translates to containment inside the touching concept. Finally, we observe that as in the case of entrapment and enclosure, rigid motions are interlocking-invariant, which translates diagrammatically to the constraint that each S, ρ expression is an entire connected component in configuration space.



CONSTRAINED MOTION

A weaker notion of interlocking is when shapes only imperfectly determine each other's potential displacements, by specifying an allowed range. Here is an understatement: there is some interest in studying how shapes mutually constrain each other's movements in this way.

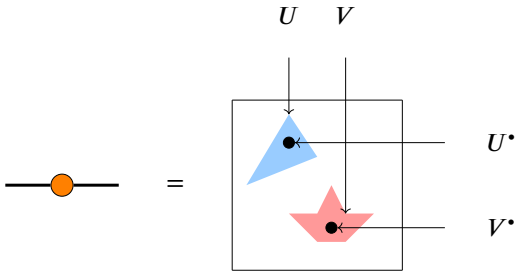


There are as many definitions to go through here as there are potential mechanical models, and among other things, there are mechanically realised clocks [], computers [], and analogues of electric circuits []. So instead, we will allow ourselves to additionally specify open sets as concepts in configuration space that correspond to whatever mechanical concepts we please, and we assure the reader seeking rigour that blueprints exist for all the mechanisms humans have built. Of course in reality mechanical motions are reversible among rigid objects, and directional behaviour is provided by a source of energy, such as gravitational potential, or

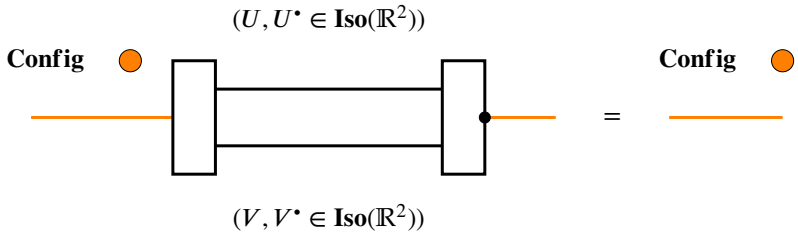
wound springs. But we may in principle replace these sources of energy by a belt that we choose to spin in one direction – our own arrow of time. We postpone discussion of causal-mechanistic understanding and analogy for a later section.

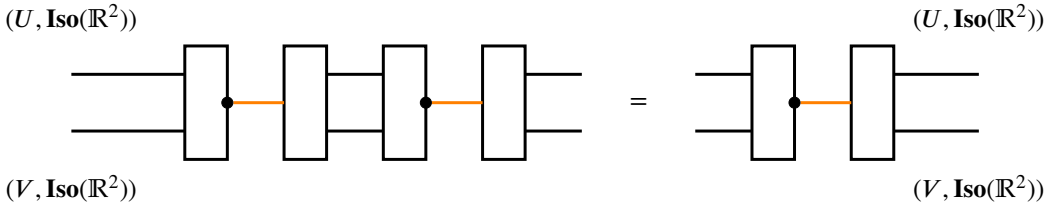
3.5.7 *States, actions, manner*

Configuration space explains why we label noun wires: each wire in expanded configuration space must be labelled with the shape within the sticky spider it corresponds to so that the section and retract know how to reconstruct the shapes, since each shape may have a different spatial extent.



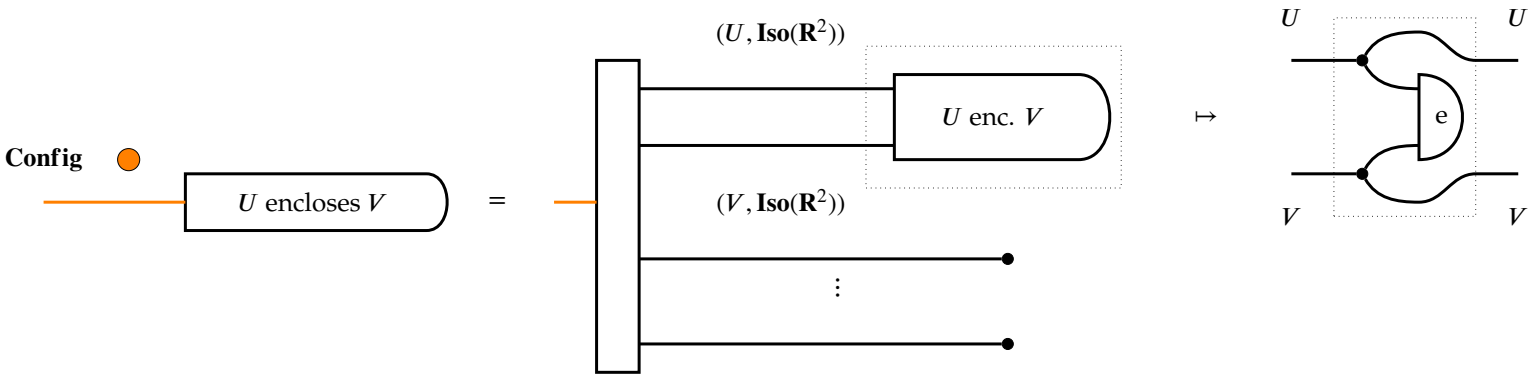
Concretely, for each shape in a sticky spider,
in order to reconstruct the shape,
the data of configuration space
only has to remember a basepoint
(which the isometries act upon)
paired with a label naming the shape
(so that the extent of the shape is reconstructible)



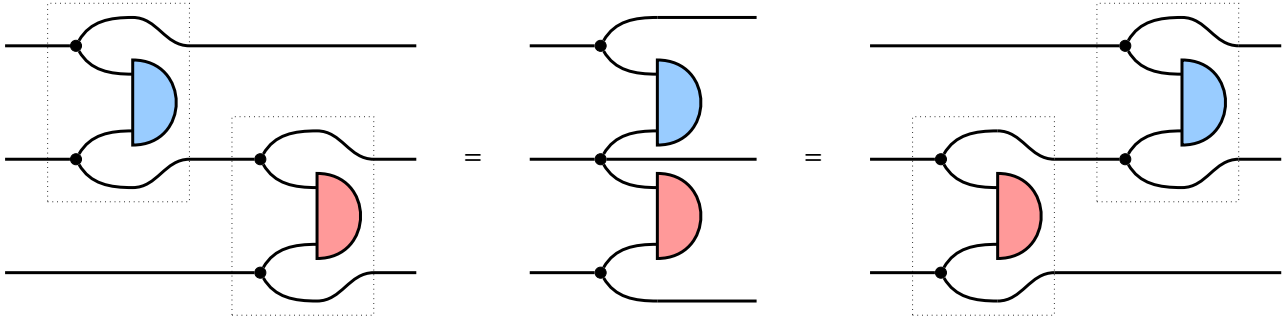


All of the concepts we have defined so far are open sets in configuration space – and for any concept that isn't, we are always free to take the interior of the set; the largest open set contained within the concept. Pass-

ing through the split idempotent, we can recast each as a circuit gate using copy maps.



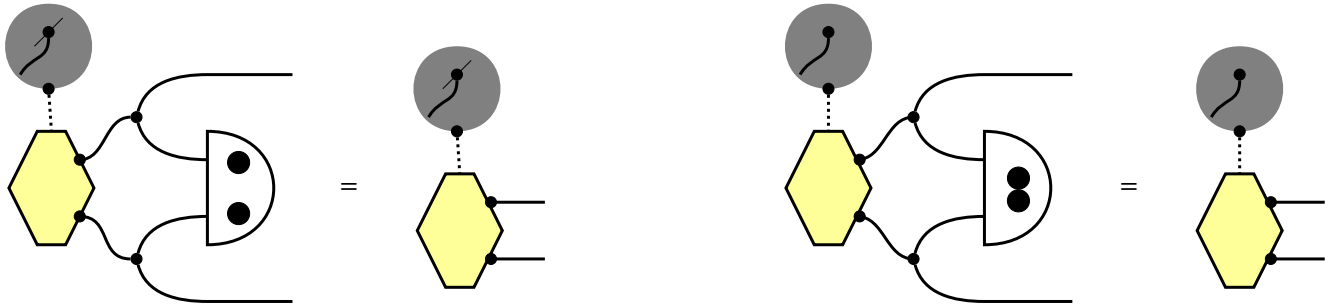
Going forward, we will just label the wires with the names of each shape when necessary. We notice that one feature of this procedure to get gates from open sets is that all gates commute, due to the commutativity of copy.



Moreover, since each gate of this form is a restriction to an open set, the gates are idempotent. So the concepts we have defined so far behave as if describing *states* of affairs in space, as if we adding commuting adjectives to space to elaborate detail. For example, `fast red car`, `fast car that is red`, `car is (red and fast)` all mean the same thing. As we add on progressively more concepts, we get diminishing subspaces of configurations in the intersection of all the concepts. So the natural extension is to ask how states of affairs can change with motion. A simple example is the case of *collision*, where two shapes start off not touching, and

then they move rigidly towards one another to end up touching.

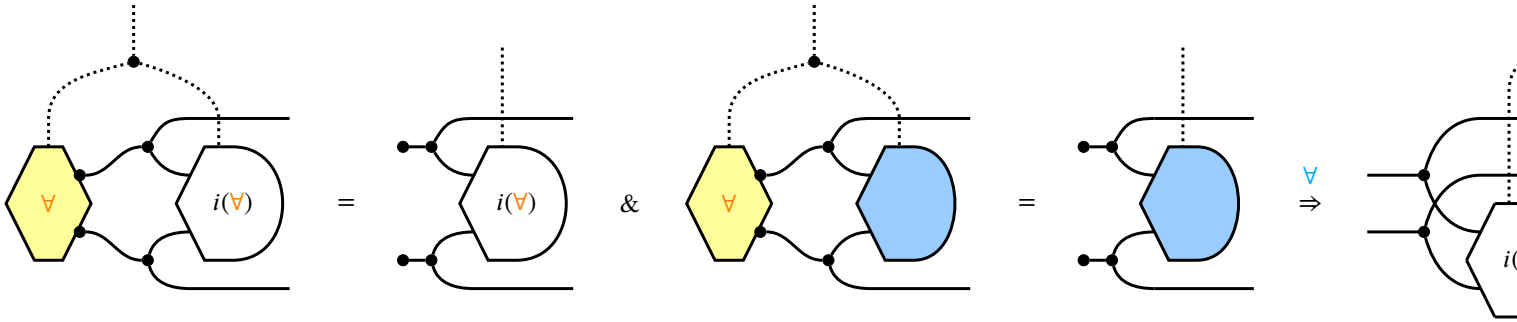
A particular collision trajectory



At $t = 0$, the shapes do not touch

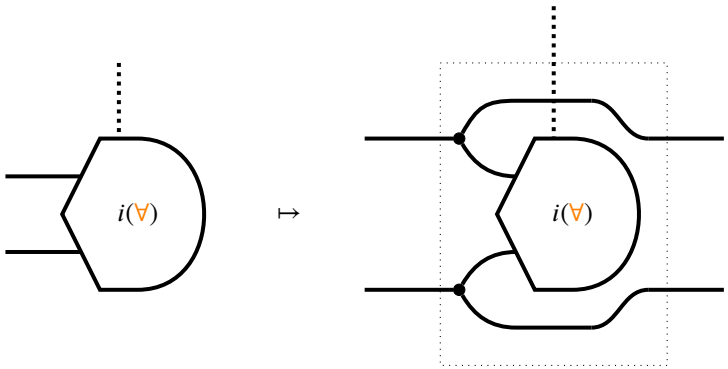
At $t = 1$, the shapes touch

Recalling that homotopies between relations are the unions of homotopies between maps, we have a homotopy that is the union of all collision trajectories, which we mark $\forall$. Now we seek to define the interior $i(\forall)$ as the concept of collision; the expressible collection of all particular collisions. But this is not just an open set on the potential configuration of shapes, it is a collection of open sets parameterised by homotopy.



