VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

(Acknowledgements will go in a margin note here.)

0

Synopsis and concluding discussion

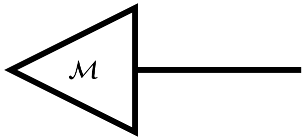


Figure 1: Let's say that the meaning of text is how it updates a model. So we start with some model of the way things are, modelled as data on a wire.

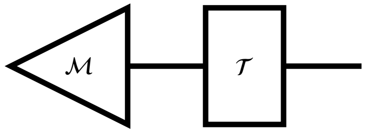


Figure 2: Text updates that model; like a gate updates the data on a wire.

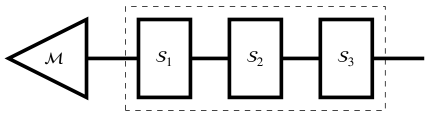


Figure 3: Text is made of sentences; like a circuit is made of gates and wires.

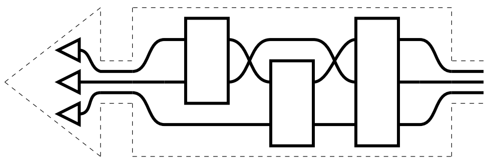


Figure 4: Let's say that ***The meaning of a sentence is how it updates the meanings of its parts.*** As a first approximation, let's say that the *parts* of a sentence are the nouns it contains or refers to. Noun data is carried by wires. Collections of nouns are related by gates, which play the roles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

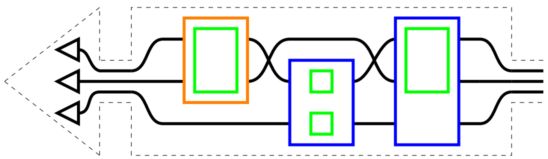


Figure 5: Gates can be related by higher order gates, which play the roles of adverbs, adpositions, and conjunctions; anything that modifies the data of first order gates like verbs.

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 [50, 6, 5], first order logic [24], causal networks [39, 29], signal flow graphs [7], electrical circuits [4], game theory [25], petri nets [1], probability theory [21], machine learning [18], and quantum theory [15, 16, 46], 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 pictorial syntax for interacting processes. They are 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 each process without having to spell out a bottom-up implementation, by asking processes to interact with other processes in certain ways. The mathematical foundation of string diagrams – applied category theory – is in short, the mathematics of compositionality.

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.

I will demonstrate how they are used in Section 1.1.5 and define them formally in Section **??**. String diagrams are a heuristically natural yet mathematically formal syntax for representing complex, composite systems. I say *mathematically formal* to emphasise that string diagrams are not merely heuristic tools backed by

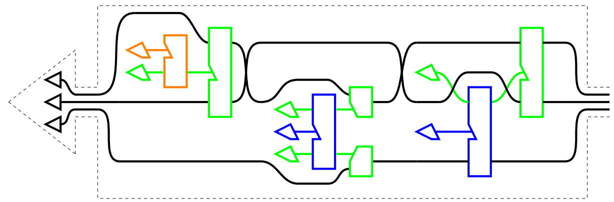


Figure 6: In practice, higher order gates may be implemented as gates that modify parameters of other gates. Grammar, and *function words* – words that operate on meanings – are in principle absorbed by the geometry of the diagram. These diagrams are natural vehicles for *dynamic semantics*, broadly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

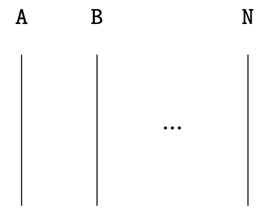


Figure 7: Nouns are represented by wires, each 'distinct' noun having its own wire.

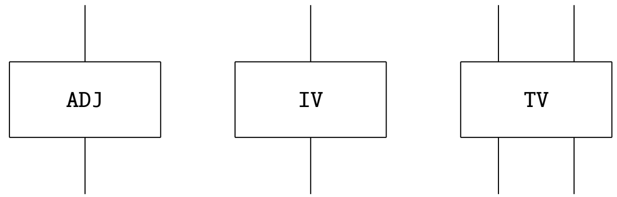


Figure 8: We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires. Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

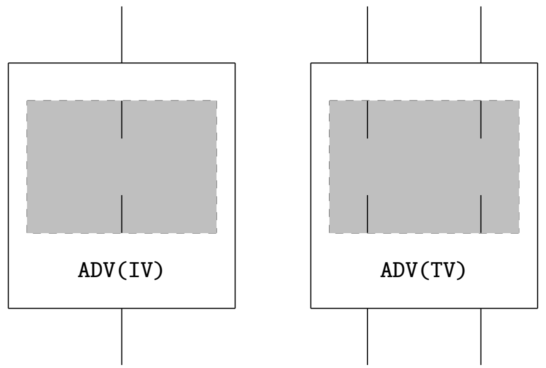


Figure 9: Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicating the shape of gate expected, and these should match the input- and output-wires of the box with the whole.

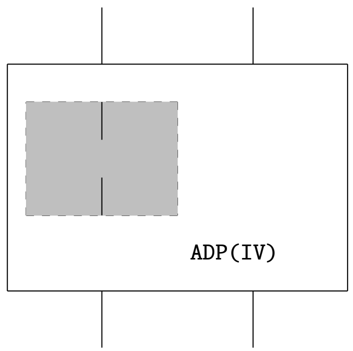


Figure 10: Similarly, adpositions also modify verbs, by moreover adding another noun-wire to the right.

a handbook of standards decided by committee: they are unambiguous mathematical objects that you can bet your life on [30**?** , 40, 35, 49].

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

Large language models 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. Although this thesis is pure theory, I wish to address the question of practical value early because I imagine practical people are impatient.

Let me outline the terms and stakes. Because the field is developing so quickly, assume that everything about LLMs is prefaced with "at the time of writing". Large Language Models are programs trained – using a lot of data and a lot of compute time – to predict the next word in text, computational techniques for which have evolved from Markov n-grams to transformers [57]. This sounds unimpressive, but – in tandem with fine-tuning from human feedback in the case of chatGPT [45] – it is enough to tell and explain jokes [2], pass the SAT [54] and score within human ranges on IQ tests [55]. There is an aspect of genuine scientific and historical surprise that text-prediction can do this kind of magic. On the account of [42], 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 [13] expressed concern that all this fruit would be picked bare in a generation to force a return to knowledge-based methods, but those concerns appear now to be 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 *explainability and safety*. These models have a tendency to hallucinate facts and (ironically, for a computer) bad arithmetic [26], and 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 [19], 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 [51]. 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.

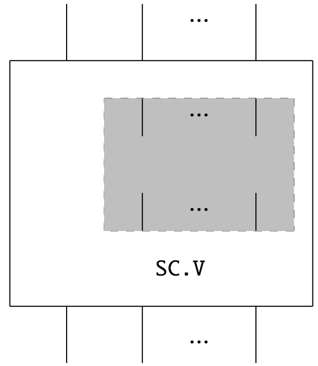


Figure 11: For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit.

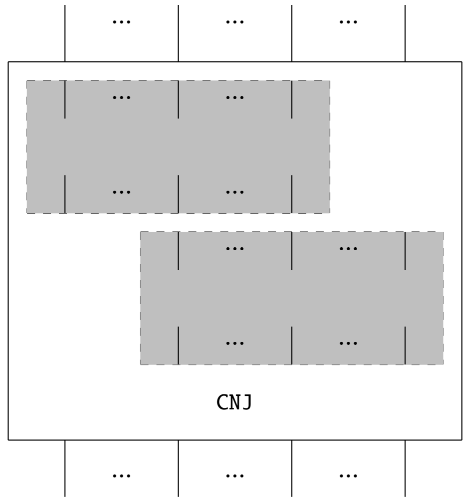


Figure 12: Conjunctions are boxes that take two circuits which might share labels on some wires.

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 am unfortunately placed in that I feel enough affinity to have defensive instincts for each camp, but I am distanced enough from each that I am sure to suffer 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.

0.3 *First Reply: I don't know. Maybe explainability, maybe something else.*

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'.

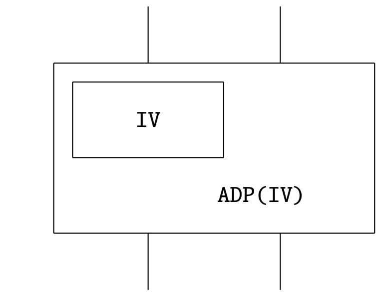


Figure 13: Of course filled up boxes are just gates.

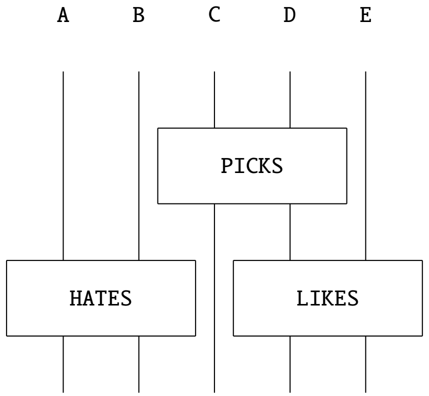


Figure 14: Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits.

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 [11] that language is an innate human faculty, the 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 [43], 90 billion neurons [27], 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 [12, 44] – 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 [32] and 5 bits per character. The architecture has 540 billion neurons, and required 3.2 million kilowatt hours of energy for training [56]. 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 [53]. 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 to feast on.

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*. Systematicity CITE refers to when a system can (generate/process) infinitely many (inputs/outputs/expressions) using finite (means/rules/pieces) in a "consistent" (or "systematic") manner – some tautologies require geniuses like

Fodor to put into words. Like pornography, examples are easier than definitions. We know finitely many words but we can produce and understand 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.

The concepts of systematicity and compositionality are deeply linked, because the only way we know how to achieve systematicity in practice is composition. Frege's initial conception of compositionality [20] was borne of meditations on language, and states that a whole is the sum of its parts – some tautologies require geniuses like Frege to put into words. Later conceptions of compositionality, the most notable deviation arising from meditations on quantum theory, are the same as the original, modulo variations on the formal definitions of parts and the method of summation [14]. So there is our starting point: 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 problem of the stimulus, if not most of it.

The reasoning I report above should clarify why some folks don't think large language models have anything to do with language. The 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 even if they do appear to be doing compositional things, 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, I hope the framework I present in this thesis can be a proposal to split the cake sensibly between the two halves of the problem: meanings for the machines, compositionality for the commons. Syntax is still difficult and vast, but the rules are finite and relatively static. We can break the black-box by reexpressing syntax as the composition of smaller black-boxes. 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 can have confidence that the internal representations of the machine – their "mind's eye" – contains something we can probe and understand.

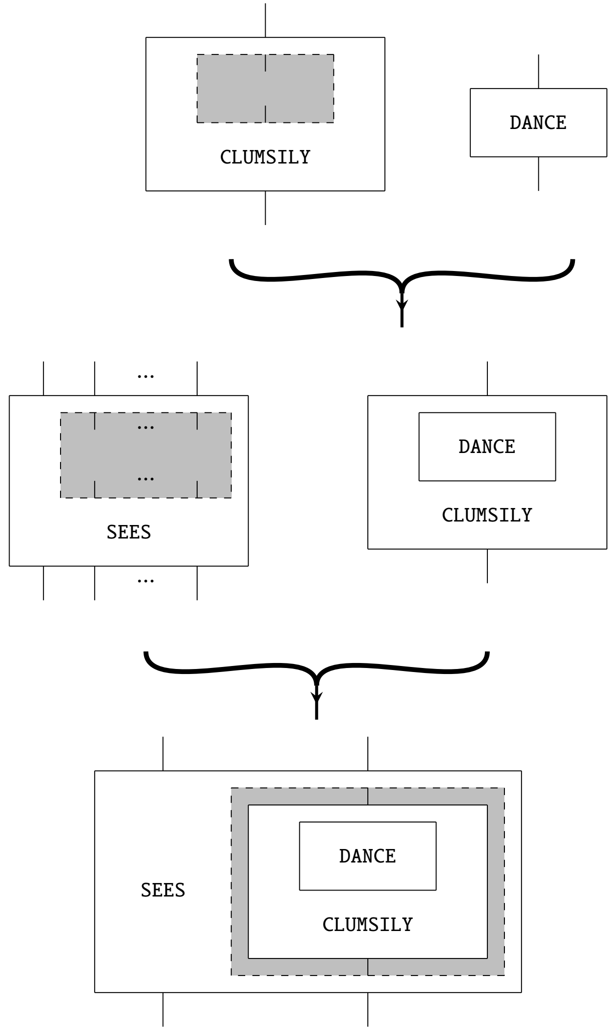


Figure 15: To summarise: composition by nesting corresponds to grammatical structure within sentences. Sentences correspond to filled gates, boxes with fixed arity correspond to first-order modifiers such as adverbs and adpositions, and boxes with variable arity correspond to sentential-level modifiers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

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 – and we see this historically, tracking the evolutionary path for data-driven language models from markov chains to deep learning [36], RNNs [47], LSTMs [28], and now transformers [57] – 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. These structural changes are motivated by understand-

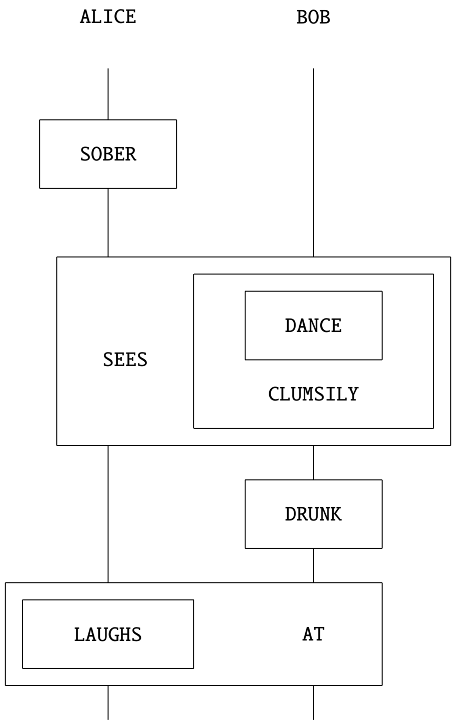


Figure 16: Composition by connecting wires corresponds to identifying coreferences in discourse. We obtain the same circuit for multiple text presentations of the same content, e.g. `Sober Alice who sees drunk Bob clumsily dance laughs at him.` yields the same circuit as the text `Alice is sober. She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

ings (at varying degrees of formality) of the "geometry of the problem" [8]. 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*. I hope to demonstrate in Section **??** how 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.

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 language. First, many linguistic phenomena are nebulous [9]: 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 even if we do have crisp mathematical models for all the constituent processes we are still left with something computationally irreducible. These two points together weakly characterise the kinds of problem domains where machine learning shines. It is just fact that LLM outputs today conform to any reasonable understanding of syntax, semantics, pragmatics, conversational implicature, and whatever else we have theorised. It is just a fact that they produce better poetry and humor than anything we could explicitly program according to our current understanding. It is almost a natural law that they will only get better from here.

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 **Point of Information:** *What is computational irreducibility?*

Computational irreducibility [59], elsewhere called "type 2" problems [41], refers to a special kind of computational difficulty where the explanation of a system amounts to a total computational simulation of it. This is best understood by example. We have a closed-form mathematical expression to shortcut the computation of the evolution of a system of two point masses under gravity, but we have no such shortcut for the three-body problem; the best we can do is simulate the system's evolution, and an inconvenient fact is that even for very simple systems, it is possible that no amount of causal-mechanistic understanding will simplify compututational simulation.

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

It's not meant to. The core value proposition for synthesis is explainable AI, which operates in a manner we can analyse, and if appropriate, constrain. For this purpose, merely knowing *what* a deep-learning model is thinking is not enough: solving 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. Moreover, purely data-driven methods to control the computation may incur ethical costs CITE , to say nothing of the potential harm that may result from a poorly safeguarded model CITE . Add to this the ever-growing dirty laundry lists of AI models failing CITE in inhuman ways, and the task of incorporating compositionality – a formal understanding of *how* models learn and reason – gains urgency.

I recount the following from [52], 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 [3], the performance of current LLMs is a form of Chinese Room [48] 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 [38]. Since we know that brain activity patterns encode abstract conceptual space with the same mechanisms as they do physical spaces [34], 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, at least for individual tokens, and so long as we take seriously some version of Gärdenfors' [23] thesis that meaning is encoded geometrically.

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 [58] and by probing their behaviour with respect to understood symbolic models [33]. 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 [31], 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 [37], or architectures whose behaviour arises from compositional structure using neural nets as constituent parts, such as GANs [22] and gradient boosted decision trees [10]. The work in this thesis builds upon a research programme – DisCoCat [17], elaborated in Section **??** – 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.

A deeper objection here is that diagrams do not look like serious mathematics. Later I will give ample space to show how they are serious, but 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, the oral tradition and living history of mathematics in institutions of learning in France 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, favouring abstraction and disdaining visualisation, in line with their contemporary artistic and musical fashions. Consequently, Bourbaki's Definition-Proposition-Theorem style of mathematical exposition is a historical aberration: a bastardisation of Euclid that eschews intuition via illustration and specificity in favour of abstraction and generality, pretending at timelessness, 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 not lend itself well to theorems, but is a happy muse for mathematical play.

0.3.5 ***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. Throughout this thesis I will be re-presenting familiar and unfamiliar things in string diagrams, so I request the reader to remember the koan and keep an open mind.

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 the possible solution: by composing architectures in the shape of language from the start, we can attempt guarantees that the latent-space representation of the machine is built up in the same way we build up a mental representation when we read a book or watch a film. I'll sketch how to approach this in Section **??** .

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 cannot help: to borrow a thought from CITE , suppose you knew the insides of a mechanical calculator by heart. Does that mean you *understand* arithmetic? At best, obliquely: implementing a computer for ideal arithmetic means compromises; the calculator is full of inessentialities and tricks against the constraints of physics. You would not know where the tricks begin and the essence ends. Similarly, suppose you knew every line of code 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 the value proposition to establish is how string diagrams come into the picture for the linguist who is (definitionally) concerned with understanding how language works. Let's entertain one more objection from the practical reader and one objection from the theoretical reader before formulating a reply.

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

Whether LLMs are even a theory of language is a best debatable. 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 CITE , 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 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 CITE . 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

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 "more correct" CITE . 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 hobby. Third, while explainability studies in terms of analysing the behaviour of subnetworks is practically important, the whole approach seems fundamentally misguided; imagine an epicyclist explaining the precession of mercury's perihelion by pointing at a collection of epicycles and calling it a "distributed representation".

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 CITE , or their weakness to adversarial prompt-injections CITE , but these weaknesses do not shed light on their linguistic competence and "understanding" directly.

Now the disappointments. LLMs are far from simple, and simplicity (Occam's Razor) is an ancient criterion for the goodness of explanation. Moreover, they fail at providing explanatory mechanisms CITE , and they do not unify or subsume our prior understandings CITE . The first two points are 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. To be fair, 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. There is something deeply wrong about our understanding – if we can call it that – of level 2. We have working understandings of several aspects about this level. We know something about the nature of internal representations in neural nets both in terms of semantic encoding within weight distributions CITE and of token-embeddings in latent space CITE . We can explain how transformer models work in terms of attention mechanisms and lookback CITE , which serves as working understanding of the procedural aspect of LLMs. We also understand mathematically how it is that these models are trained using data to produce the outputs they do. The deep problem is that in spite of these understandings which should jointly cover all of level 2, we only obtain explanations at the wrong level of abstraction for the purposes we care about CITE . Level 2 is in a sense the important level to get right for the purposes of explainability, auditability, and extension, since it is at the level of representation and procedure that we can investigate internal structure and match levels of abstraction across domains. Since the three levels interact, the challenge is to slot in a story about level 2 that coheres with what we already know about levels 1 and 3. I claim that we can go about this challenge using string diagrams as a lingua franca for mathematical linguists and machines.

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

There is nothing wrong with punchcard machines either, as far as computability is concerned. String diagrams, and applied category theory more broadly, are a good metalanguage for formal linguistics. The usual choice of set theory is not well-suited for complex and interacting moving parts. The chief drawback is that set theory requires 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[1]. As you may already know, this representation-dependency is a bureaucratic nightmare when dealing with a complex system. This leads to at least three problems.

1. The sociological problem is that this makes things difficult to understand unless you have invested a lot of time into mathematics in general.

2. Interoperability is tricky. When a programmer wants to use a data structure or algorithm, they do not always write it from scratch or copy code from stackoverflow or get their LLM to do it for them; they may use a library that provides them structures and methods they can call without worrying about how those structures and methods are implemented all the way down. However, if you building a complex theory by spelling out implementations set-theoretically from the start, incorporating a new module from elsewhere becomes difficult if that module has encoded things in sets differently. A lot of busywork goes into translating foundations of formalisms at an analogous level to machine code, which is time better spent building upwards and outwards. A computer scientist might say that some abstraction is needed, and being one, I say so.

3. Third, and related to the second, is that set-theory is not the native language for the vast majority of practical computation. Often in the design of complex theories, we do not care about how precisely representations are implemented, instead we only care about placing constraints or guarantees on the behaviour of interactions – that is, we care about operational semantics.

The problems I have mentioned above are obstacles, and I hope to show that using applied category theory as a metalanguage may be a solution. A broad theme of this thesis is to illustrate the economy and reach of applied category theory for dealing with compositional phenomena. 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? The discipline 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. I show

[1] 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. What we really want of the product is the property that $(a, b) = (c, d)$ just when $a = c$ and $b = d$. Now 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:

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

Novel contributions:

- Section **??** demonstrates how string-diagrammatic reasoning allows for graphical proofs of strong equivalences between typelogical, string-production, and further strong equivalence to a fragment of tree-adjoining grammars.
- Text diagrams and text circuits lie at the heart of the above correspondences and of this thesis, which we introduce and investigate in Section **??** in an abridged re-presentation of CITE , culminating in a proof relating the expressive capacity of text circuits to a controlled fragment of English that serves as evidence that text circuits are a natural metalanguage for grammatical relationships that make no extraneous distinctions.
- In Section **??**, moving towards applications, I introduce the category of continuous relations, to set a mathematical stage upon which we can build toy models, expanding upon my previous work on linguistically compositional spatial relations CITE towards modelling mechanical systems and containers.
- I mathematically investigate the possibilities and limitations of textual modelling with text circuits on classical and quantum computers in Section **??** by examining the limitations of cartesian monoidal categories for modelling text circuits, taking the universal approximation theorem into account.
- In Section **??**, I extend the string-diagrammatic techniques used to prove correspondences between different syntactic theories to text circuits provides a framework for the formal, conceptually-compliant modelling of textual metaphor.
- I demonstrate a formal connection between tame topologies and tensed language in Section **??**, which extends to a formal framework to model narratives as database rewrites in Section **??**.

that we can progress towards this aim using formal diagrams that our visual cortexes have built-in rules to manipulate, and that allow us to work at the level of abstraction we choose, so that we may easily incorporate other modules and find implementations in a variety of settings.

To summarise the first value proposition, 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. In this vein, theories of syntax expressed in terms of string diagrams makes it easier to reason about expressive equivalence between theories at a compositional level. More precisely, a theory of syntax is expressed as a finitely presented symmetric monoidal category, and relationships between theories are expressed as symmetric monoidal functors, which are generalised structure-preserving maps. The upshot of reasoning in this way is that equivalences are established at a structural level between the atomic components of corresponding theories, which lets us push the boundaries of what may be considered formal semantics in Section **??** .

TL;DR of introduction: LLMs do not explain language, and formal linguists "explain" language using the mathematical equivalent of punchcard machines. Both sides stand to gain from synthesis, but synthesis requires a shared mathematical metalanguage. I propose string diagrams and applied category theory as a candidate, and the rest of the thesis is about justifying the proposal.

0.5 Synopsis of the thesis

Chapter 2 provides the relevant background and foundations for category theory, machine learning, and formal syntax for this thesis, which lives at the intersection. The ideas required from the parent fields will be basic, so the exposition is meant to get readers across disciplines on the same page, not impress experts. For string diagrams I will first provide a primer for how process-theoretic reasoning with string diagrams work, by example. On the category theoretic end, I will recount symmetric monoidal categories as the mathematical objects that string diagrams are syntax for, as well as provide a working understanding of PROPs CITE and n-categories as formalised by CITE , which provide a metalanguage for specifying families of string diagrams. Once the reader is happy with string diagrams, for machine learning I will just introduce how deep neural nets and backpropagation work in string-diagrammatic terms to provide a foundation of formal understanding, and I will explain the mathematical and real-world reasons why deep learning is so powerful. For formal linguistics, I will sketch out a partial history of categorial linguistics in general, along the way briefly recasting CITE in more modern mathematical terminology to justify string diagrams as a generalisation of Montague's original conception of syntax and semantics.

Chapter 3 is about string diagrams for formal syntax. Here I recount context-free, pregroup, and tree-adjoining grammars to the reader, recast them string-diagrammatically, and relate them by means of discrete monoidal fibrations, a piece of mathematics I will develop. Then we (re)introduce text circuits as the common structure between those different theories of grammar that abstracts away differences in linear syntactic presentation while conserving a core set of grammatical relations. During my DPhil, I wrote a paper CITE in collaboration with Jonathon Liu and Bob Coecke which introduced text circuits in a pedestrian way, and characterised their expressive capacity with respect to a controlled fragment of English. I will just recover the main beats of that paper, this time using the metalanguage of n-categories.

Chapter 4 sets a mathematical stage for us to model and calculate using text circuits, for which purpose I introduce the category of continuous relations **ContRel**, a naïve generalisation of the category of continuous maps between topological spaces. **ContRel** is new, in the sense that the category-theoretically obvious approaches to obtaining such a category either do not work or yield something different. This section culminates in formal semantics for topological concepts such as `inside` which underpin the kinds of schematic doodle cartoons we might draw on paper to illustrate events occurring in space.

Chapter 5 is a formal invitation to playtime for the reader who gets that far. I don't expect that I've explored any novel linguistic phenomena, and I don't think I've invented any substantially new mathematics. All I've done is a form of intellectual arbitrage, putting tools from one field to work in another. To properly give weight to my claim that string diagrams and category theory are a good metalanguage for linguistics as a whole, it is necessary to demonstrate breadth. So, I model linguistic topological concepts; I give a mathematical setting for the study of generalised anaphora that reference any meaningful part of text; I provide formal semantics for the container metaphor in particular and textual metaphors in general; I sketch a formal correspondence between tensed language and tame topologies that extends to formally reckoning with narrative structure. All of this is to show that the methods I use are flexible and not doctrinal. I am not interested in whether these topics have been mathematicised more thoroughly and deeply before; what I care to demonstrate is that a little category theory and some imagination can go a long way.

Finally, I close with a discussion and prospectus. For the convenience of the reader, bibliographies are placed at the end of each chapter. Corrections, comments, and suggestions are welcome at `vincentwangsemailaddress@gmail.com`. I hope you enjoy the read, or if nothing else, I hope you like my diagrams!

1

Background

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 1.1.1 (Category)**.** A *category* $\mathcal{C}$ consists of the following data

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

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

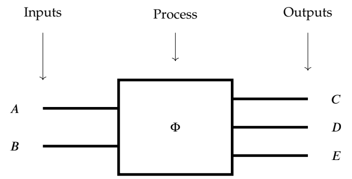
Unitality: For any morphism $f : a \to b$, $1_a; f = f = f; 1_b$

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

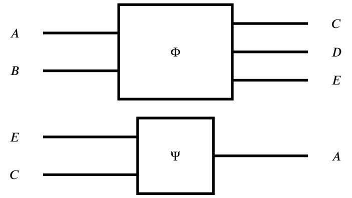
1.1 Process Theories

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.

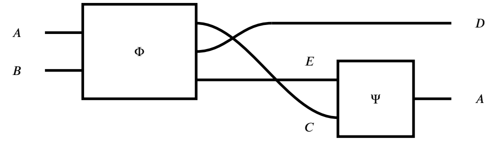
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. Unless otherwise specified, we read processes from left to right.



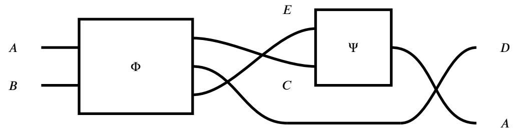
Processes may compose in parallel, depicted as placing boxes next to each other.



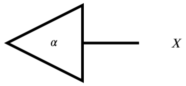
Processes may compose sequentially, depicted as connecting wires of the same type.



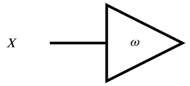
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



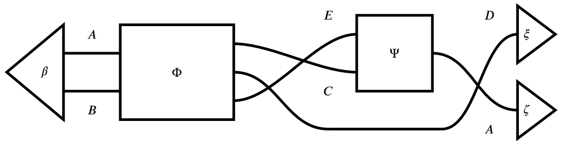
Some processes have no inputs; we call these *states*.