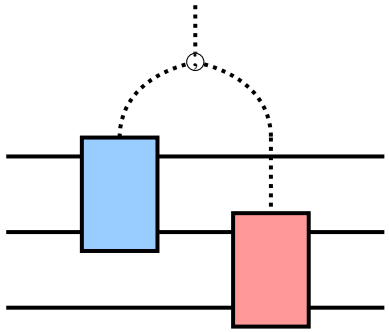
Once we have the open set $i(\forall)$ that corresponds to all expressible collisions, we have a homotopy-parameterised gate. Following a similar procedure, we can construct gates of motion that satisfy whatever pre- and post-

conditions we like.

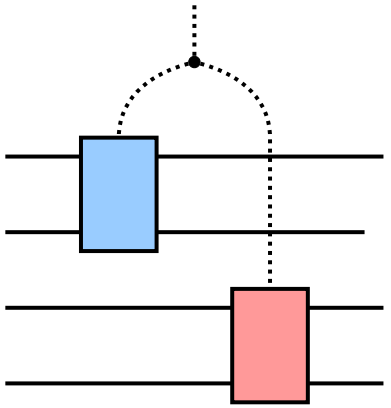


We can compose multiple rigid motions sequentially by a continuous function ; that splits a single unit interval into two: ; $:= x \mapsto \begin{cases} (2x, 0) \text{ if } x \in [0, \frac{1}{2}) \\ (1, 2x-1) \text{ if} x \in [\frac{1}{2}, 1] \end{cases}$. The effect of the map is to splice two vignettes of the same length together by doubling their speed, then placing them one after the other. We can achieve the same thing without resorting to units of measurement, because recall by Theorem 3.5.3 and by construction that we have access to a map that selects midpoints for us; we will revisit a string-diagrammatic treatment of homotopy and tenses in a later section. We can also compose multiple motions in parallel by copying the unit interval, allowing it to parameterise multiple gates simultaneously.

Sequential composition of motions

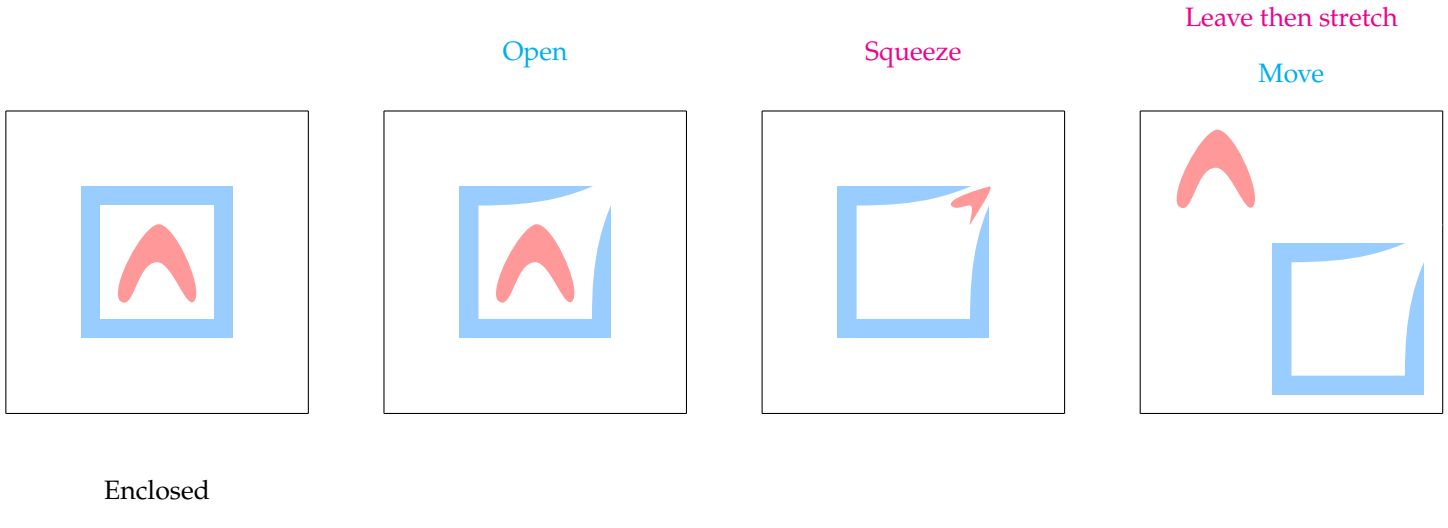


Parallel composition of motions

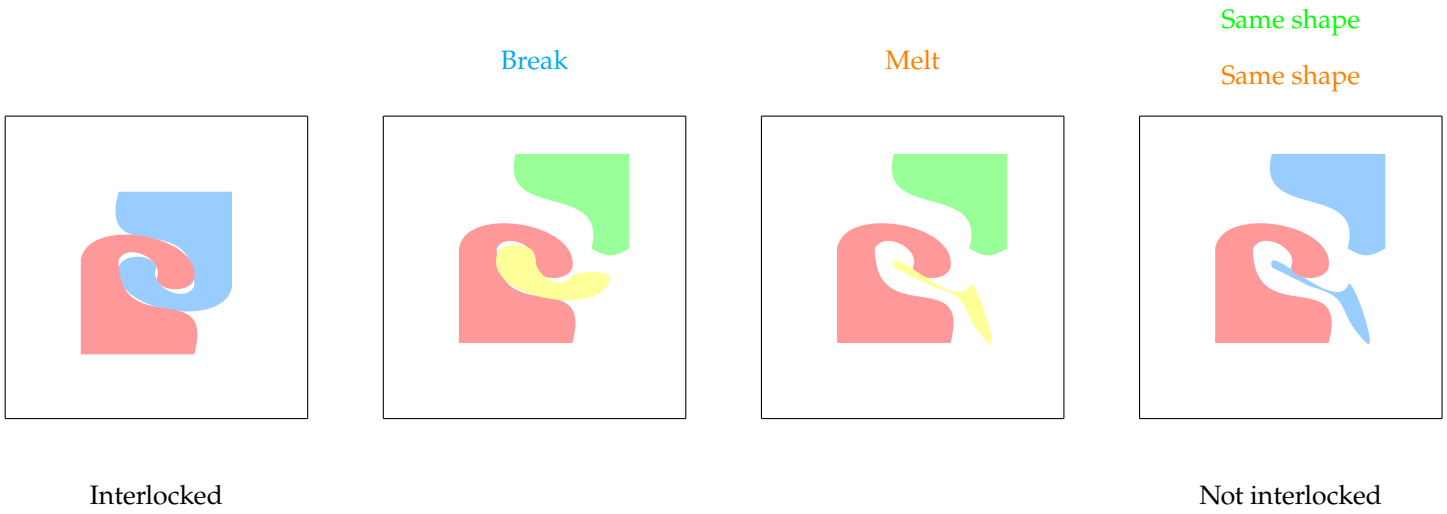


It is easy to see that the gates can always be rewritten to respect the composition order given by ; and copy, since for any input point at the unit interval the gates behave as restrictions to open sets. These new gates do

not generally commute; consider comparing the situation where a tenant moves into one apartment and then another, with the situation where the tenant reverses the order of the apartments. These are different paths, as the postconditions must be different. So now we have noncommuting gates that model *actions*, or verbs. What kinds of actions are there? In our toy setting, in general we can define actions that arbitrarily change states of affairs if we do not restrict ourselves to rigid motions. The trick to doing this is the observation that arbitrary homotopies allow deformations, so our verb gates allow shapes to shrink and open and bend in the process of a homotopy, as long as at the end they arrive at a rigid displacement of their original form.



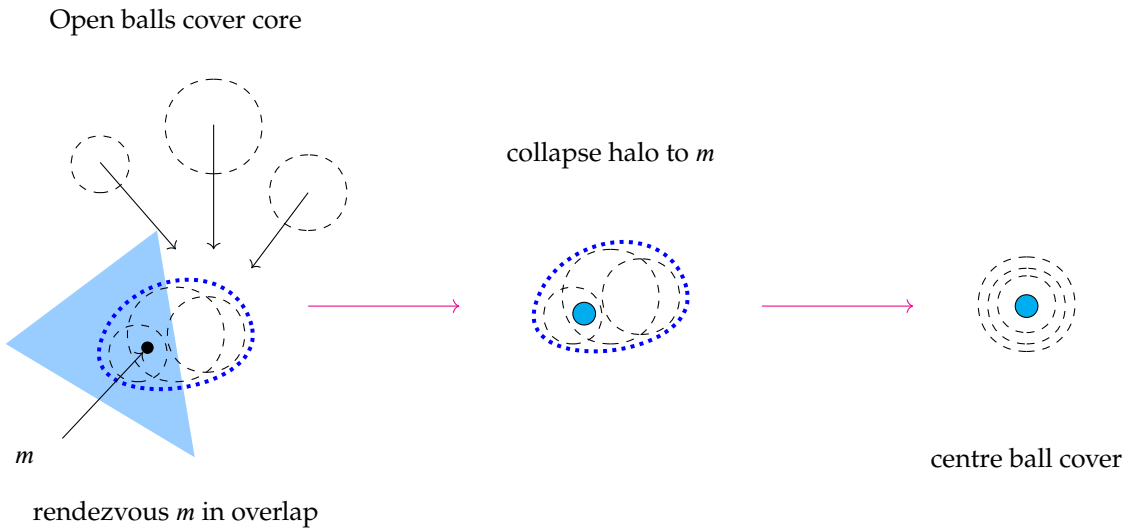
We can further generalise by noting that completely different spiders can be related by homotopy, so we can model a situation where there is a permanent bend, or how a rigid shape might shatter.



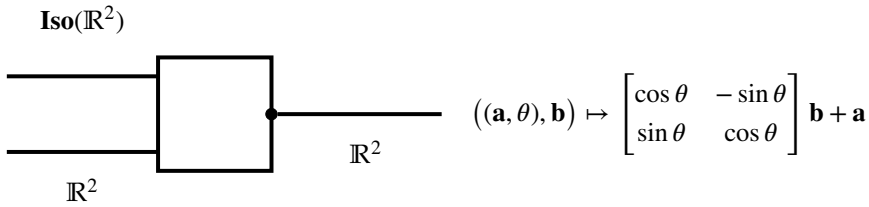
We provide the following construction as a general recipe to construct homotopies between spiders.