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 1.1.13 (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 1.1.2 (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.

$$\begin{tikzcd} & x \arrow[dl, "f"'] \arrow[d, dashed, "\langle f,g\rangle"] \arrow[dr, "g"] & \\ a & a \times b \arrow[l, "\pi_0"] \arrow[r, "\pi_1"'] & b \end{tikzcd}$$

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:

$$\begin{tikzcd}
& A \overset{K}{\times} B \arrow[dl, "\overset{K}{\pi_0}"'] \arrow[dr, "\overset{K}{\pi_1}"] \arrow[ddl, dashed, bend right] & \\
A & & B \\
A \overset{W}{\times} B \arrow[u, "\overset{W}{\pi_0}"] \arrow[urr, "\overset{W}{\pi_1}"] \arrow[uur, dashed, bend right] \arrow[rr, dashed, "id", leftrightarrow] & & A \overset{W}{\times} B \arrow[ull, "\overset{W}{\pi_0}"'] \arrow[u, "\overset{W}{\pi_1}"']
\end{tikzcd}$$

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 1.1.14 (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 1.1.15 (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:

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

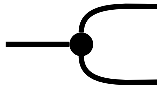
Parallel composition of systems is the cartesian product of sets. The parallel composition of relations $A \otimes C \overset{\Phi \otimes \Psi}{\nrightarrow} B \otimes D$ of relations $A \overset{\Phi}{\nrightarrow} B$ and $C \overset{\Psi}{\nrightarrow} D$ is defined:

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

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.

1.1.1 *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 1.1.16 (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 1.1.17 (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 1.1.18 (Sets and relations)**.** Consider a set A. The copy relation is defined:

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

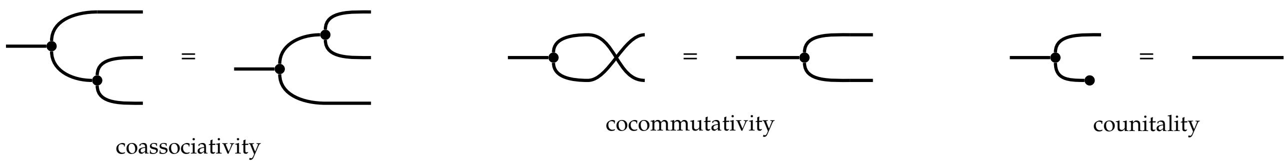
The delete relation is defined:

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

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.

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



(1.1)

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.

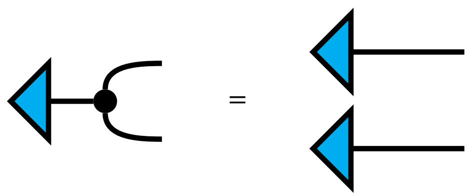
Coassociativity: says there is no difference between copying copies.

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 1.1.19 (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:

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

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

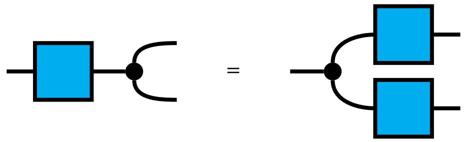
Definition 1.1.3 (Functor)**.** A functor $F : \mathcal{C} \to \mathcal{D}$ (read: with domain a category $\mathcal{C}$ and codomain a category $\mathcal{D}$) consists of two suitably related functions. An object function $F_0 : \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{D})$ and a morphism function (equivalently viewed as a family of functions indexed by pairs of objects of $\mathcal{C}$) $F_1(X, Y) : \mathcal{C}(X, Y) \to \mathcal{D}(F_0 X, F_0 Y)$. F_1 must map identities to identities – i.e., be such that for all $X \in \mathcal{C}$, $F_1(1_X) = 1_{F_0 X}$ – 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 1.1.4 (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 $\mathcal{D}$:

$$\begin{tikzcd} FX \arrow[r, "Ff"] \arrow[d, "\theta_X"] & FY \arrow[d, "\theta_Y"] \\ GX \arrow[r, "Gf"] & GY \end{tikzcd}$$

Remark 1.1.20. 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*.

Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 1.1.19 and Remark 1.1.20, 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 1.1, 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 1.1. In fact, quantum physicists *do* do this; see Dodo: [].

1.1.2 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

Definition 1.1.5 (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 1.1.6 (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 1.1.7 (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.

$$
\begin{tikzcd}
& ((W \otimes X) \otimes (Y \otimes Z)) \arrow[dd, "\alpha"] \\
(W \otimes (X \otimes (Y \otimes Z))) \arrow[ur, "\alpha"] \arrow[dd, "1\otimes\alpha"'] & \\
& (((W \otimes X) \otimes Y) \otimes Z) \\
(W \otimes ((X \otimes Y) \otimes Z)) \arrow[dr, "\alpha"'] & \\
& ((W \otimes (X \otimes Y)) \otimes Z) \arrow[uu, "\alpha\otimes 1"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
(X \otimes (I \otimes Y)) \arrow[dr, "\alpha"] \arrow[dd, "1\otimes\lambda"'] & \\
& ((X \otimes I) \otimes Y) \arrow[dl, "\rho\otimes 1"] \\
(X \otimes Y) &
\end{tikzcd}
$$

Remark 1.1.8 (Coherence)**.** Coherence is about getting rid of syntactic bureaucracy. Addition for example is a binary associative operation, and knowing that $(x + (y + z)) = ((x + y) + z)$ is what allows us to safely drop all possible bracketings and just write $1 + 2 + 3$. We further know that addition is a monoid with unit 0, so we can always write $x + 0 = x = 0 + x$. Now the situation is that we have replaced the associativity equation with associator natural transformations $((X \otimes Y) \otimes Z) \overset{\alpha_{XYZ}}{\to} (X \otimes (Y \otimes Z))$, and unit equations with left and right unitors $(X \otimes I) \overset{\rho_X}{\to} X \overset{\lambda_X}{\to} (I \otimes X)$. Recalling that we're happy with isomorphism in place of equality, we would like to know that every possible composite of these structural operations is an isomorphism.

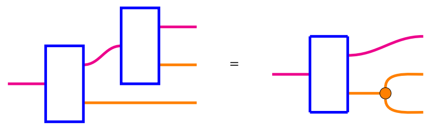
the value of a field – *independently of the specific programming language implementation of a database* – so that they had reasoning tools to ensure program correctness []. The problem is reducible to axiomatising a *rewrite*: we can think of updating a value as first calculating the new value, then *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.

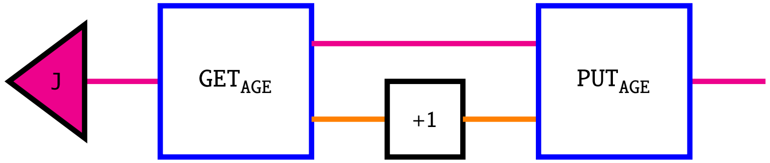
***GetGet**:* Getting a value from a field twice is the same as getting the value once and copying it.



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.



1.1.3 *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.

1.1.4 *Equational Constraints and Frobenius Algebras*

1.1.5 *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

Definition 1.1.9 (Symmetric Monoidal Category)**.** A symmetric monoidal category is a monoidal category with an additional natural isomorphism $\theta : - \otimes = \mapsto = \otimes -$. Coherence requires the following pair of hexagons
.

$$\begin{tikzcd}
(X \otimes (Y \otimes Z)) \arrow[r, "\theta"] \arrow[d, "\alpha^{-1}"'] & (Z \otimes (X \otimes Y)) \arrow[d, "\alpha"] \\
(X \otimes (Z \otimes Y)) \arrow[d, "1\otimes\theta"'] & ((Z \otimes X) \otimes Y) \arrow[d, "\theta^{-1}"] \\
(X \otimes (Z \otimes Y)) \arrow[r, "\alpha"'] & ((X \otimes Z) \otimes Y)
\end{tikzcd}$$

$$\begin{tikzcd}
(X \otimes (Y \otimes Z)) \arrow[r, "\theta"] \arrow[d, "\alpha"'] & ((Y \otimes Z) \otimes X) \arrow[d, "\alpha^{-1}"] \\
((X \otimes Y) \otimes Z) \arrow[d, "\theta^{-1}"'] & (Y \otimes (Z \otimes X)) \arrow[d, "1\otimes\theta"] \\
((Y \otimes X) \otimes Z) \arrow[r, "\alpha^{-1}"'] & (Z \otimes (X \otimes Y))
\end{tikzcd}$$

Remark 1.1.10 (Coherence for symmetric monoidal categories)**.** Going back to addition, we can rearrange sums by commutativity of the addition monoid: $x + y = y + x$. In the monoidal setting, the natural isomorphism θ is the twisting of wires. Just as we would like to have addition equations such as $(x + (y + z)) = ((z + y) + x)$, we would like the twists θ to behave well with respect to the bundling of objects and morphisms with the associator α.

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.

1.2 *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.

1.2.1 *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 1.1: In English, we may consider a noun to have type n, and an transitive english 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.

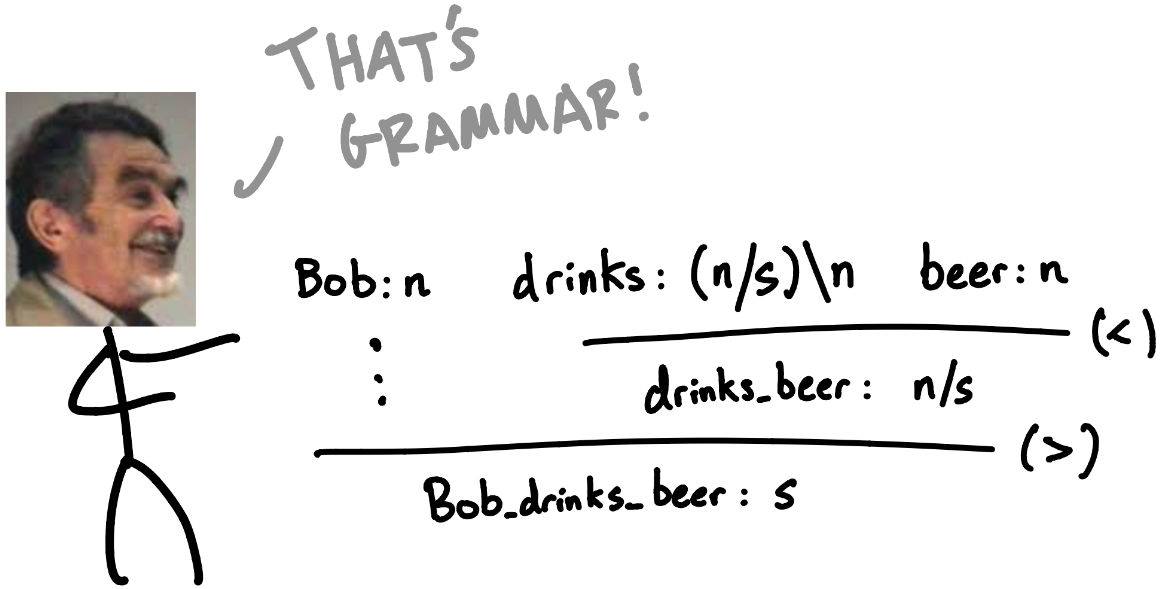


Figure 1.2: We can notice an asymmetry in the above formulation when we examine the transitive verb type $(n/s) \setminus n$ again; it asks first for a noun to the right, and then a noun to the left. We could just as well have asked for the nouns in the other order with the typing $(n/s) \setminus n$ and obtained all of the same proofs.

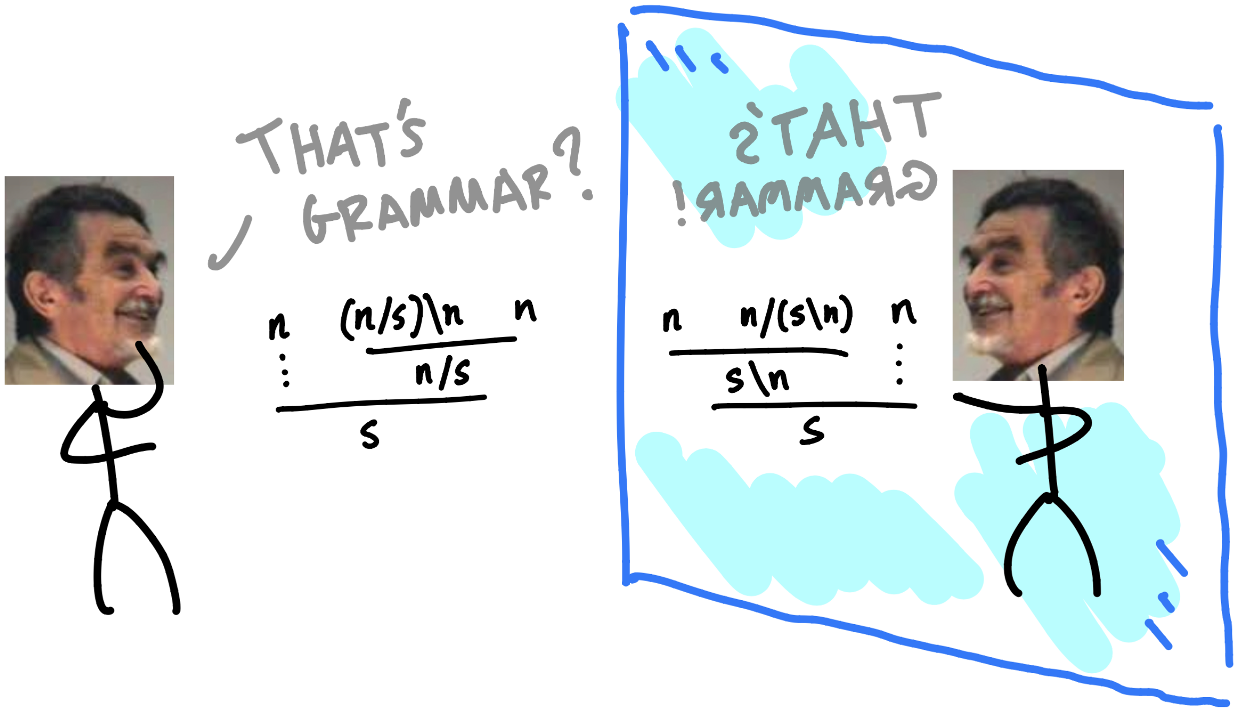
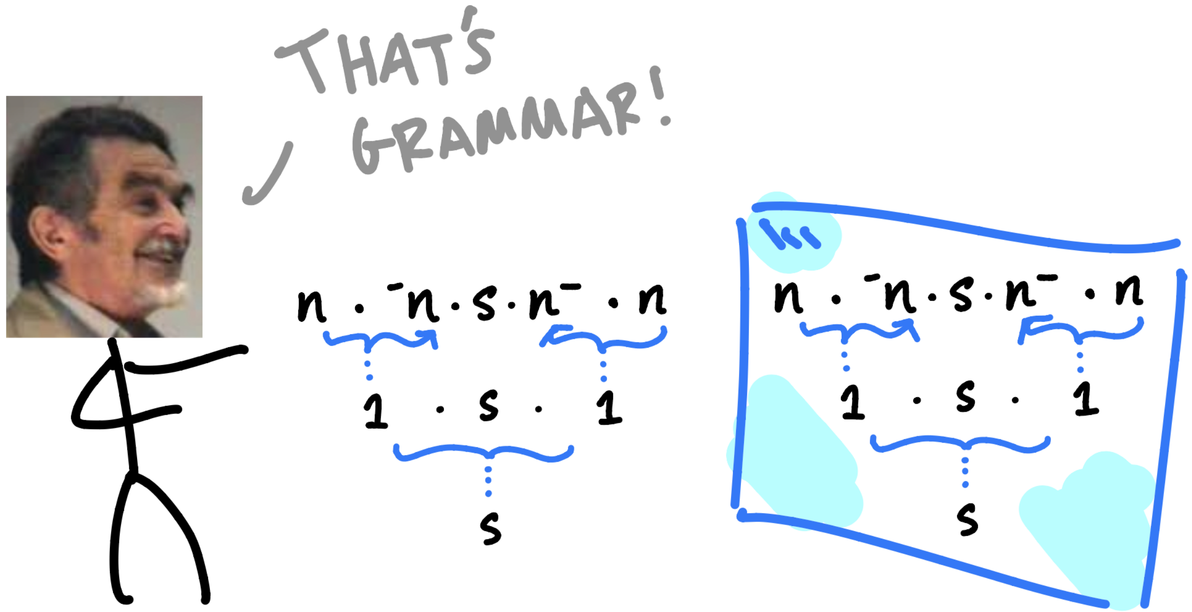