Construction 3.5.13 (Morphing sticky spiders with homotopies)**.** We aim to construct homotopies relating (almost) arbitrary sticky spiders. For now we focus on just changing one shape into another arbitrary one. The idea is as follows. First, we need a cover of open balls $\cup\mathcal{J} = T^0$ and $\cup\mathcal{K} = T^1$ of the start and end cores T^0 and T^1 such that each $k \in T^1$ is expressible as a rigid isometry of some core $j \in \mathcal{J}$; this is so we can slide and rearrange open balls comprising T^0 and reconstruct them as T^1. As an intermediate step to eliminate holes and unify connected components, we gather all of the balls at a meeting point m (to be determined shortly.)

Intuitively we can illustrate this process as follows:

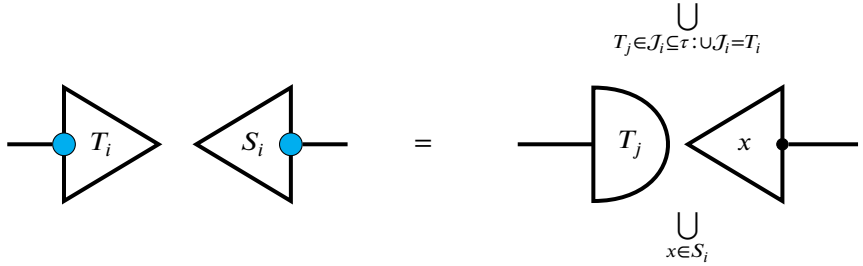


Second, in order to perform the sliding of open balls, we observe that, given a basepoint to act as origin (which we assume is provided by the data of the split idempotent of configuration space) we can express the group action of rigid isometries $\mathbf{Iso}(\mathbf{R}^2)$ on $\mathbf{R}^2$ as a continuous function:



Third, before we begin sliding the open balls, we must ensure that the halo of the shape cooperates. We observe that a given shape i in a sticky spider may be expressed as the union of a family of constant continuous partial functions in the following way. Given an open cover $\mathcal{J}$ such that $\cup\mathcal{J} = T_i$, where T_i is the core of the shape i, each function is a constant map from some $T_j \in \mathcal{J}$ to some point $x \in S_i$, where S_i is the halo of the shape i. For each $T_j \in \mathcal{J}$ and every point $x \in S_i$, the constant partial function that maps T_j to x is in the

family.



By definition of sticky spiders, there must exist some point m that is in both the core and the halo: we pick such a point as the rendezvous for the open balls. For each partial map in the family, we provide a homotopy that varies only the image point x continuously in the space to finish at m. Now we can slide the open balls to the rendezvous m. Since homotopies are reversible by the continuous map $t \mapsto (1-t)$ on the interval, we can perform the above steps for shapes T^0 and T^1 to finish at the same open ball, reversing the process for T^1 and composing sequentially to obtain a finished transformation. The final wrinkle to address is when dealing with multiple shapes. Recalling our exclusion conditions **??** for shapes, it may be that parts of one shape are enclosed in another, so the processes must be coordinated so that there are no overlaps. For example, the enclosing shape must be first opened, so that the enclosed shape may leave. I will keep it an article of faith that such coordinations exist. I struggle to come up with a proof that all spiders $\mathbf{R}^2$ are mutually transformable by homotopy in this (or any other) way, so that will remain a conjecture. But it is clear that a great deal of spiders are mutually transformable; almost certainly any we would care to draw. So this will just be a construction for now.

GOING FORWARD, I WILL CONSIDER ANY LINGUISTIC SEMANTICS THAT CAN BE GROUNDED BY A MECHANICAL OR TABLETOP MODEL TO BE FORMAL. The preceding analysis extends to talk of rigid and deforming bodies and the manner, order, and coordination of their movement and interaction three-dimensional Euclidean space. At this point, I have sketched out enough to, in principle, linguistically specify mechanical models. Further, by Example 3.5.2, we have enough technology to speak of locations in space, so we have access to "tabletop semantics": anything that in principle can be represented by counters and meeples in a boardgame, with for instance reserved spaces on the board for health and hunger and whatever else is necessary. Wherever this talk falls short, I consider videogame design to be applied formal semantics, so I permit myself more or less any conceivable interactive world with its own internal logic.

OBJECTION: THAT IS WAY OUTSIDE THE SCOPE OF FORMAL SEMANTICS. Insofar as semantics is sensemaking, we certainly are capable of making sense of things in terms of mechanical models and games by means of metaphor, the mathematical treatment of which is concern of Section 4.1.2, so I claim that I am, definitionally,

doing formal semantics for natural language. Whether or not I'm exceeding the scope of what a linguist might consider formal semantics is ultimately irrelevant, as I am not ultimately concerned with the modal human mechanism. There is maybe also a prejudice that formal semantics must necessarily resolve in some symbolic logic, to which I might charitably respond that I'm working with algebraic system, just not a one-dimensional one. Less charitably, I don't care what these people think.

3.6 *Interpreting text circuits in **ContRel***

Recall that there were some mathematically odd choices in conventions for text circuits, such as the labelling of noun wires and some subtleties with respect to interchange of parallel gates and text order. The explanation of those choices was deferred "to semantics", and since we're here now we have to make good. The aim of this section is to now explain those choices through an interpretation of text circuits in **ContRel**.

3.6.1 *Sticky spiders: iconic semantics of nouns*

Without loss of generality we may consider sticky spiders to be set-indexed collections of disjoint open subsets of an ambient space, i.e. a collection of shapes in space that a labelled from a set of names. So particular sticky spiders will be particular models of a collection of nouns; the indexing set names them, and the corresponding shapes in space are a particular iconic representation of those nouns in space.

3.6.2 *Open sets: concepts*

Apart from enabling us to paint pictures with words, **ContRel** is worth the trouble because the opens of topological spaces crudely model how we talk about concepts, and the points of a topological space crudely model instances of concepts. We consider these open-set tests to correspond to "concepts", such as redness or quickness of motion. Figure **??** generalises to a sketch argument that insofar as we conceive of concepts in (possibly abstractly) spatial terms, the meanings of words are modellable as shared strategies for spatial deixis; absolute precision is communicatively impossible, and the next best thing mathematically requires topology.

This may explain the asymmetry of why tests are open sets, but why are states allowed to be arbitrary subsets? One could argue that states in this model represent what is conceived or perceived. Suppose we have an analog photograph whether in hand or in mind, and we want to remark on a particular shade of red in some uniform patch of the photograph. As in the case of pointing out a point on the real interval, we have successively finer approximations with a vocabulary of concepts: "red", "burgundy", "hex code #800021"... but never the point in colourspace itself. If someone takes our linguistic description of the colour and tries to reproduce it, they will be off in a manner that we can in principle detect, cognize, and correct: "make it a little darker" or "add a little blue to it". That is to say, there are in principle differences in mind that we cannot distinguish linguistically in a finite manner; we would have to continue the process of "even darker" and "add a bit less blue than last time" forever. All this is just the mathematical formulation of a very common observation: sometimes you cannot do an experience justice with words, and you eventually give up with "I guess you just had to be there". Yet the experience is there and we can perform linguistic operations on it, and the states accommodate this.

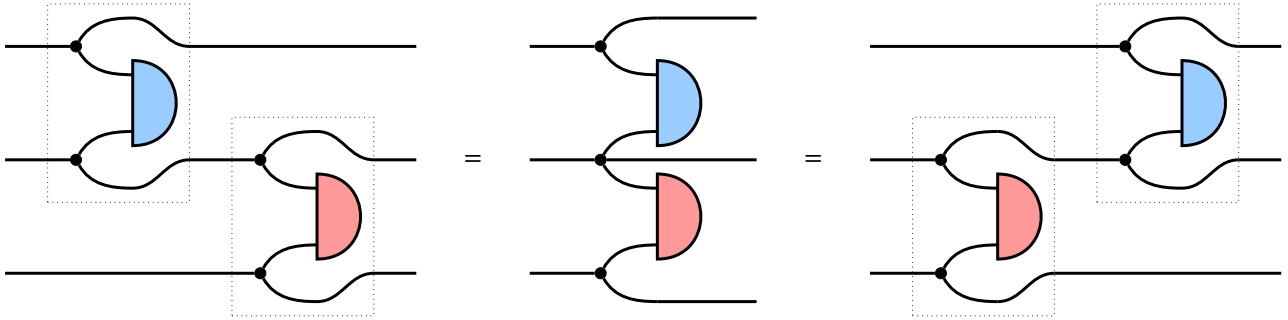
3.6.3 *Configuration spaces: labelled noun wires*

Whereas sticky spiders correspond to particular iconic representations, we want to interpret text circuits in the configuration spaces of sticky spiders, so that text process-theoretically restricts, expands, and modifies a space of permissible iconic representations[1]. So it is via configuration spaces that we aim for an iconic semantics of general text. Recall from the definition of configuration spaces via split idempotents REF that the section and retract diagrammatically allow the configuration space wire to open up to guitar strings we can place our circuits in. However, this section and retract pair is not uniquely determined. For example, in the case of configuration spaces of rigid transformations of shapes, one obtains a family of encodings of the configuration space of an n-shape sticky spider in the n-fold tensor of isometry spaces by choosing a basepoint for each shape and rigidly transforming with respect to that basepoint, which on the plane may lead to different reifications of rotating shapes. Interpreting text circuits with respect to a particular split idempotent of configuration space hence gives a good reason to label the noun wires: so that we can remember which space of isometries corresponds to which noun-shape in the absence of knowing how precisely the split idempotent encodes configuration space as the n-fold contribution of spatial possibilities for each noun.

Example 3.6.1 (Differing encodings)**.**

3.6.4 *Copy: stative verbs and adjectives*

Stative verbs are those that posit an unchanging state of affairs, such as `Bob likes drinking`. Insofar as stative verbs are restrictions of all possible configurations to a permissible subset, they are conceptually similar to adjectives, such as `red car`, which restricts permissible representations in colourspace. When we interpret concepts as open-set tests, **ContRel** conspires in our favour by giving us free copy maps on every wire REF . This allows us to define a family of processes that really behave like stative constructions that merely restrict possibilities. The desirable property we obtain is that in the absence of *dynamic* verbs that posit a change in the state of affairs, stative constructions commute in text: if I'm just telling you static properties of the way things are, it doesn't matter in what order I tell you the facts because restrictions commute. Recall:



Example 3.6.2 (Adjectives by analysis of configuration spaces)**.**

3.6.5 Homotopies: dynamic verbs and weak interchange

Dynamic verbs are those that posit a change in state, such as `Bob goes home`. We want to model these verbs by homotopies, where the unit interval parameter models time. A nice and diagrammatically immediate property is that dynamic verbs are obstacles to the commutation of stative words.