Figure 1.3: To eliminate this asymmetry, Lambek devised pregroup grammars. Whereas a group is a monoid with inverses up to left- and right-multiplication, a pregroup weakens the requirement for inverses so that all elements have distinct left- and right- inverses, denoted x^{-1} and ${}^{-1}x$ respectively. 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.

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



1.2.2 Coecke's Composition

Figure 1.4: 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.

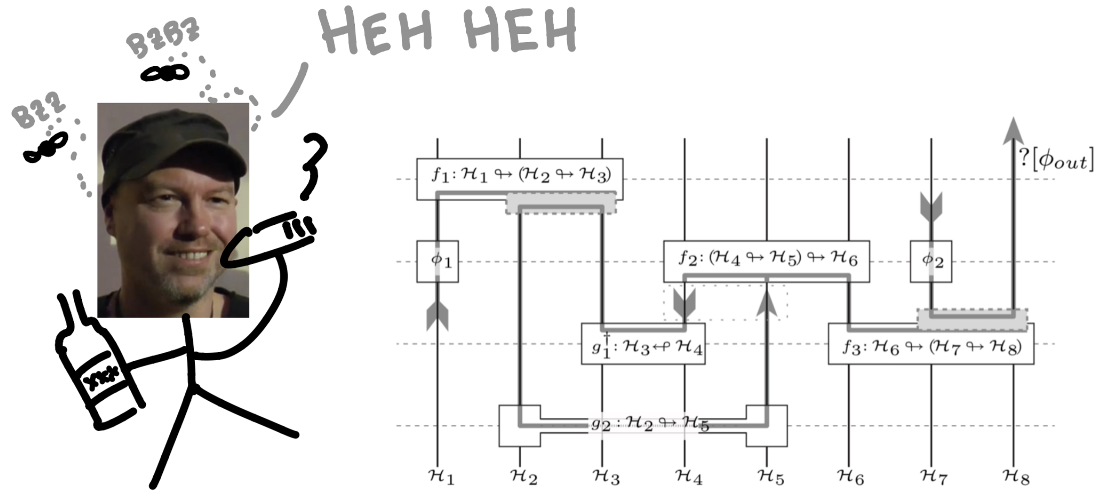


Figure 1.5: There were two impressive things about these diagrams. First, the effects such as transparencies for text boxes and curved serifs for angled arrows give a modern feel, but they were done manually in macdraw, the diagrammatic equivalent of sticks and stones. Second, though the diagrams were 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.

1.2.3 Categorical quantum mechanics

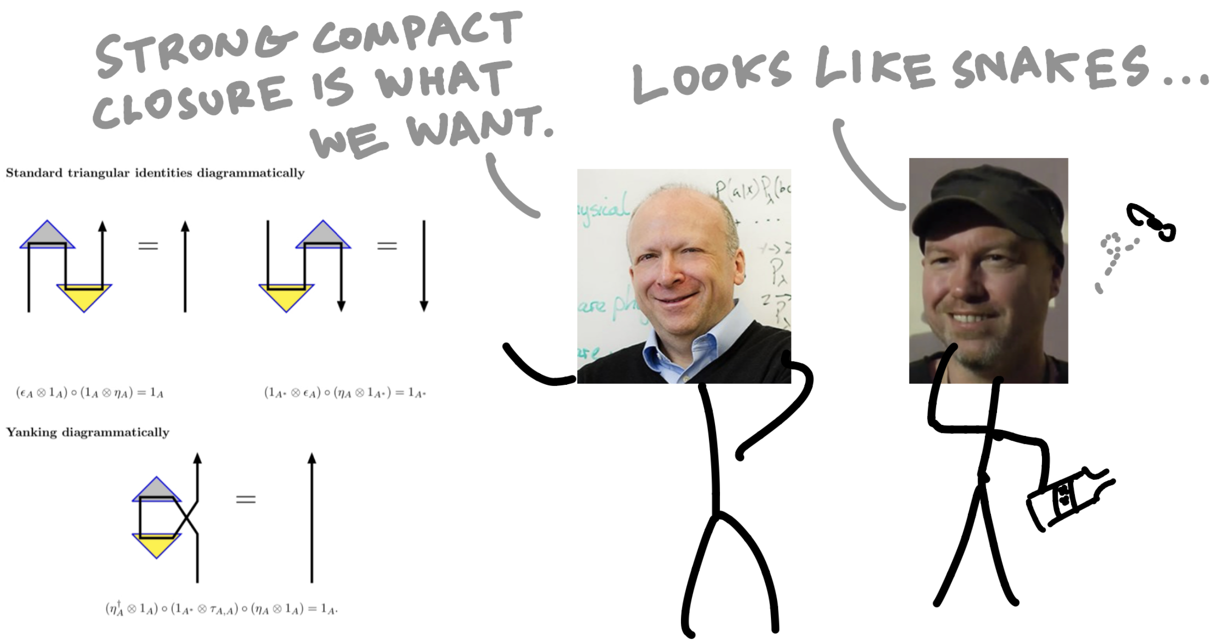
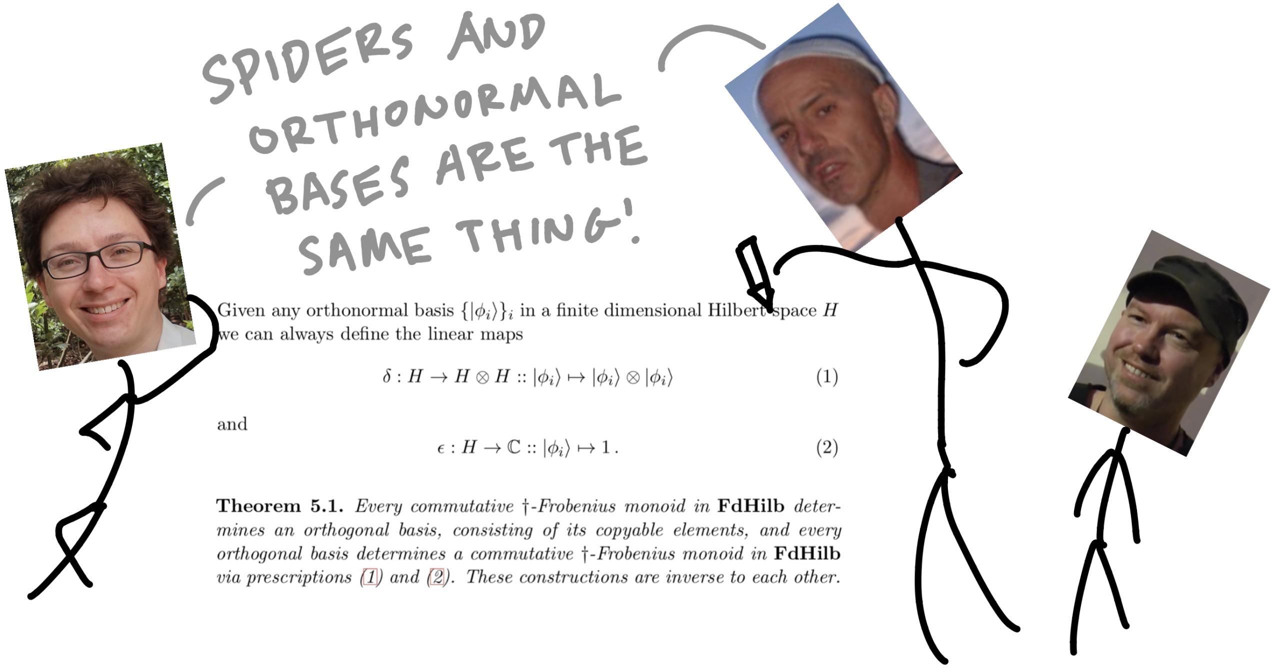


Figure 1.6: Category theorists and physicists such as Abramsky and Baez were excited about these diagrams, which looked like string diagrams waiting to be made formal. The graphical cups and caps in the important diagram were determined to correspond to a special form of symmetric monoidal closed category called strong compact closed.

Figure 1.7: 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.



Given any orthonormal basis $\{|\phi_i\rangle\}_i$ in a finite dimensional Hilbert space H we can always define the linear maps

$$\delta : H \to H \otimes H :: |\phi_i\rangle \mapsto |\phi_i\rangle \otimes |\phi_i\rangle \tag{1}$$

and

$$\epsilon : H \to \mathbb{C} :: |\phi_i\rangle \mapsto 1\,. \tag{2}$$

Theorem 5.1. *Every commutative †-Frobenius monoid in* **FdHilb** *determines an orthogonal basis, consisting of its copyable elements, and every orthogonal basis determines a commutative †-Frobenius monoid in* **FdHilb** *via prescriptions (1) and (2). These constructions are inverse to each other.*

Figure 1.8: 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 1.9: 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.

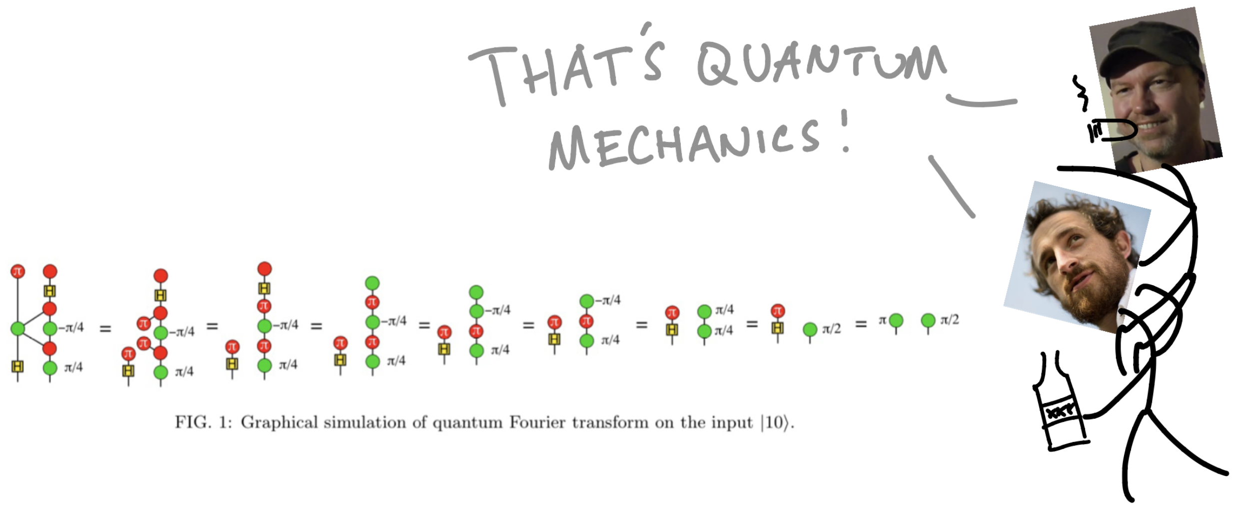


Figure 1.10: And so the stage was set for a purely diagrammatic treatment of ZX quantum mechanics. The story of ZX diverges away from our interest, so I will summarise what happened afterwards. In no particular order, the development of ZX went on to accommodate a third axis of measurement to yield a 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 .

1.2.4 *Enter computational linguistics*

Figure 1.11: 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.

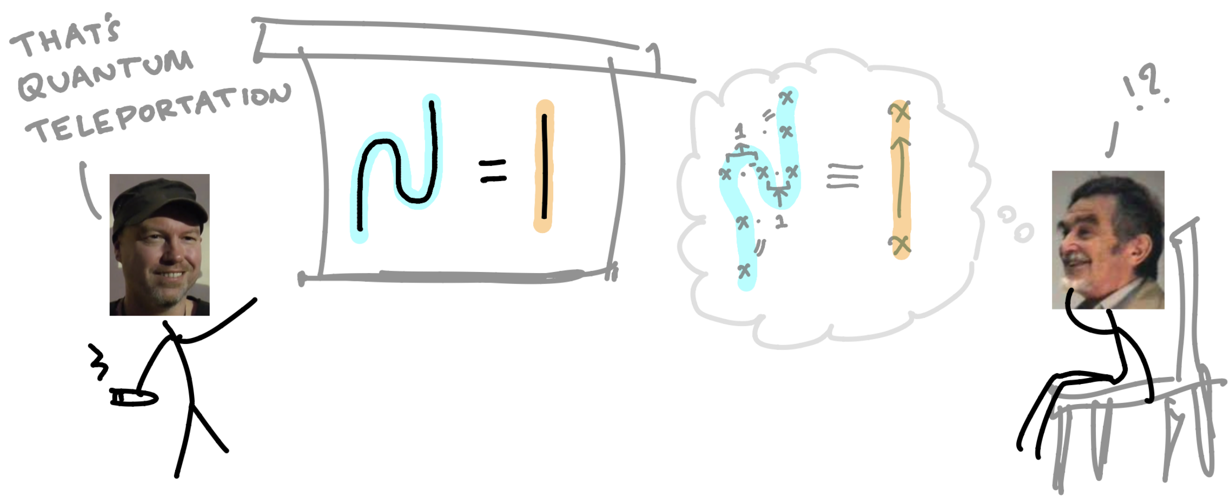
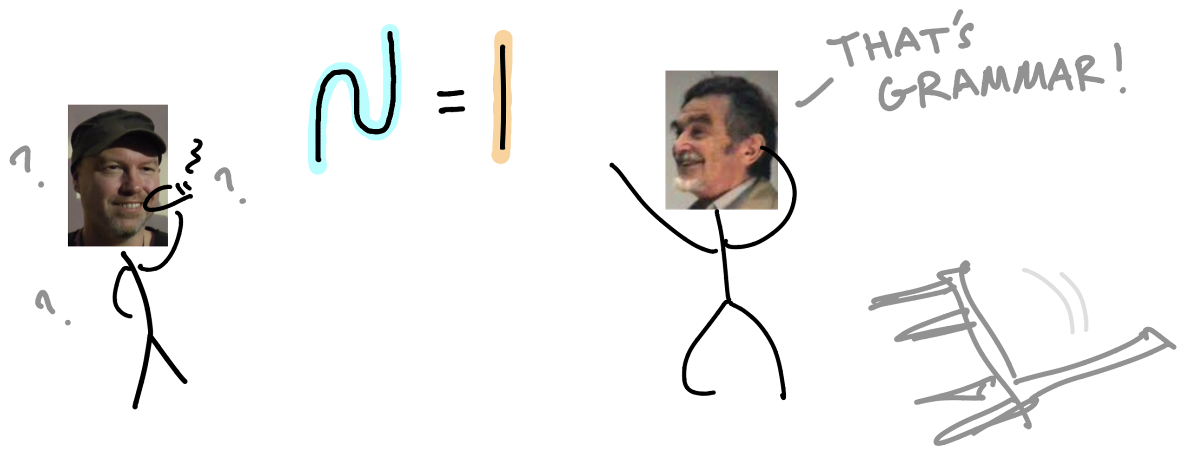


Figure 1.12: Bob and Jim's meeting put the adjectives *compositional* and *categorical* on the same table, but the cake wasn't ready. Two more actors Steve and Mehrnoosh were required to introduce *distributional*, which refers to Firth's maxim CITE "you shall know a word by the company it keeps". In 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.



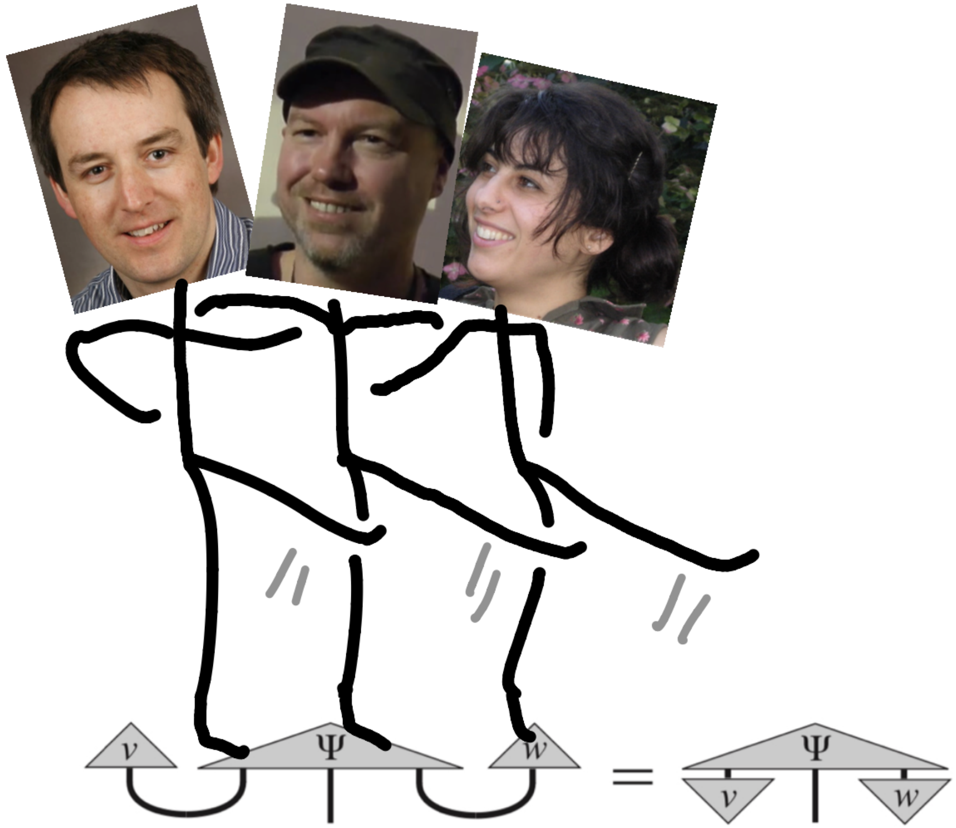


Figure 1.13: Steve Clark was a professor in the computer science department at Oxford, and he was wondering how to compose vector-based semantic representations. Steve asked Bob, who realised suddenly what Jim was talking about. Mediated by the linguistic expertise of Mehrnoosh who was a postdoctoral 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 hadamard 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:

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

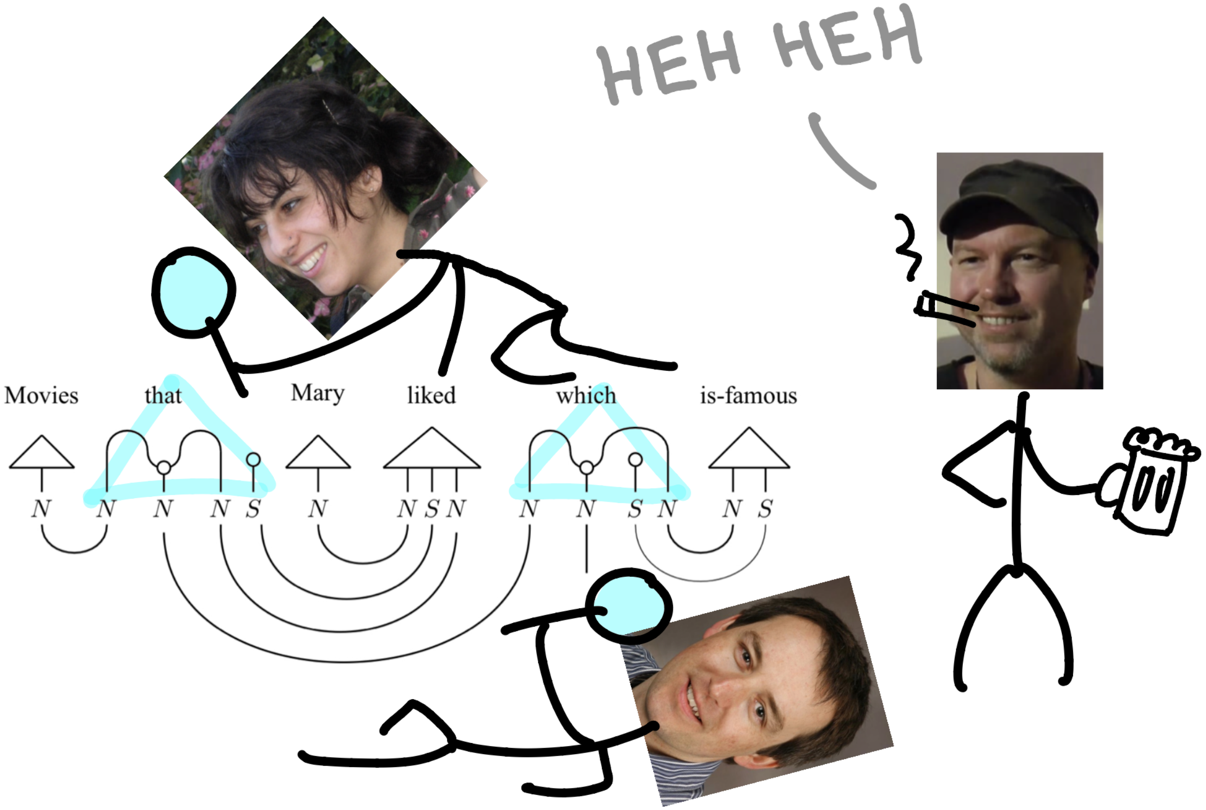


Figure 1.14: In *the frobenius anatomy of relative pronouns*CITE , the trio realised that spiders could play the role of relative pronouns, which was genuinely novel linguistics. If one follows the noun-wire of "movies", one sees that by declaring the relative pronoun to be a vector made up of a particular 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 .

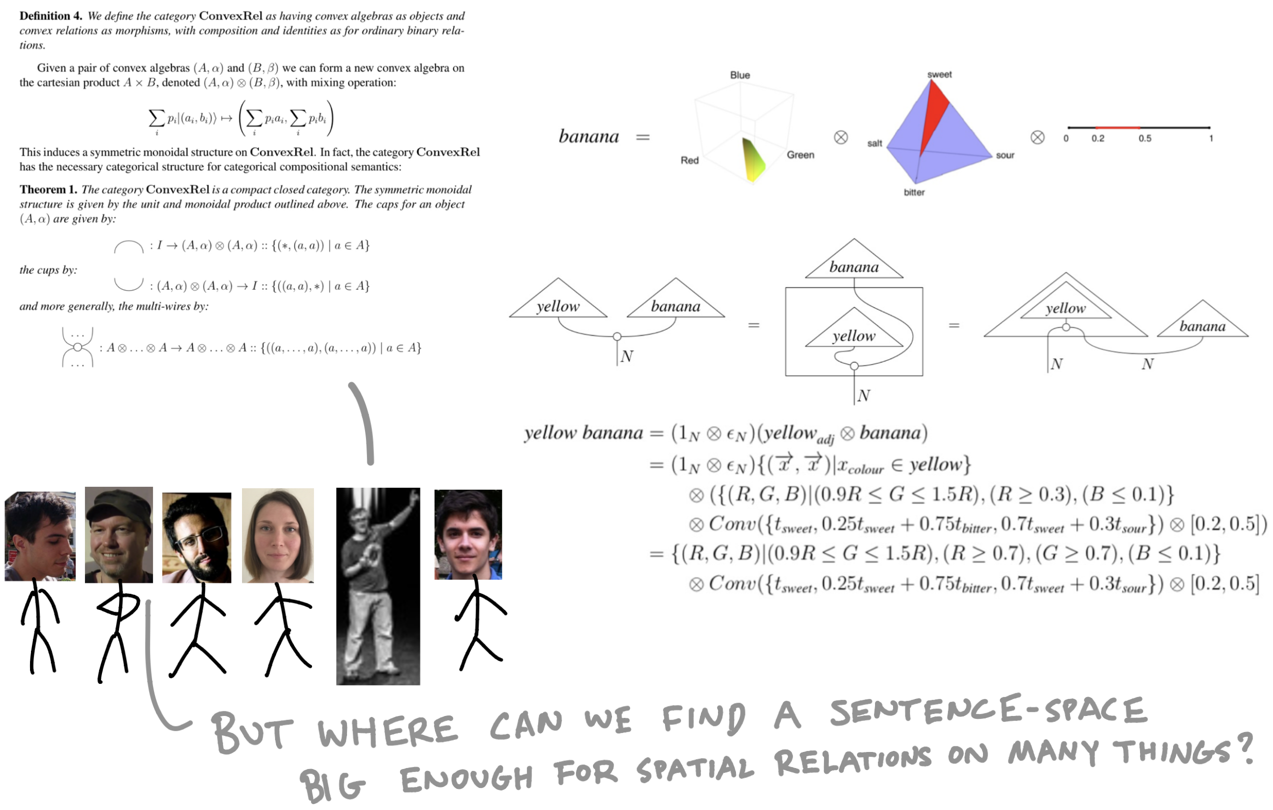


Figure 1.15: Keeping the structure of the diagrams but seeking set-relational rather than vector-based semantics, a bridge was made between linguistics and cognitive science in *interacting conceptual spaces I*CITE . Briefly, Gärdenfors posits that spatial representations of concepts mediate raw sense 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 .

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

Figure 1.16: 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 .

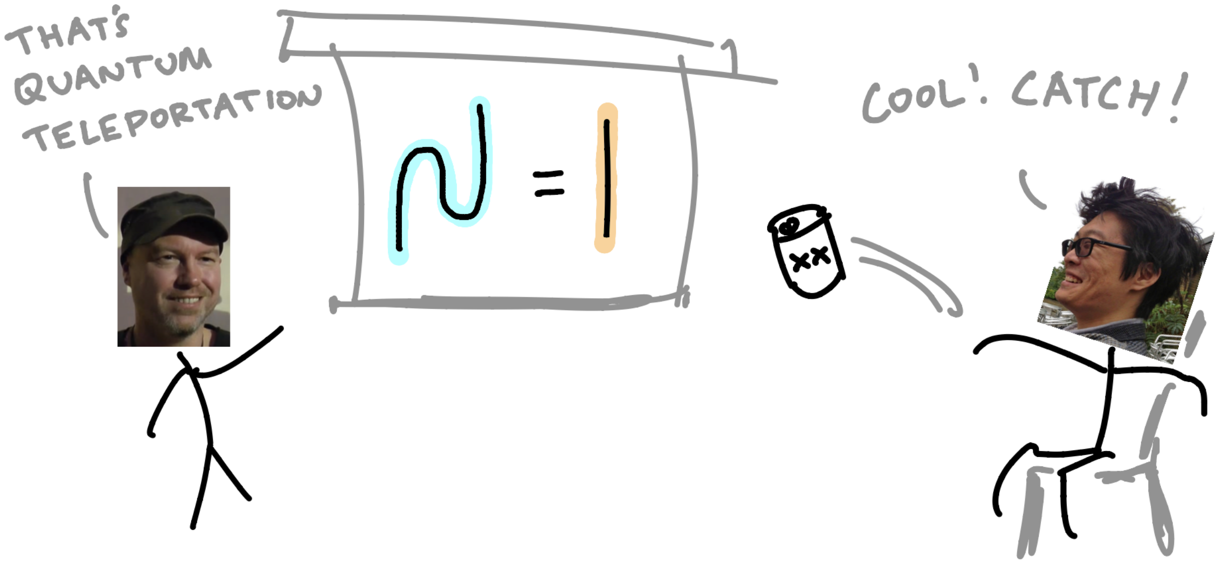


Figure 1.17: I met Bob in my master's in 2019, where he taught the picturing quantum processes course. When quantum teleportation was explained in half a minute by a diagram, I decided to pursue a DPhil in diagrammatic mathematics. In the last lecture, I threw Bob a cider, after which he seemed to like 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.