Example 3.6.3 (Dynamic verbs block stative commutations)**.**

The composition of dynamic verbs in time also explains the weak-interchange subtlety of text diagrams. When we are dealing with dynamic verbs, the order of sentences does make a difference in semantics between `Bob goes home. Bob gets drunk.` and `Bob gets drunk. Bob goes home.` So we have:

Example 3.6.4 (Sequential versus concurrent)**.**

Three short sketches

I want to sketch in passing three mathematically and linguistically interesting avenues to do with playing with unit-interval parameters that I won't explore fully here. The first is to do with the string-diagrammatic algebra of tenses, which can be modelled like so:

$$before$$

Recall that unit intervals come with a $\leq$ relation and that **ContRel** is a rig category with respect to unions. So suppose we model a tense algebra to be generated by the following PROP, to which we also include the ability to take unions:

$$prop$$

Using these generators it's easy to create more complex temporal relations string-diagrammatically such as `during`, `after`, `between` and so on. The interesting claim would be that *all* temporal relationships in natural languages are *necessarily* obtained in this way. The argument is one from computational feasibility, and the path to it is unexpected. The gist is that the generators along with union correspond to the axioms of an o-minimal structure via interpreting the copy-delete comonoid as product and projection (via Fox's theorem ref), and o-minimal structures are considered good candidates for so-called "tame topologies" that don't have counterintuitive counterexamples that plague the usual definition of topology ref . It turns out that o-minimality and tameness is a sufficient condition for learnability in a formal sense ref , so there lurks an argument for the canonicity of tensed language on the grounds of computational hardness.

Second, there is a strong but little-known criterion for the goodness of a theory of language called Becker's criterion CITE , which is stated "Any theory (or partial theory) of the English Language that is expounded in the English Language must account for (or at least apply to) the text of its own exposition", and followed by the comment: "Using this handy guideline, you can pretty much wipe your theoretical linguistics shelf clean and start over". Text circuits with iconic semantics in **ContRel** appear to have sufficient structure to satisfy Becker's criterion in some sense. Recall that the generative syntactic formulation of text circuits is defined in terms of weak n-categories with strict unitality and associativity but weak interchange. In **ContRel**, the composition of homotopies satisfies the first two strictness conditions and is close to satisfying weak interchange: for any composite gluing of the unit interval, there is a continuous endo-relation on the real line that achieves arbitrary permutation of the constituent intervals (up to finitely many discontinuities, at the gluing points), since continuous relations are unions of partial continuous functions REF . So modulo a model of weak interchange that is insensitive to finitely many discontinuities, and a daunting Gödel-style self-encoding argument, we could have a (perhaps the only) theory of language that suffices to model itself.

Third, the interaction of stative and dynamic (or, more generally, costates intersected by comonoids versus parameterised morphisms) appears to be a suitable model for categorified hypergraphs, just as monoidal categories are categorified monoids. The gist is that any open hypergraph is representable as a morphism in a hypergraph category CITE , and we may use the special frobenius laws to rewrite such morphisms to only use the comonoid part of the spiders in stative form. The non-commutation of statives past dynamics then justifies the view that the dynamic parameterised morphisms can be viewed going between objects enriched in a hypergraph structure.

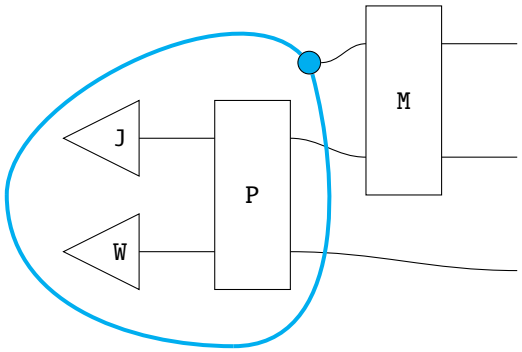
hypergist

3.6.6 Coclosure: adverbs and adpositions

3.6.7 Turing objects: sentential complementation

3.7 On entification, general anaphora, computers, and lassos.

ENTIFICATION IS THE PROCESS OF TURNING WORDS AND PHRASES THAT AREN'T NOUNS INTO NOUNS. We are familiar with morphological operations in English, such as *inflections* that turn the singular `cat` into the plural `cats`, by adding a suffix `-s`. Another morphological operation, generally classed *derivation*, turns words from one category into another, for example the adjective `happy` into the noun `happiness`. With suffixes such as `-ness` and `-ing`, just about any lexical word in English can be turned into a noun, as if lexical words have some semantic content that is independent of the grammatical categories they might wear. I'll call this process *entification*, which extends beyond morphology towards more complicated constructions such as a prefix `the fact that` that converts sentences into noun-like entities, insofar as these entities can be referred to by anaphora: for example, in the sentence `Jono was paid minimum wage but he didn't mind it`, it may be argued that `it` refers to `the fact that Jono was paid minimum wage`. Graphically, we might want to depict the gloss as a circuit with a lasso that gives another noun-wire:



A MATHEMATICAL MODELLING PROBLEM FOR SEMANTICISTS ARISES WHEN ANYTHING CAN BE A NOUN WIRE. The problem at hand is finding the right mathematical setting to interpret and calculate with such lassos. In principle, any meaningful (possibly composite) part of text can be referred to as if it were a noun. For syntax, this is a boon; having entification around means that there is no need to extend the system to accommodate wires for anything apart from nouns, so long as there is a gadget that can turn things into nouns and back. For semantics this is a challenge, since this requires noun-wires to "have enough space in them" to accommodate full circuits operating on other noun-wires, which suggests a very structured sort of infinity.

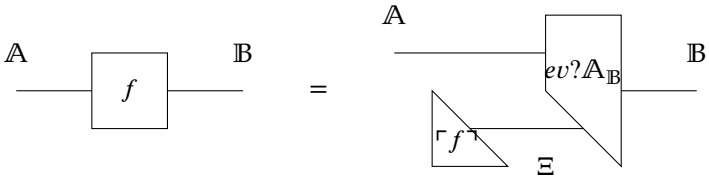
COMPUTER SCIENCE HAS HAD A PERFECTLY SERVICEABLE MODEL OF THIS KIND OF NOUN-WIRE FOR A LONG TIME. What separates a computer from other kinds of machine is that a computer can do whatever any other kind of machine could do – modulo church-turing on computability and the domain of data manipulation

– so long as the computer is running the right *program*. Programs are (for our purposes) processes that manipulate variously formatted – or typed – data, such as integers, sounds, and images. They can operate in sequence and in parallel, and wires can be swapped over each other, so programs form a process theory, where we can reason about the extensional equivalence of different programs – whether two programs behave the same with respect to mapping inputs to outputs. This aim of reasoning about programs in an implementation-independent fashion contributed to the birth of computer *science* from programming, which was attended by the independent discovery of proto-string-diagrams in the form of flowcharts.

placeholder

What makes computer programs special is that on real computers, they are specified by code. Code is just another format of data. Programs that are equivalent in their extensional behavior may have many different implementations in code: for example, there are many sorting algorithms, though all of them map the same inputs to the same outputs. Conversely, every possible program in a process theory of programs must have some implementation as code. Diagrammatically, we would summarise the situation like this: for every pair of input formats and output formats $(\mathbb{A}, \mathbb{B})$, there is a computer for that format $ev\{\mathbb{A}_{\mathbb{B}} : \mathbb{A} \otimes \Xi \to \mathbb{B}$, which takes code-format (which we will just denote Ξ going forward) as an additional input, and for every possible program $f : \mathbb{A} \to \mathbb{B}$, there exists a state $\ulcorner f \urcorner : I \to \Xi$ such that:

$$\forall \mathbb{A}, \mathbb{B} \in Ob(\mathcal{C}) \exists ev\{\mathbb{A}_{\mathbb{B}} : \mathbb{A} \otimes \Xi \to \mathbb{B} \forall f : \mathbb{A} \exists \ulcorner f \urcorner : I \to \Xi \tag{3.1}$$



(3.2)

The above equation, which characterises computers as code-evaluators, provides a plan of attack for the semantic modelling problem of entification: if we take noun-wires to be the code object in a monoidal computer, we have restricted the candidate symmetric monoidal categories to model text-circuits in a way that allows for entification.

Scholium 3.7.1. Another observation we could have made is that since computers really just manipulate code, every data format is a kind of restricted form of the same code object Ξ, but this turns out to be a mathematical consequence of the above equation (and the presence of a few other operations such as copy and compare that form a variant of frobenius algebra), demonstrated in Pavlovic's forthcoming monoidal computer book [], itself a crystallisation of three monoidal computer papers []. I would be remiss to leave out

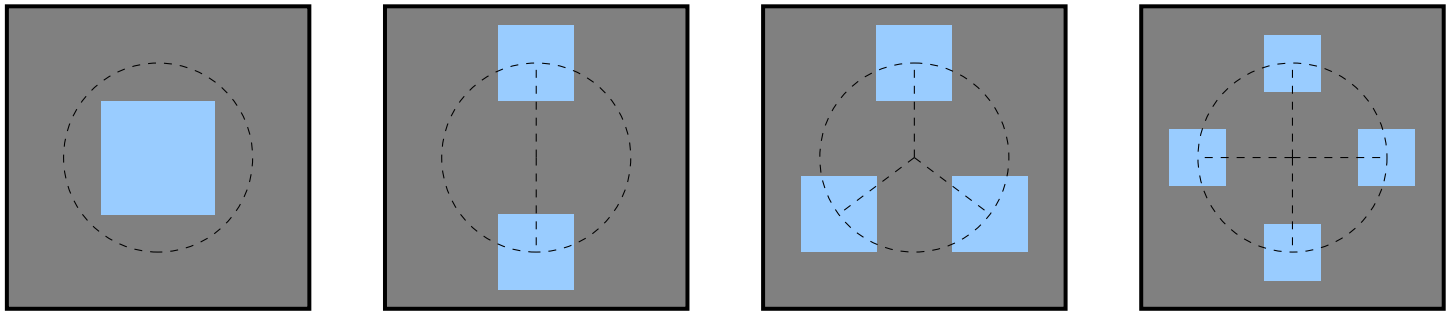
Cockett's work on Turing categories []. Both approaches to a categorical formulation of computability theory share the common starting ground of a special form of closure (monoidal closure in the case of monoidal computer and exponentiation in Turing categories) where rather than having dependent exponential types $\mathbb{A} \multimap \mathbb{B}$ or $\mathbb{B}\{\mathbb{A}$, there is a single "code-object" Ξ. They differ in the ambient setting; Pavlovic works in the generic symmetric monoidal category, and Cockett with cartesian restriction categories, which generalise partial functions. I work in Pavlovics' formalism because I prefer string diagrams to commuting diagrams.

It would be true but unhelpful to conclude that any programming language is a model for text circuits, using the code data format as the noun wire. Since semanticists like to work with sets, I provide the following construction.

Construction 3.7.2 (Sticky spiders on the open unit square model the category relations between countable sets equipped with a code object)**.** Using the open unit square with its usual topology as the code object, there is a subcategory of **ContRel** which behaves as the category of countable sets and relations equipped with universal evaluators.

Proposition 3.7.3 ($(0,1) \times (0,1)$ splits through any countable set X)**.** For any countable set X, the open unit square ▦ has a sticky spider that splits through $X^\star$.

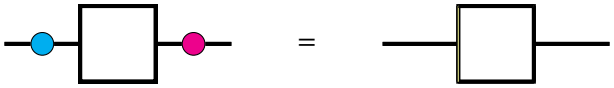
Proof. The proof is by construction. We'll assume the sticky-spiders to be mereologies, so that cores and halos agree. Then we only have to highlight the copiable open sets. Take some circle and place axis-aligned open squares evenly along them, one for each element of X. The centres of the open squares lie on the circumference of the circle, and we may shrink each square as needed to fit all of them.



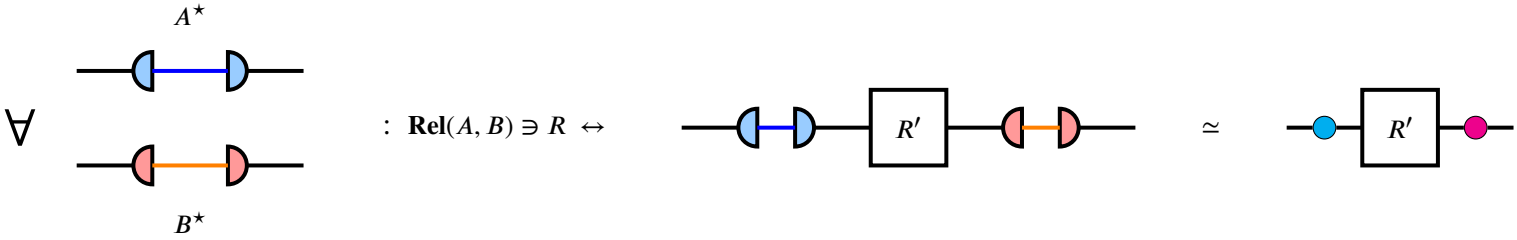
□

Definition 3.7.4 (Morphism of sticky spiders)**.** A morphism between sticky spiders is any morphism that

satisfies the following equation.

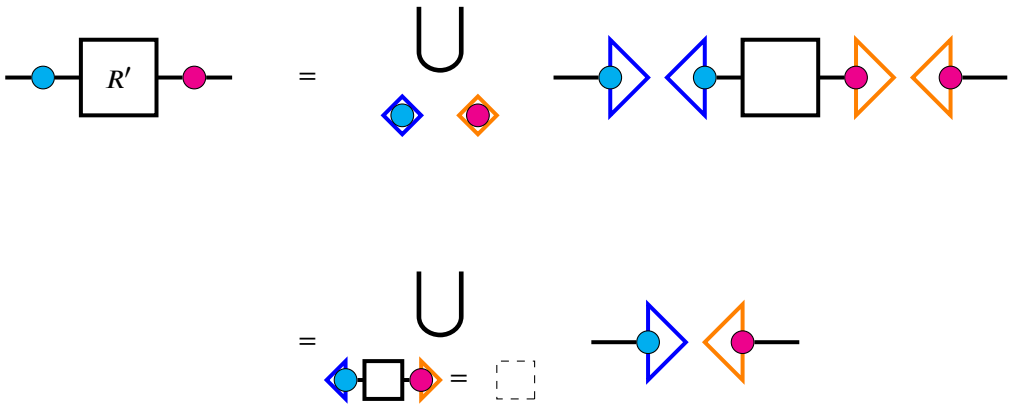


Proposition 3.7.5 (Morphisms of sticky spiders encode relations)**.** For arbitrary split idempotents through $A^\star$ and $B^\star$, the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \to B$.

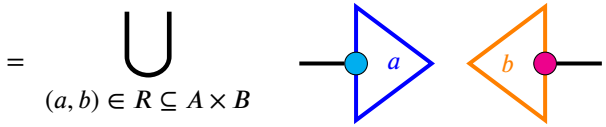


Proof.

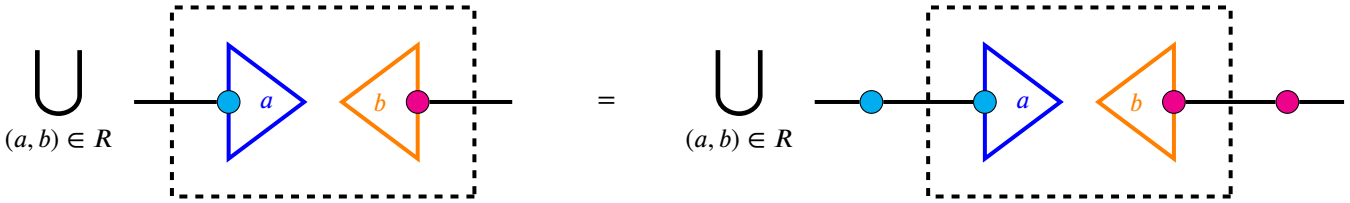
($\Leftarrow$) : Every morphism of sticky spiders corresponds to a relation between sets.



Since (co)copiables are distinct, we may uniquely reindex as:



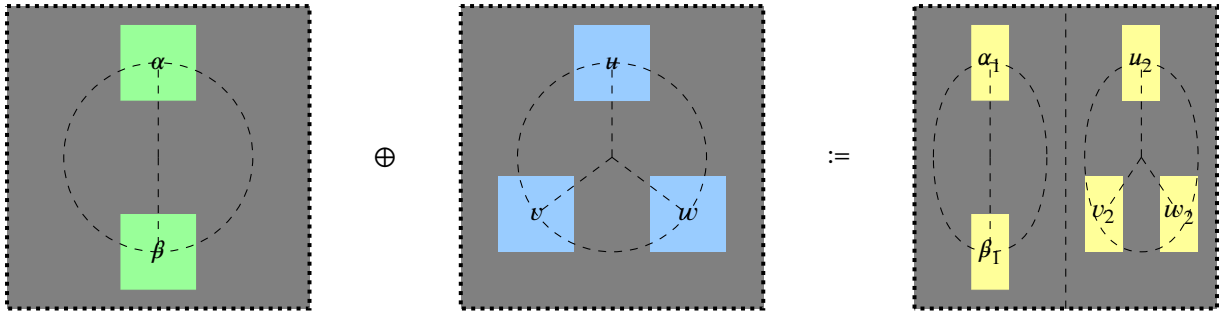
($\Rightarrow$) : By idempotence of (co)copiables, every relation $R \subseteq A \times B$ corresponds to a morphism of sticky spiders.



□

Construction 3.7.6 (Representing sets in their various guises within ▦). We can represent the direct sum of

two ▦-representations of sets as follows.

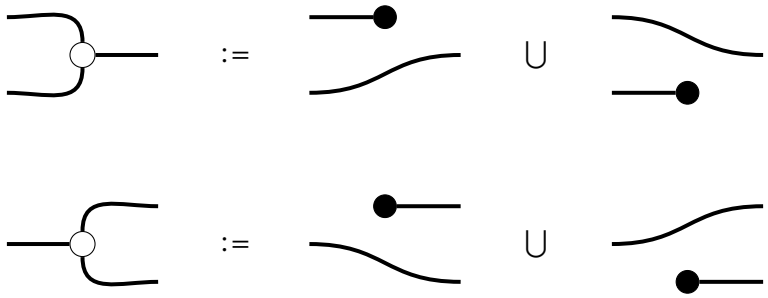


The important bit of technology is the homeomorphism that losslessly squishes the whole unit square into one half of the unit square. The decompressions are partial continuous functions, with domain restricted to the appropriate half of the unit square.

$(x, y) \mapsto (\frac{x}{2}, y)$ $(x, y) \mapsto (\frac{x+1}{2}, y)$ $(x, y)|_{x<\frac{1}{2}} \mapsto (2x, y)$ $(x, y)|_{x>\frac{1}{2}} \mapsto (2x-1, y)$

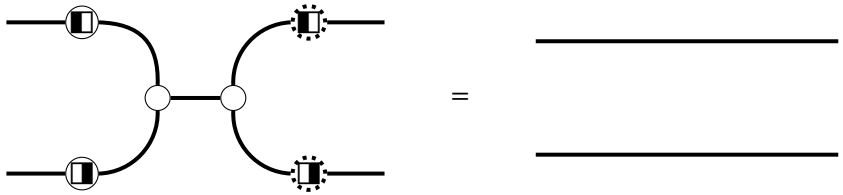
We express the ability of these relations to encode and decode the unit square in just either half by the following graphical equations.

Now, to put the two halves together and to take them apart, we introduce the following two relations. In tandem with the squishing and stretching we have defined, these will behave just as the projections and injections for the direct-sum biproduct in **Rel**.

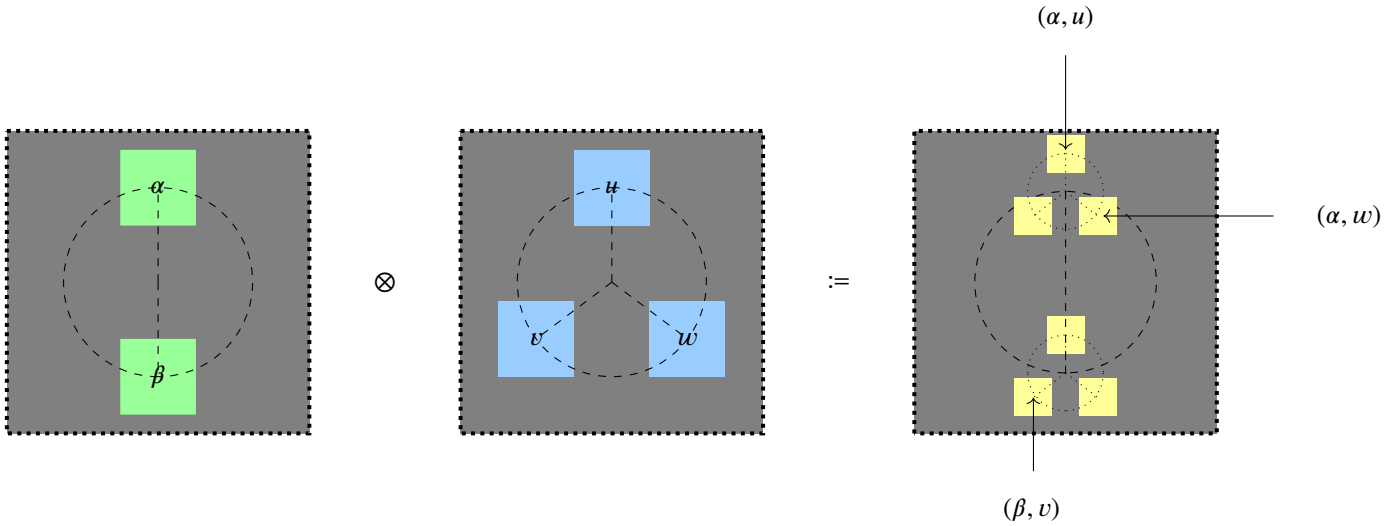


The following equation tells us that we can take any two representations in ▦, put them into a single copy of

▦, and take them out again. Banach and Tarski would approve.

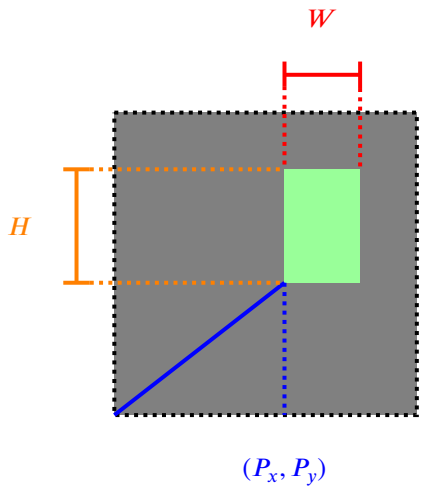


We encode the tensor product $A \otimes B$ of representations by placing copies of B in each of the open boxes of A.