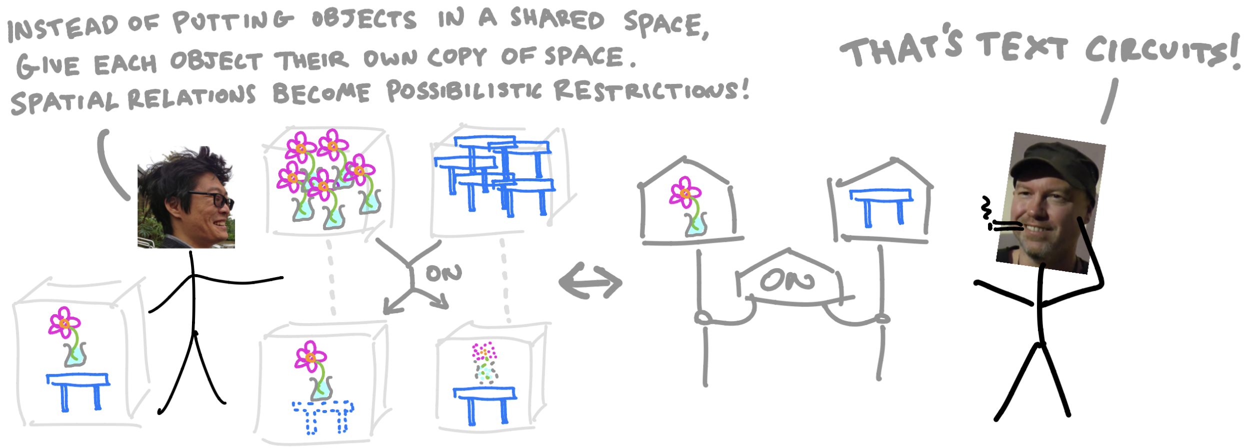


Figure 1.18: Then COVID happened. During the first lockdown, I visited Bob's garden under technically legal circumstances, and I suggested a solution to the longstanding problem of representing linguistic spatial relationships. My theoretical concern was the culprit: the initial attempts at the problem 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.

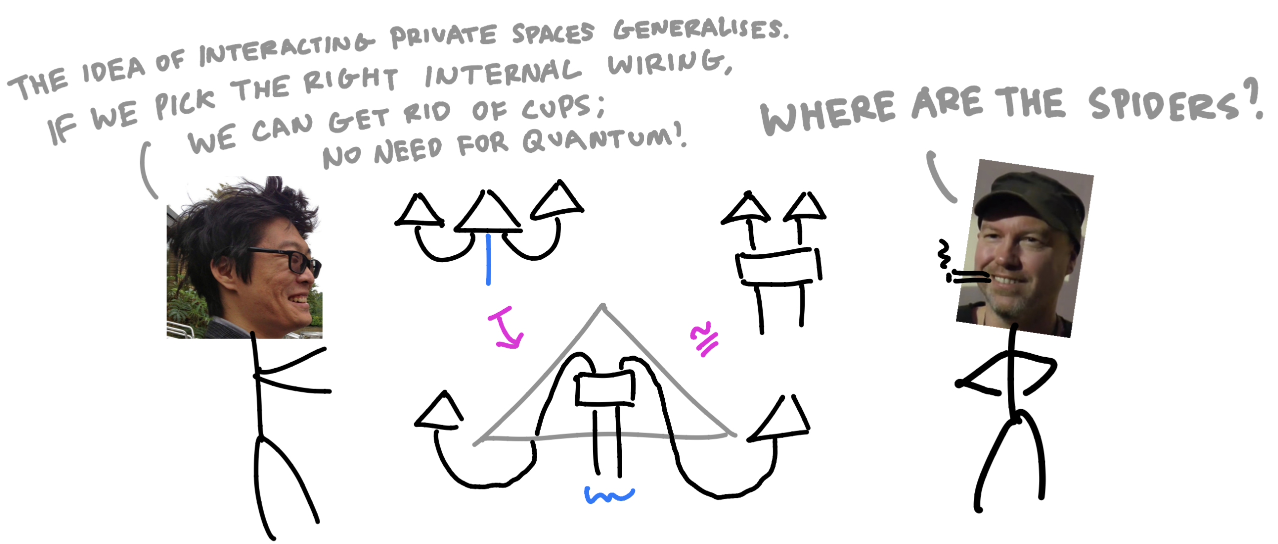


Figure 1.19: That this move of splitting up the sentence-wire into a sentence-dependent collection of wires was sufficient to solve what had appeared to be a difficult problem prompted some re-examination of foundations. The free autonomisation trick in conjunction with sentence-wire-as-tensored-nouns seemed promising, but it became clear that right way to drown a DisCoCat thoroughly was to explain and eliminate the spiders.

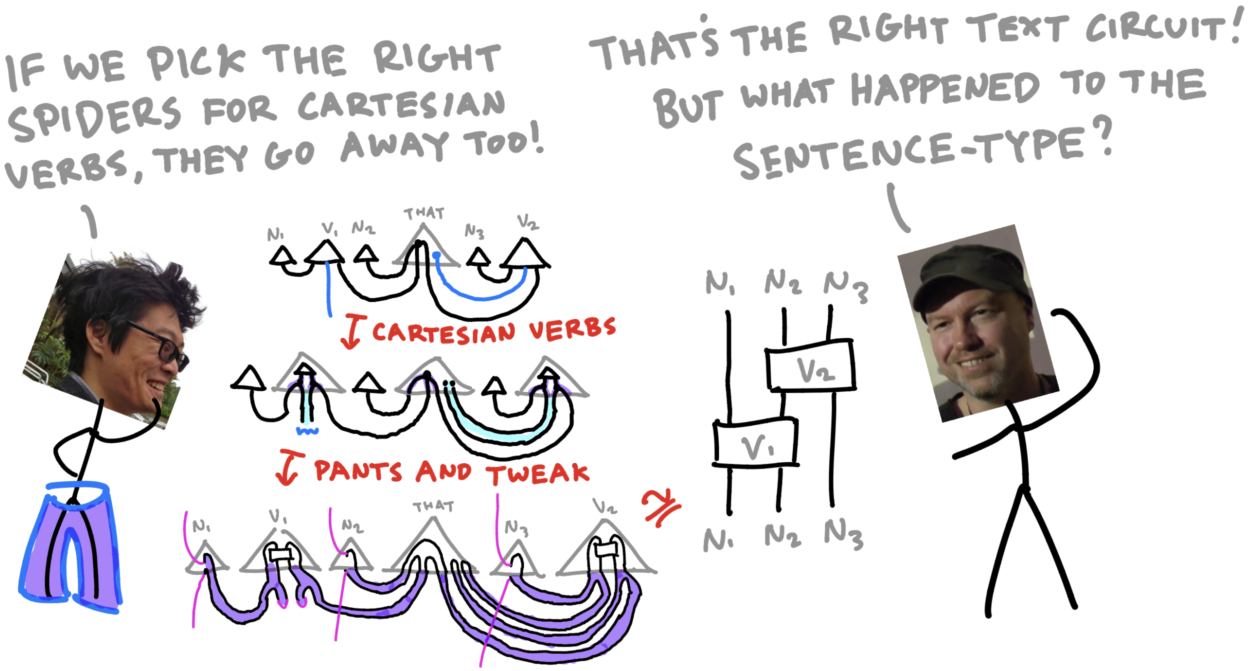


Figure 1.20: I then discovered that by interpreting spiders as the well-known "pair of pants" algebra in a compact closed monoidal setting allowed for a procedure in which the final form was purely symmetric monoidal – the absence of cups and caps meant that there was no practical necessity to interpret 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 1.21: 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.

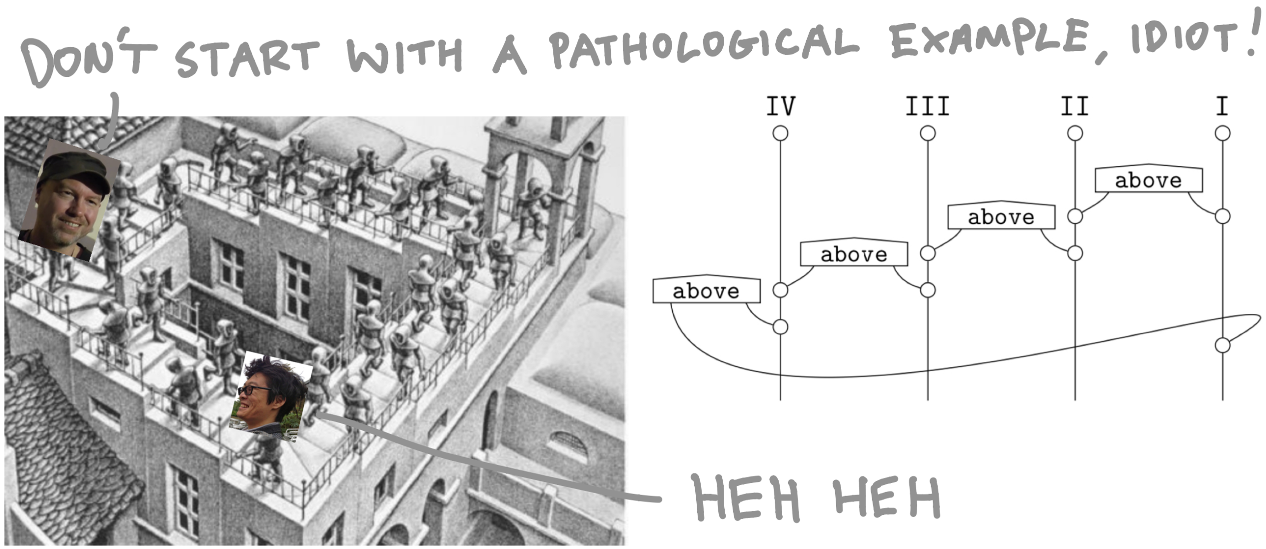


Figure 1.22: We wrote up the story about spaces in CITE , the spiritual successor to *interacting conceptual spaces I*. We could formally calculate the meanings of sentences that used linguistic spatial relations, all using a simple and tactile diagrammatic calculus.

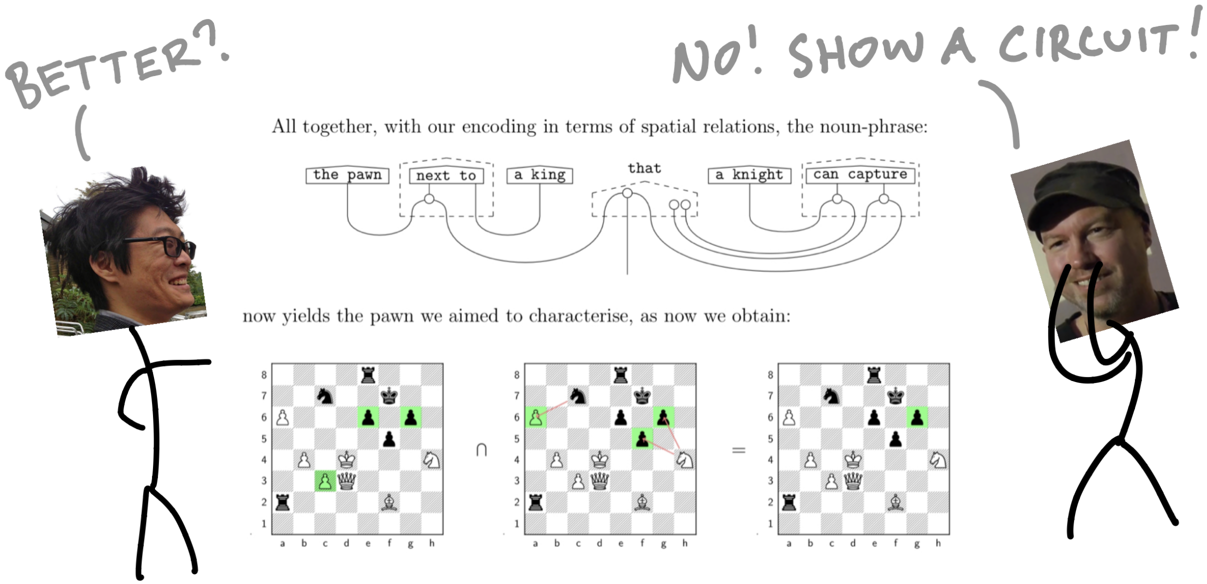
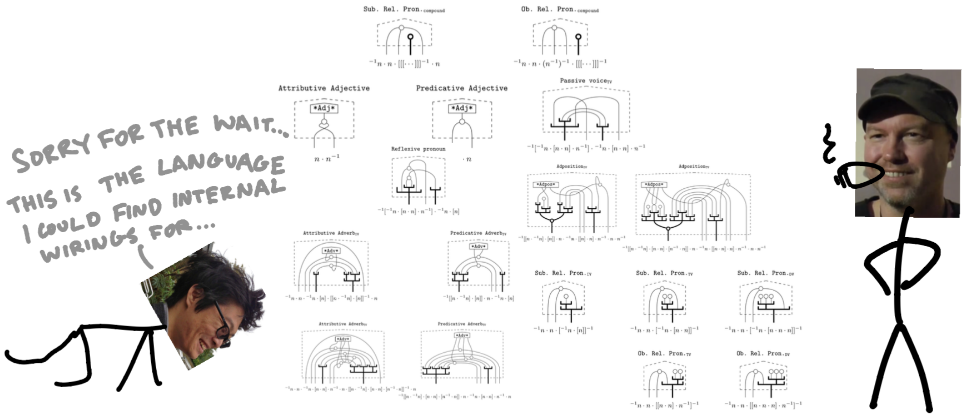


Figure 1.23: The paper on spatial relations actually came very late, because I was busy with Bob's ludicrous request to go turn "all of language" into circuits. I bitched and moaned about how I wasn't a linguist and how it was an impossible task, but I was in too deep to back out.

Figure 1.24: 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".