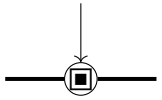


The important bit of technology here is a family of homeomorphisms of ▦ parameterised by axis-aligned open boxes. We depict the parameters outside the body of the homeomorphism for clarity. The squish is on

the left, the stretch on the right.

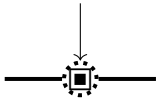


(W, H, P_x, P_y)



$(x, y) \mapsto (Wx + P_x, Hy + P_y)$

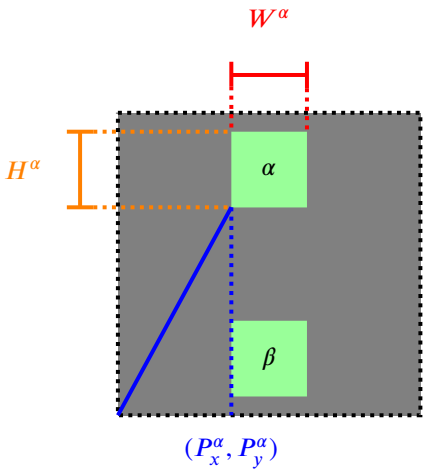
(W, H, P_x, P_y)



$(x, y)|_{\in (P_x, P_x+W)\times(P_x, P_y+H)} \mapsto (\frac{x-P_y}{W}, \frac{y-P_y}{H})$

Now, for every representation of a set in ▦ by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a squish-stretch homeomorphism via the parameters of

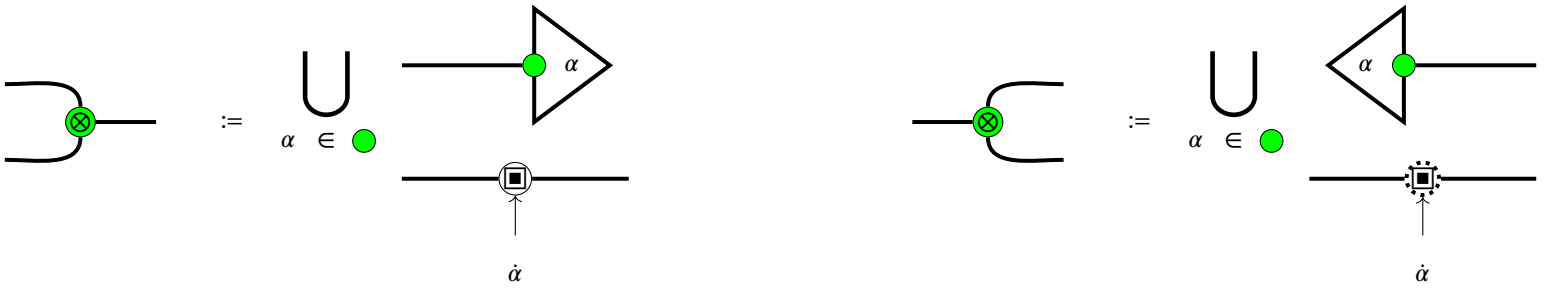
the open box, which we notate with a dot above the name of the element.

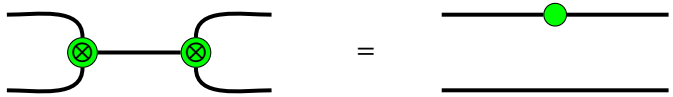


$$\dot{\alpha} = (W^\alpha, H^\alpha, P_x^\alpha, P_y^\alpha)$$

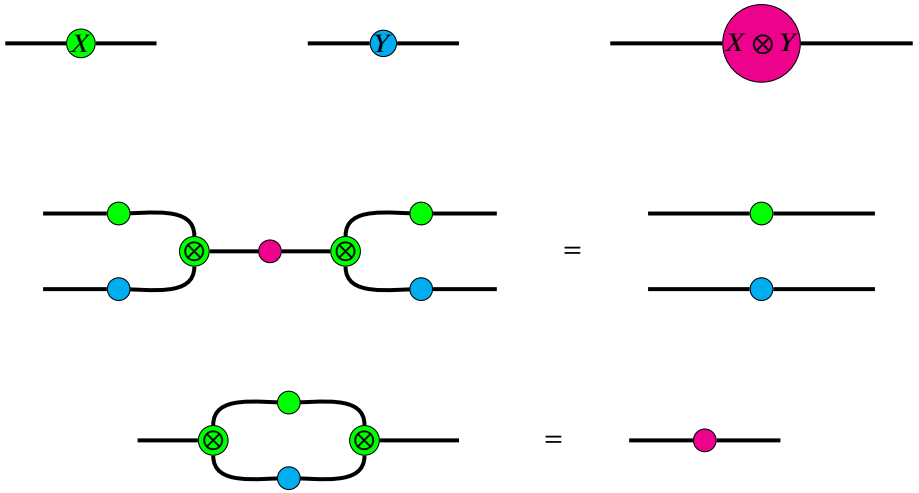
$$\dot{\beta} = (W^\beta, H^\beta, P_x^\beta, P_y^\beta)$$

Now we can define the "tensor X on the left" relation $\_ \to X \otimes \_$ and its corresponding cotensor.

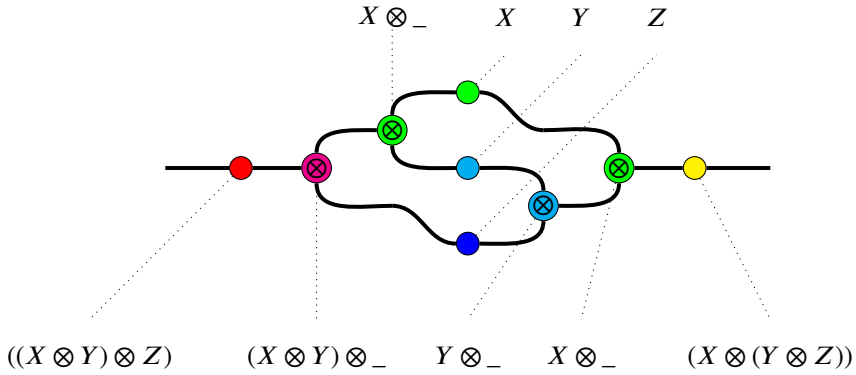




The tensor and cotensor behave as we expect from proof nets for monoidal categories.

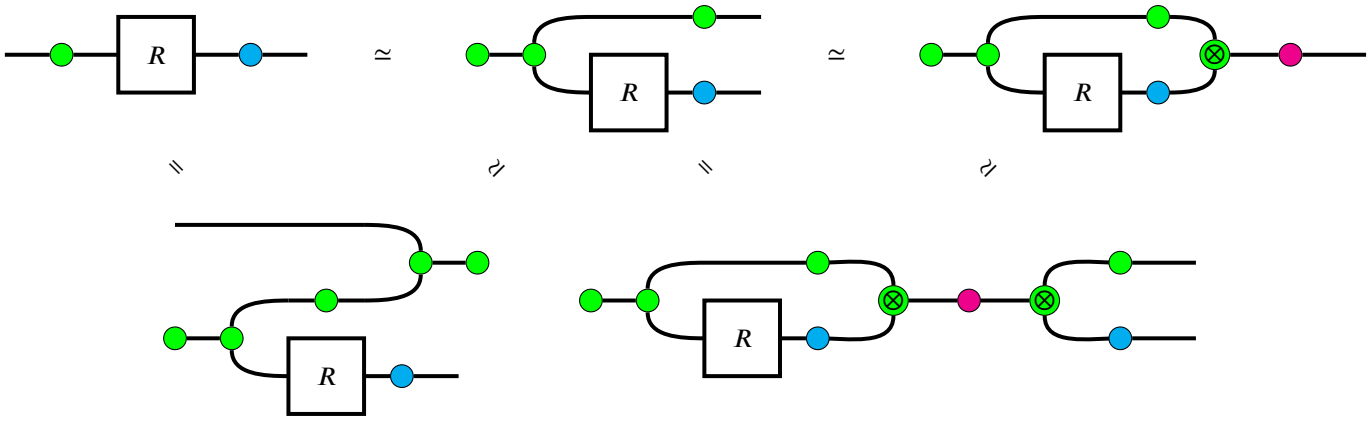


And by construction, the (co)tensors and (co)pluses interact as we expect, and they come with all the natural isomorphisms between representations we expect. For example, below we exhibit an explicit associator natural isomorphism between representations.

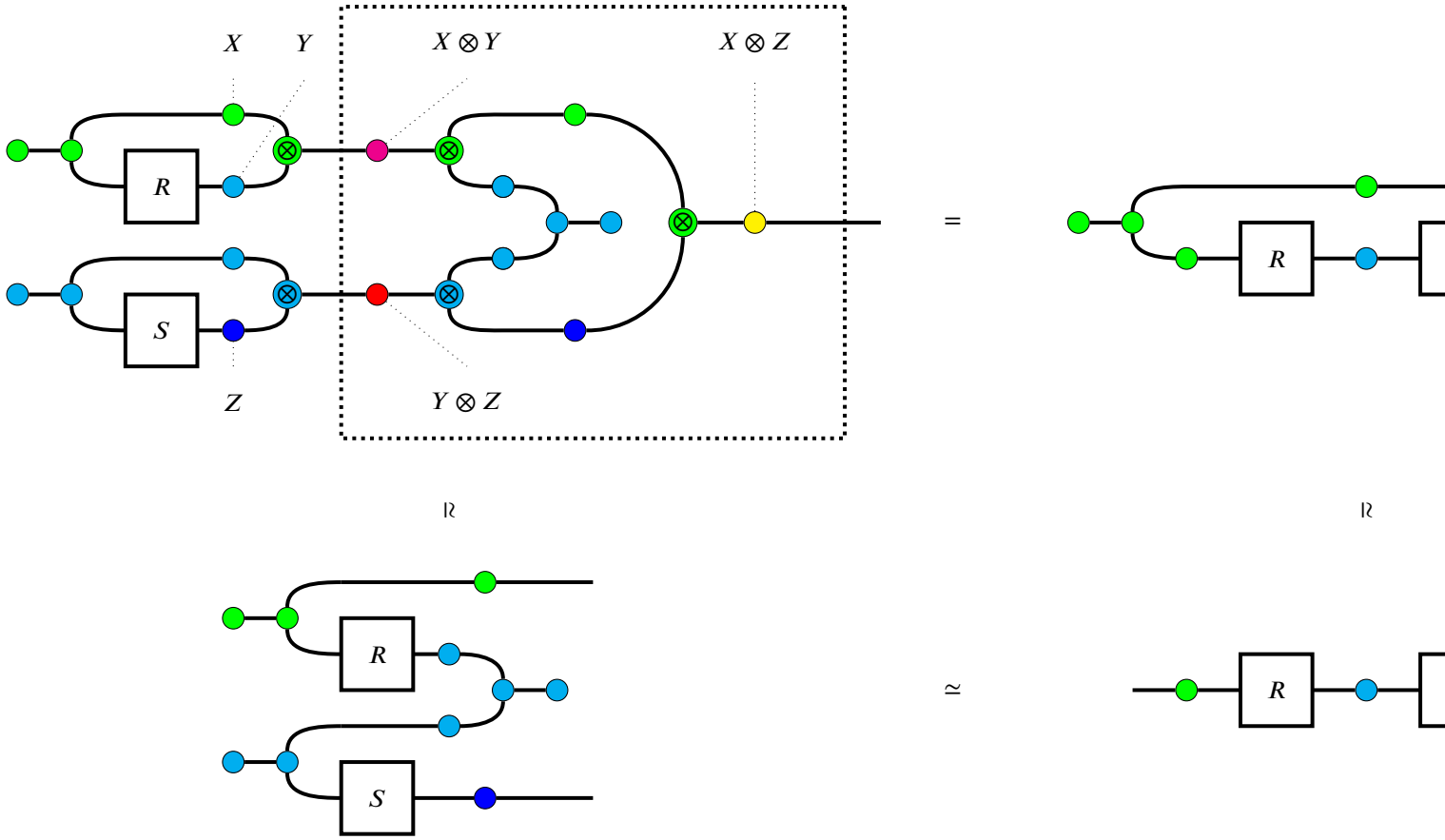


Construction 3.7.7 (Representing relations between sets and their composition within ▦)**.** With all the above, we can establish a special kind of process-state duality; relations as processes are isomorphic to states of ▦,

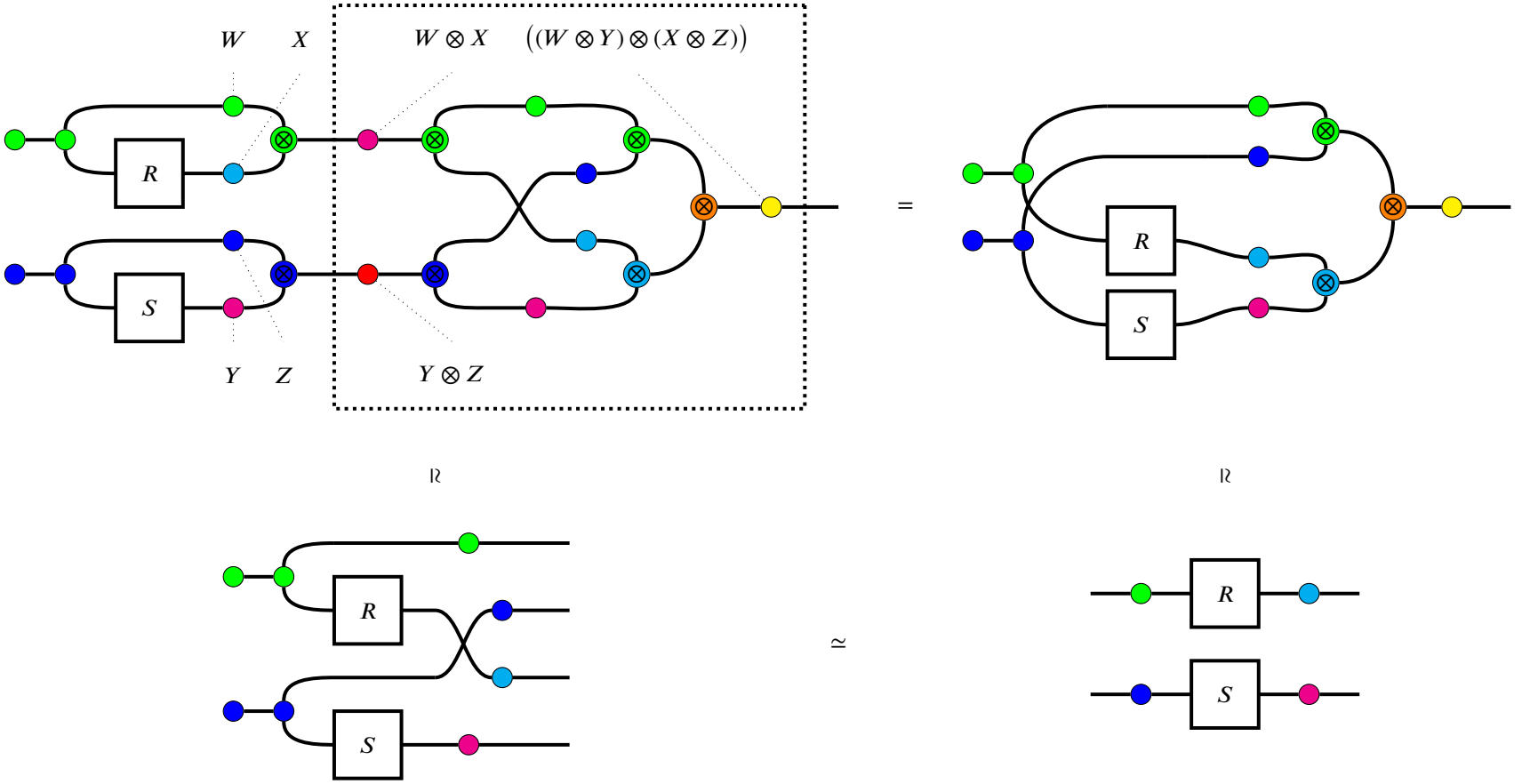
up to the representation scheme we have chosen.



Moreover, we have continuous relations that perform sequental composition of relations.



And we also know how to take the parallel composition of relations by tensors.



4
What formal linguistics could look like

I outline some potential directions of research and theses. I sketch what formal linguistics could be.

4.1 Formal models from figurative language

Figurative language is when language is used non-literally, e.g. `to bathe in another's affection`. Figurative language subsumes analogy (`built like a mountain`), metaphor (`she got a lot out of that lecture`) and some idioms (`raining cats and dogs`). The issue with figurative language for formal semantics, insofar as formal semantics is concerned with truth-conditions, is that one requires an underlying model in order to begin truth-conditional analysis. The role of figurative language, especially that of metaphor, is in some sense to provide those models in the first place[1], so the truth-theoretic (or inquisitive, or dynamic) approach to semantics operates at an inappropriate stage of abstraction. We might illustrate or depict schema to represent figurative language, but to the best of my knowledge, there is no formal account of how the systematicity of a chosen schematic corresponds to the organisation of a metaphor or concept. So what is required is a methodology to construct the underlying models from the figurative language in a more-or-less systematic way.

The whole point of mucking around with **ContRel** earlier is this: figurative language can be formally interpreted as vignettes involving topological figures. I will demonstrate here that cofunctors from **ContRel** into text-circuits representing utterances are promising candidates for the formalisation of figurative language. My focus will exclude idiomatic language and one-off analogies in favour of metaphor just because the latter is most interesting, though the methodology applies in other cases of figurative language. I will take a *metaphor* to be figurative language that utilises the systematic structure in one conceptual domain to give partial structure in another conceptual domain[2]. This may subsume some cases of what would otherwise be called *similes* or *analogies*. The differences far as I can tell between a metaphor and an analogy is the presence of systematicity in the former, and a weak requirement that the correspondence involves separate conceptual domains. It doesn't really matter for this discussion what the difference is.

First, we observe that we can model certain kinds of analogies between conceptual spaces by considering structure-preserving maps between them. For example, Planck's law gives a partial continuous function from part of the positive reals measuring tempature of a black body in Kelvin to wavelengths of light emitted, and the restriction of this mapping to the visible spectrum gives the so-called "colour temperature" framework used by colourists. It will turn out that a decategorified cofunctor has the right kind of structure.

Second, we observe that we can use simple natural language to describe conceptual spaces, instead of geometric or topological models. Back to the example of colour temperature, instead of precise values in Kelvin, we may instead speak of landmark regions that represent both temperature and colour such as `incandescent` and `daylight`, which obey both temperature-relations (e.g. `incandescent is cooler than daylight` and colour-relations (e.g. `daylight is bluer than incandescent`).

Third, we observe that we can also use simple natural language to describe more complex conceptual schemes with interacting agents, roles, objects, and abilities. This will require a cofunctor. Organising this linguistic data in the concrete structure of a text circuit allows us to formally specify what it means for one

conceptual scheme to structure another by describing structure-preserving maps between the text circuits. This will allow us construct topological models of metaphors such as `TIME is MONEY`.

Finally, I will describe a preliminary taxonomy of different kinds of metaphor, and discuss what is to be done.

4.1.1 Temperature and colour: the Planckian Locus

Example 4.1.1 (The Physicists' Planckian Locus)**.** Planck's law describes the spectral radiation intensity of an idealised incandescent black body as a function of light frequency and temperature. Integrating over light frequencies in the visible spectrum yields a function from temperature of the black body to chromaticity.

Abstractly, the Planckian Locus is a continuous function mapping the positive real line representing the conceptual domain of temperature into the plane representing the conceptual domain of colour. The Planckian locus is the basis of colourist-talk about colour schemes in terms of temperature, which allows them to coordinate movements in colourspace using the terminology of temperaturespace, e.g. `make this shot warmer`. This fits with what we would prototypically expect a metaphor to allow us to do with meanings.

However, the particular mathematical conception of metaphor-as-map in Example 4.1.1 is too rigid: it only goes one way. It is a specific and inflexible kind of metaphor that does not behave at all outside its specified boundaries. For example, colourists have to deal with offsets towards green and magenta, which are not in the chromaticity codomain of the function given by Planck's law. It would be truer to life if we further analysed the function as mediated by a strip.

Example 4.1.2 (The colourist's Planckian Locus)**.** Now we aim to extend our mathematical model to accommodate the fact that colourists deal with chromatic offsets or deviations from the mathematically precise locus given by Planck's law.

A refinement we have just captured is the partial nature of metaphor CITE . In the language of our running example, pure green is outside the scope of the colour-temperature correspondence given by the Planckian Locus, so the metaphor is only a partial structuring of the colour domain according to the temperature domain. This partiality in the colour domain means that it would have been inappropriate to model the passage of colour-talk to temperature-talk as a function from colour to temperature, as functions are total, rather than partial, on their domain. While it is conceptually nice that we are on the way to recovering monoidal cofunctors as a model of metaphor, why didn't we stay simple and just use a partial function? The answer is that the strip at the apex represents the *talk* part of colour- and temperature-talk.

Example 4.1.3 (Conceptual transfer between domains)**.** When colourists use the temperature metaphor they might say "hot", "warm", "cooler", which are not specific temperature ranges in Kelvin, but concepts in temperature-space. Recalling that we may consider concepts to be open sets of a topology (and comparatives as opens of the product), we observe that we can linguistically model regions on the positive reals with

words `little` (labelled l), `lot` (labelled L), and `more` (labelled M), an algebraic basis from which derive `less` by symmetry, and other regions such as `more than a little, less than a lot`. In this particular running example, it happens that both legs of the span of functors have a lifting property, which explains how we might model the fact that conceptual colourist-talk of "daylight" or "candlelight" in the colour domain can be sensibly interpreted in the temperature domain. The formalisation of this fact follows by symmetry from this example.