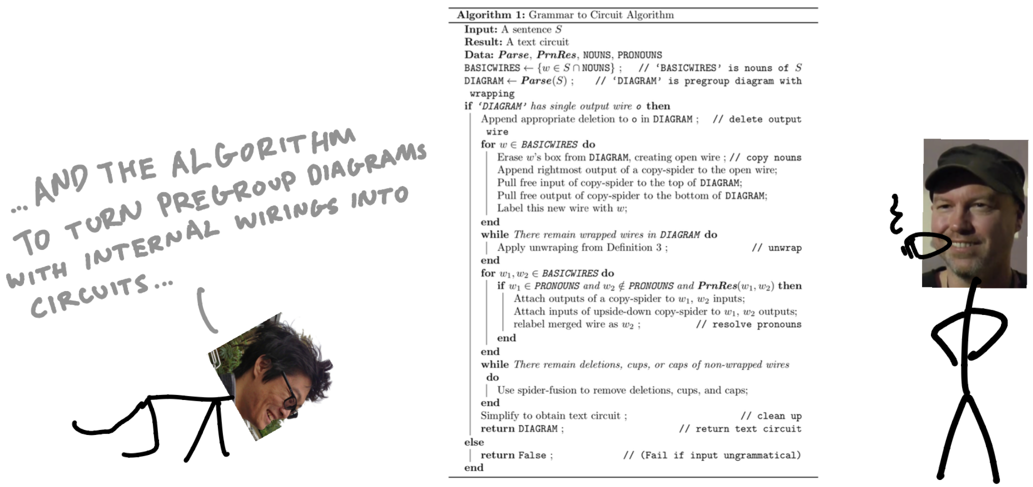


Figure 1.25: Bob had a good point. Everything worked, but we had no understanding as to why, and accordingly, whether or not it would all break. At this point in time, Jonathon Liu, who was a masters' student I taught during COVID, had committed the error of thinking diagrams were cool, and was now hanging 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.

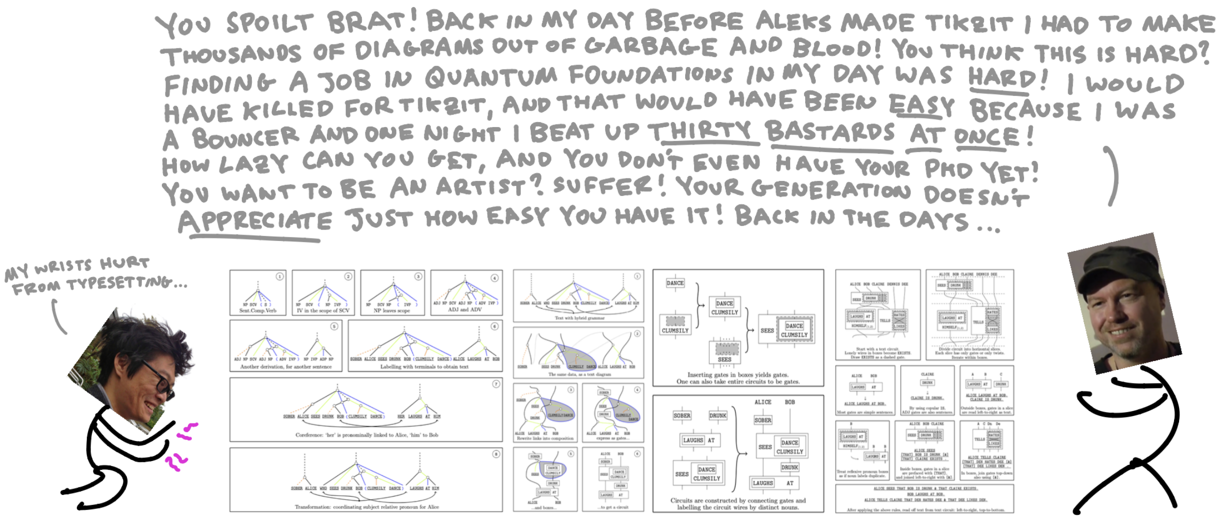


Figure 1.26: I realised that our intuitions were coming from an implicit productive grammar, rather than a parsing one, and that the path of least resistance for obtaining formal guarantees for the language-to-circuit procedure was to just handcraft a generative grammar for the fragment of language we 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.

2

Text circuits

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.

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

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

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

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.

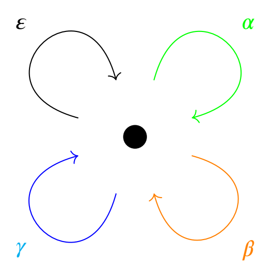


Figure 2.1: The category in question can be visualised as a commutative diagram.

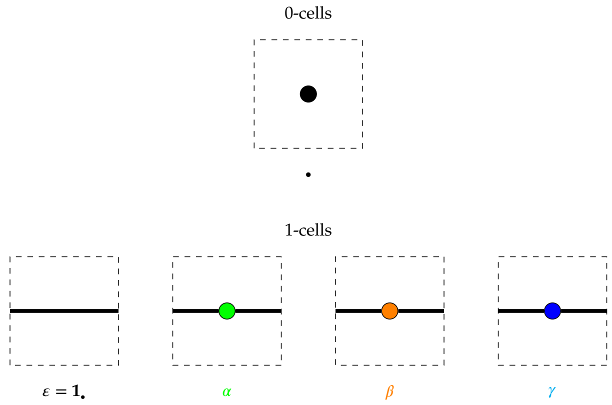


Figure 2.2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from 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 $\epsilon \cdot x = x = \epsilon \cdot x$.

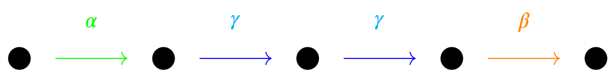
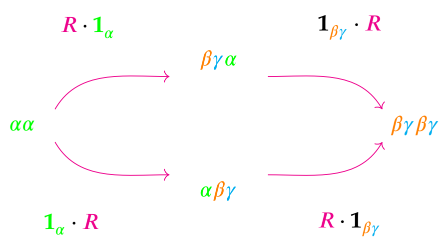
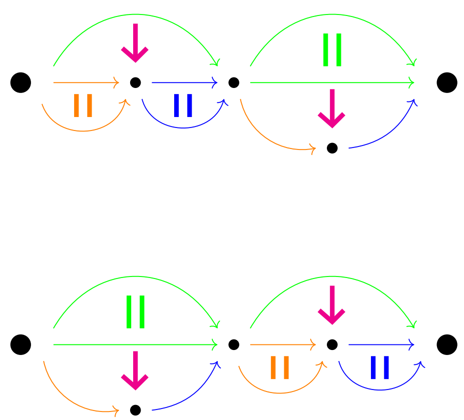


Figure 2.3: For a concrete example, we can depict the string $\alpha \cdot \gamma \cdot \gamma \cdot \beta$ as a morphism in a commuting diagram.

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-

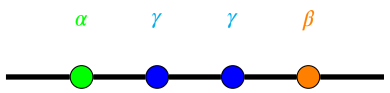


Figure 2.4: The string-diagrammatic view, where $\star$ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

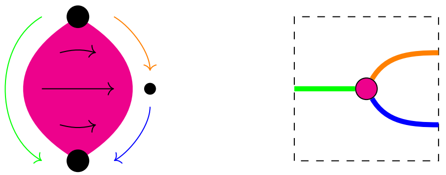


Figure 2.5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual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

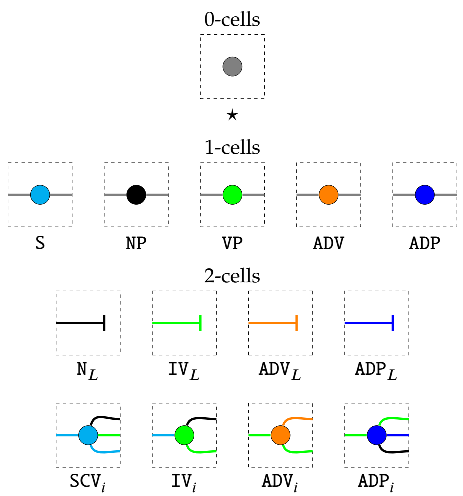
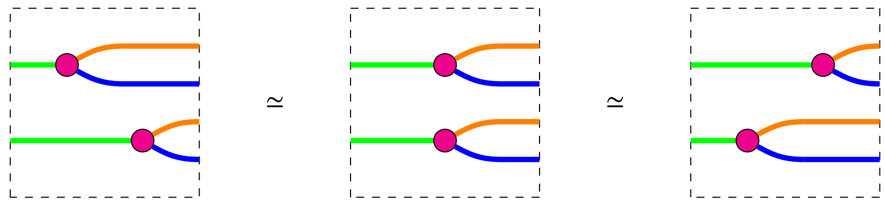


Figure 2.6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for `Alice sees Bob quickly run to school`.

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.

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

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



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

2.1.2 Tree Adjoining Grammars

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

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

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

Auxiliary trees satisfy the following constraints:

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

Further, there are two operations to build what are called *derived trees* from elementary and 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 $\mathtt{X}^{\star}$.

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

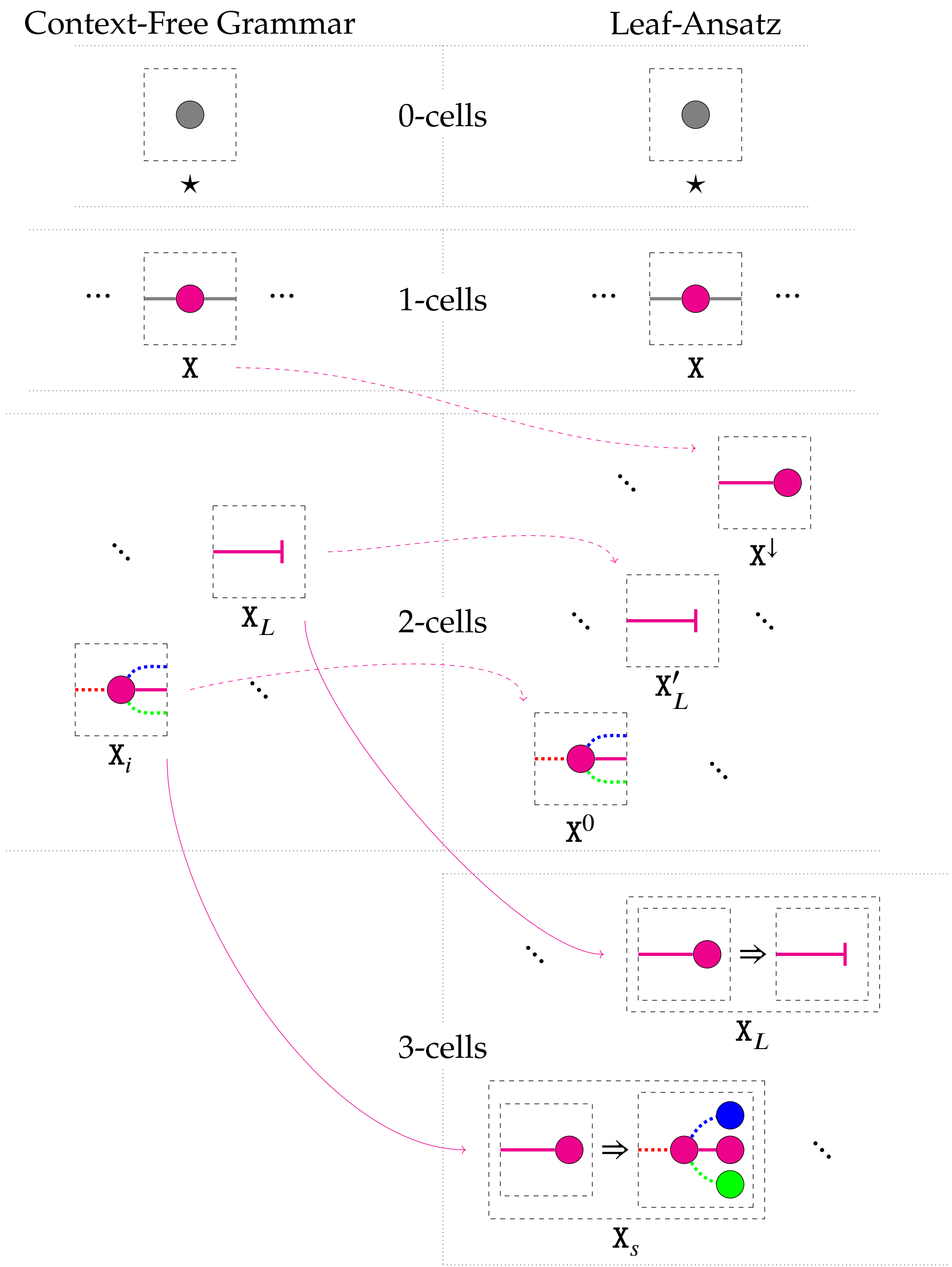


Figure 2.7:

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.