4.1.2 *Time and Money: complex conceptual structure*

Metaphor is perhaps the only methodology we have for making sense of certain abstract concepts, such as Time. For example, many languages make use of the metaphor `TIME is SPACE`, in which space-talk is used to structure time. In English, the future is ahead of us and the past behind, while conversely, for the Aymara the future is behind and the past is ahead. Orthogonally, in Mandarin the future is below and the past is above. We have already demonstrated that we have the tools to deal with conceptual transfer between static conceptual spaces viewed as topological spaces via spans of continuous maps. What is of concern to us are *dynamic* metaphors that involve a conceptual space-*time* with agents and capabilities and so on. The following discussion draws heavily from CITE .

For example, in English, we make ample use[3] of the metaphor `TIME is MONEY`. There two mathematically relevant aspects of metaphor that I want to draw attention to for this metaphor. Firstly, that the conceptual affordances money-talk is marshalled to give structure to time-talk, where there is no such structure were it not for the metaphor. Secondly, that metaphor has a partial nature, in that it is not the case that the metaphor licenses all kinds of money-talk to structure time-talk.

To establish the first point of conceptual transfer, a phrase like `Do you have time to look at this?` is completely sensible to us, but literally meaningless; even if we had an oracle to measure possession, what would we point it at to measure a person's possession of time? Even if we accept some argument that the concept of possession is innate to the human faculty, when we say `This is definitely worth your time!` or `What a waste of time.`, we are drawing upon value-talk that is properly contingent in the socially constructed sense upon the conceptual complex of money.

To establish the second point of partiality, consider that money can be stored in a bank, whereas there is no real corresponding thing in the common conceptual vocabulary which one can store time and withdraw it for later use[4]. But the partiality constraint is itself partial. For instance, one can invest money into an enterprise in the expectation of greater returns, and this is not appropriate for many domains of time-talk, but there is a metaphorical match in some specific contexts, such as text-editor-talk: `learning vim slows you down at first but it will save you time later`.

Now I'll try to demonstrate by example that cofunctors between text circuits do all of the things we have asked for. The components of text circuits serve as an algebraic basis for dynamic conceptual complexes, while the cofunctor handles partial structuring of one conceptual domain in terms of another.

Example 4.1.4 (`Vincent spends his morning writing`)**.** To begin a formal figurative interpretation via the metaphor `TIME is MONEY`, we require some model of the conceptual domain of money, as well as a topological interpretation. As a first pass, we understand that money can be exchanged for goods and services, so we will settle for a text-circuit signature for trade to serve as the conceptual domain as the apex of a cofunctor, given in Figure **??**. The elements of the topological model are given in Figure **??**. The behaviour of the fibration part of the cofunctor is detailed in Figure **??**, and that of the identity-on-objects functor in Figures **??**,

??, and **??**. The figurative model serves as a foundation from which truth-theoretical semantics can begin. In the sketched interpretation, there aren't too many interesting questions one can ask, but the purpose of this example is to point out that in principle, we can exploit the systematicity of metaphor by constructing figurative mechanical models for which interesting questions can be asked and answered truth-theoretically, as in Figure **??**.

5
Bibliography

[Bas22] Matthias Bastian. Google PaLM: Giant language AI can explain jokes, April 2022.

[BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. arXiv:2104.13478 [cs, stat].

[BK20] Emily M. Bender and Alexander Koller. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online, 2020. Association for Computational Linguistics.

[BM20] John C. Baez and Jade Master. Open Petri Nets. *Mathematical Structures in Computer Science*, 30(3):314–341, March 2020. arXiv:1808.05415 [cs, math].

[BPSZ19] Filippo Bonchi, Robin Piedeleu, Pawel Sobociński, and Fabio Zanasi. Graphical Affine Algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2019.

[BS22] Guillaume Boisseau and Paweł Sobociński. String Diagrammatic Electrical Circuit Theory. *Electronic Proceedings in Theoretical Computer Science*, 372:178–191, November 2022. arXiv:2106.07763 [cs].

[BSZ14] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. A Categorical Semantics of Signal Flow Graphs. volume CONCUR 2014 - Concurrency Theory - 25th International Conference, September 2014.

[BSZ17] Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Interacting Hopf Algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, January 2017. arXiv:1403.7048 [cs, math].

[CD11] Bob Coecke and Ross Duncan. Interacting Quantum Observables: Categorical Algebra and Diagrammatics. *New Journal of Physics*, 13(4):043016, April 2011. arXiv:0906.4725 [quant-ph].

[CG16] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016. arXiv:1603.02754 [cs].

[CGG+22] Geoffrey SH Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *Programming Languages and Systems: 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*, pages 1–28. Springer International Publishing Cham, 2022.

[Cha10] Chapman, David. Nebulosity | Meaningness, December 2010.

[Cho00] Noam Chomsky. *New Horizons in the Study of Language and Mind*. Cambridge University Press, Cambridge, 2000.

[Chu11] Kenneth Church. A Pendulum Swung Too Far. *Linguistic Issues in Language Technology*, 6, October 2011.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017.

[CND+22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways, October 2022. arXiv:2204.02311 [cs].

[Coe21] Bob Coecke. Compositionality as we see it, everywhere around us, October 2021. arXiv:2110.05327 [quant-ph].

[CSC10] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning, March 2010. arXiv:1003.4394 [cs, math].

[dav] davidad. An Open Agency Architecture for Safe Transformative AI.

[DLS+23] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality, June 2023. arXiv:2305.18654 [cs].

[FGP21] Tobias Fritz, Tomáš Gonda, and Paolo Perrone. De Finetti's Theorem in Categorical Probability. *Journal of Stochastic Analysis*, 2(4), November 2021. arXiv:2105.02639 [cs, math, stat].

[Flo14] Luciano Floridi. *The Fourth Revolution: How the Infosphere is Reshaping Human Reality*. OUP Oxford, New York ; Oxford, June 2014.

[FP88] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988. Place: Netherlands Publisher: Elsevier Science.

[Fre84] Frege, Gottlob. Selbst concrete Dinge sind nicht immer vorstellbar. Man muss die Wörter im Satze betrachten, wenn man nach ihrer Bedeutung fragt. In *Die Grundlagen der arithmetik*. 1884.

[FS19] Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 1 edition, July 2019.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat].

[Gä14] Peter Gärdenfors. *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. January 2014.

[HBK+21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset, November 2021. arXiv:2103.03874 [cs].

[Hed15] Jules Hedges. String diagrams for game theory, March 2015. arXiv:1503.06072 [cs, math].

[HH12] Suzana Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences of the United States of America*, 109 Suppl 1(Suppl 1):10661–10668, June 2012.

[HK98] Irene Heim and Angelika Kratzer. *Semantics in Generative Grammar*. Malden, MA: Blackwell, 1998.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[HS20] Nathan Haydon and Paweł Sobociński. Compositional Diagrammatic First-Order Logic. In Ahti-Veikko Pietarinen, Peter Chapman, Leonie Bosveld-de Smet, Valeria Giardino, James Corter, and Sven Linker, editors, *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science, pages 402–418, Cham, 2020. Springer International Publishing.

[JKZ19] Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal Inference by String Diagram Surgery, July 2019. arXiv:1811.08338 [cs, math].

[JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991.

[Kan19] Pentti Kanerva. Computing with High-Dimensional Vectors. *IEEE Design & Test*, 36(3):7–14, June 2019.

[Kha23] Tabarak Khan. What are tokens and how to count them?, 2023.

[KS16] Nikolaus Kriegeskorte and Katherine R. Storrs. Grid Cells for Conceptual Spaces? *Neuron*, 92(2):280–284, October 2016.

[KWM23] Philipp Koralus and Vincent Wang-Maścianica. Humans in Humans Out: On GPT Converging Toward Common Sense in both Success and Failure, March 2023. arXiv:2303.17276 [cs].

[Lan10] Saunders Mac Lane. *Categories for the Working Mathematician: 5*. Springer, New York, NY, 2nd ed. 1978. softcover reprint of the original 2nd ed. 1978 edition edition, November 2010.

[LAS21] Bastien Liétard, Mostafa Abdou, and Anders Søgaard. Do Language Models Know the Way to Rome? In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 510–517, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[LGT23] Ziming Liu, Eric Gan, and Max Tegmark. Seeing is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability, May 2023. arXiv:2305.08746 [cond-mat, q-bio].

[LT23] Robin Lorenz and Sean Tull. Causal models in string diagrams, April 2023. arXiv:2304.07638 [cs, math].

[Mac63] Saunders MacLane. Natural Associativity and Commutativity. *Rice Institute Pamphlet - Rice University Studies*, 49(4), October 1963. Accepted: 2011-11-08T19:13:47Z Publisher: Rice University.

[Mar77] D. Marr. Artificial intelligence—A personal view. *Artificial Intelligence*, 9(1):37–48, August 1977.

[MN21] Marjorie McShane and Sergei Nirenburg. *Linguistics for the Age of AI*. March 2021.

[MP19] Francis Mollica and Steven T. Piantadosi. Humans store about 1.5 megabytes of information during language acquisition. *Royal Society Open Science*, 6(3):181393, March 2019.

[NC22] Sharan Narang and Aakanksha Chowdhery. Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance, 2022.

[noa22] Riley Goodside (@goodside) / Twitter, December 2022.

[Ope22] OpenAI. ChatGPT: Optimizing Language Models for Dialogue, November 2022.

[PWS+23] Boldizsár Poór, Quanlong Wang, Razin A. Shaikh, Lia Yeh, Richie Yeung, and Bob Coecke. Completeness for arbitrary finite dimensions of ZXW-calculus, a unifying calculus, April 2023. arXiv:2302.12135 [quant-ph].

[RM87] David E. Rumelhart and James L. McClelland. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MIT Press, 1987. Conference Name: Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.

[Sea80] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–424, September 1980. Publisher: Cambridge University Press.

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. volume 813, pages 289–355. 2010. arXiv:0908.3347 [math].

[Sob15] Paweł Sobociński. Graphical Linear Algebra, June 2015.

[Sut19] Richard Sutton. The Bitter Lesson, 2019.

[Søg23] Anders Søgaard. Grounding the Vector Space of an Octopus: Word Meaning from Raw Text. *Minds and Machines*, 33(1):33–54, March 2023.

[ted22] teddy [@teddynpc]. I made ChatGPT take a full SAT test. Here's how it did: https://t.co/734sPFU3HY, December 2022.

[TGZ+23] Taori, Rohan, Gulrajani, Ishaan, Zhang, Tianyi, Dubois, Yann, Li, Xuechen, Guestrin, Carlos, Liang, Percy, and Hashimoto, Tatsunori B. Stanford CRFM, 2023.

[Tho22] Alan D. Thompson. GPT-3.5 IQ testing using Raven's Progressive Matrices, 2022.

[Tom22] Tom Goldstein [@tomgoldsteincs]. Training PaLM takes 3.2 million kilowatt hours of power. If you powered TPUs by riding a bicycle, and you pedaled hard (nearly 400 watts), it would take you 1000 years to train PaLM, not including bathroom breaks. In that time, you'd make 320 trips around the globe!, July 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. arXiv:1706.03762 [cs].

[Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc, Champaign, IL, illustrated edition edition, August 2002.

[WWS+23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, January 2023. arXiv:2201.11903 [cs].

[YK21] Richie Yeung and Dimitri Kartsaklis. A CCG-Based Version of the DisCoCat Framework, May 2021. arXiv:2105.07720 [cs, math].