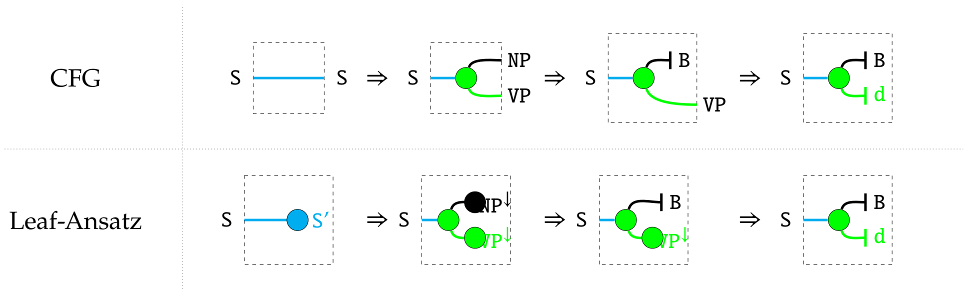


Figure 2.8: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction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

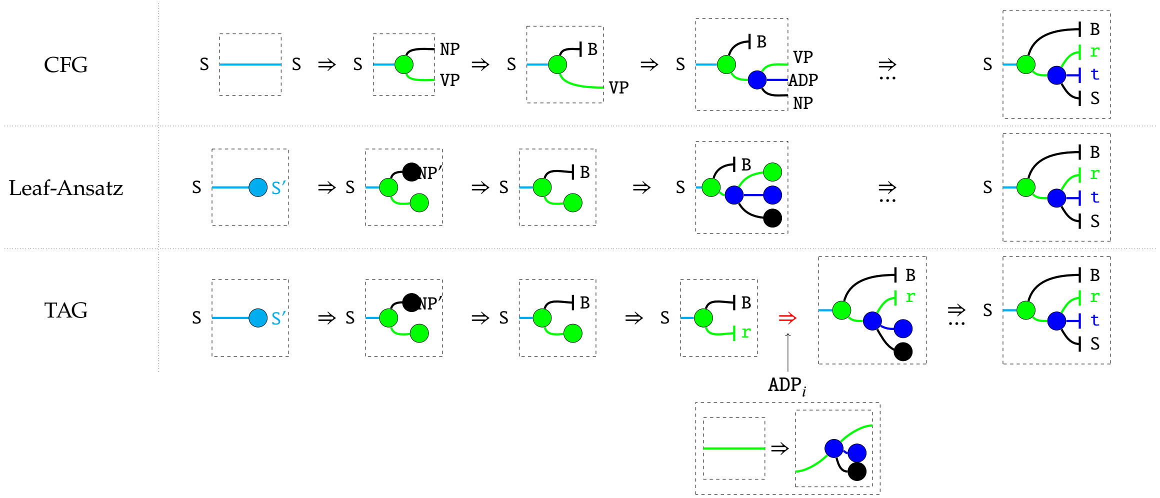
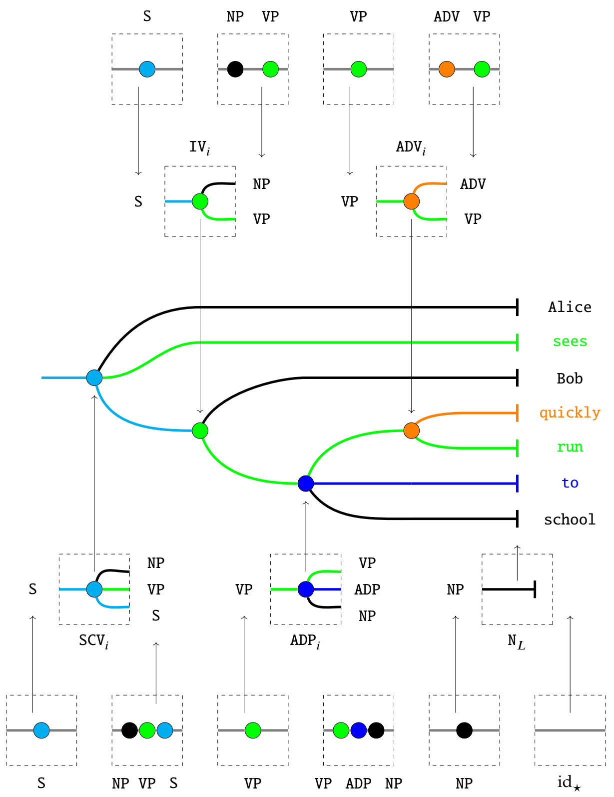
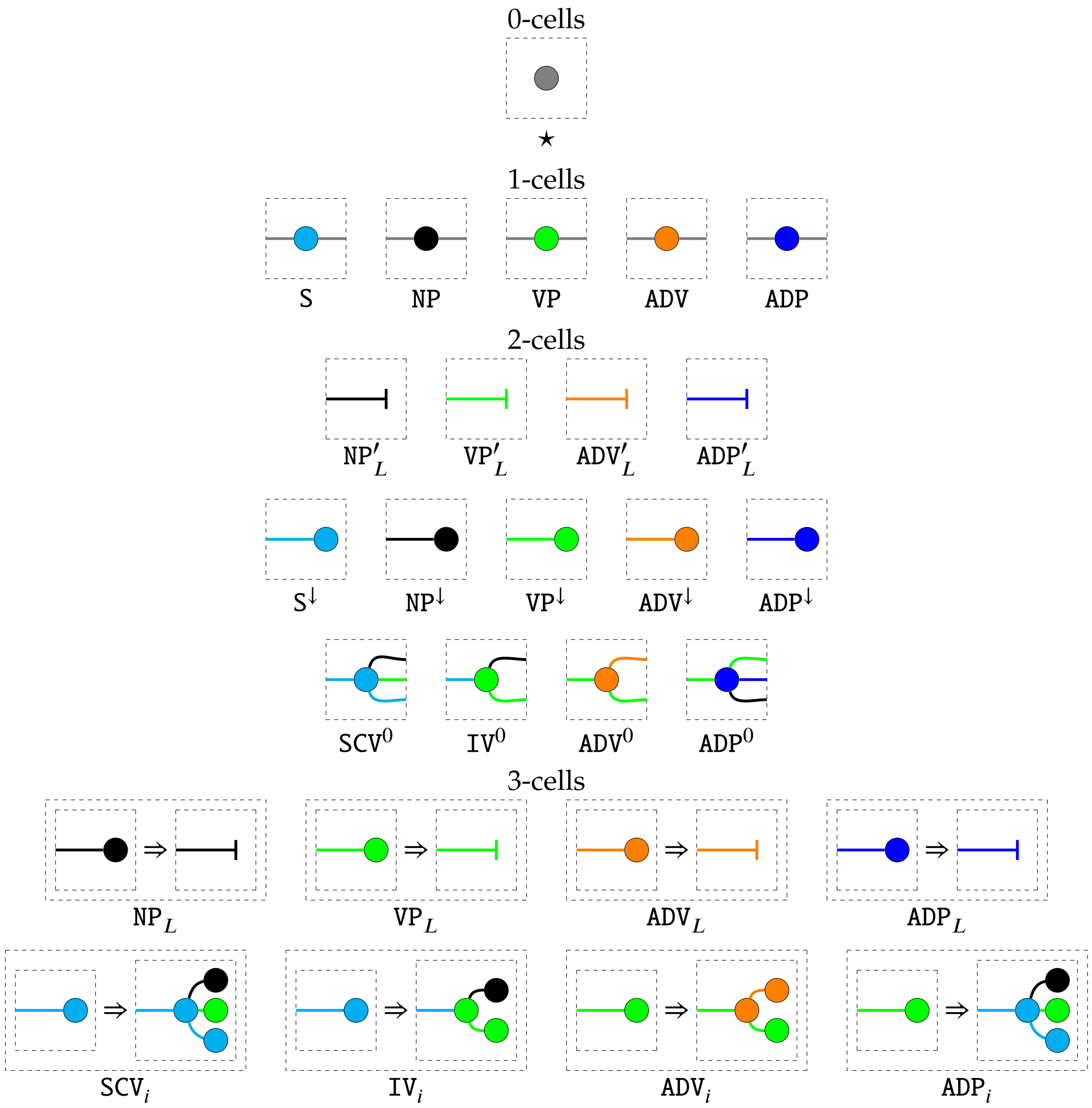


Figure 2.9: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted 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.





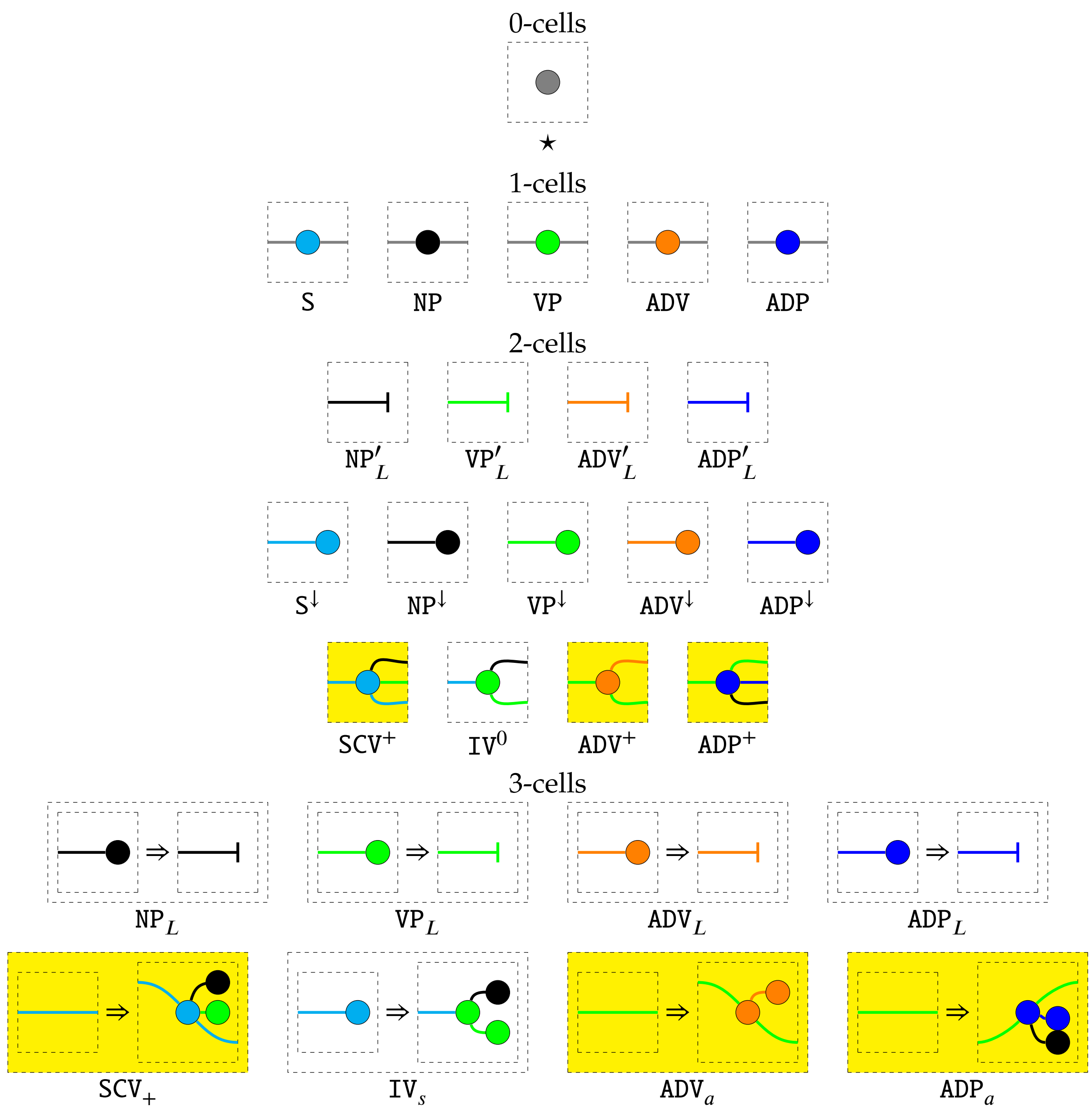


Figure 2.11: TAG signature of `Alice sees Bob quickly run to school`. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction 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

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.

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 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.

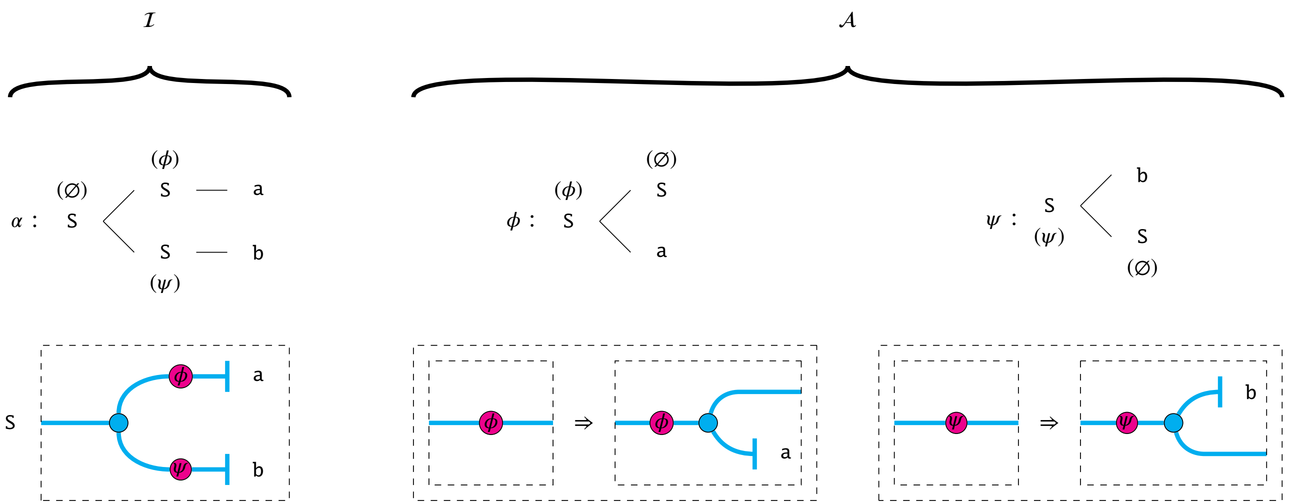


Figure 2.12: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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.

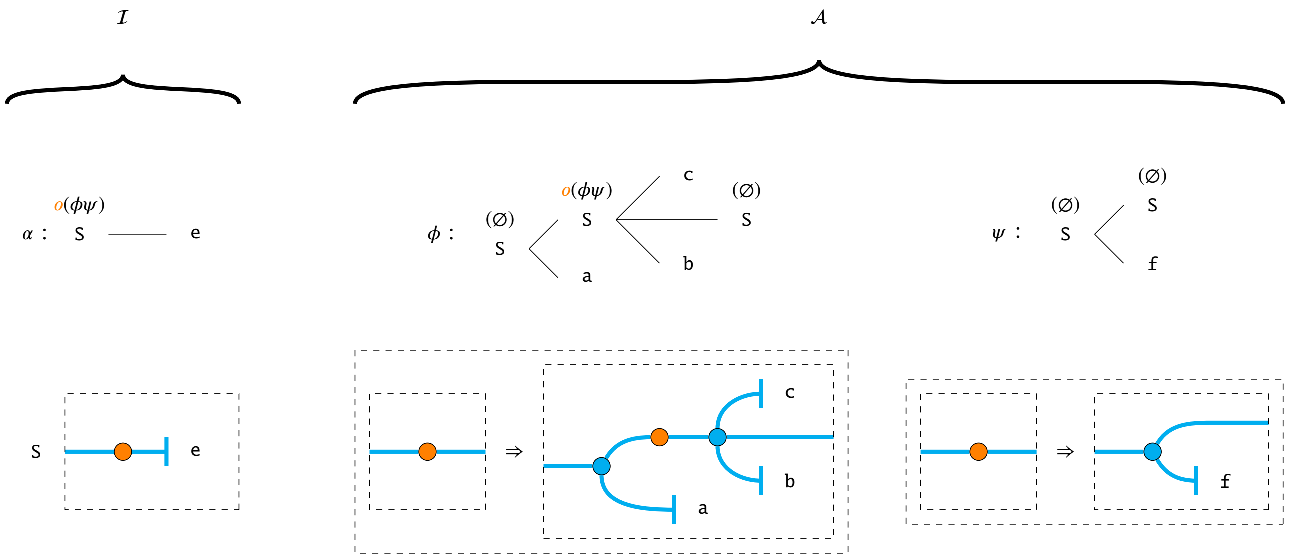
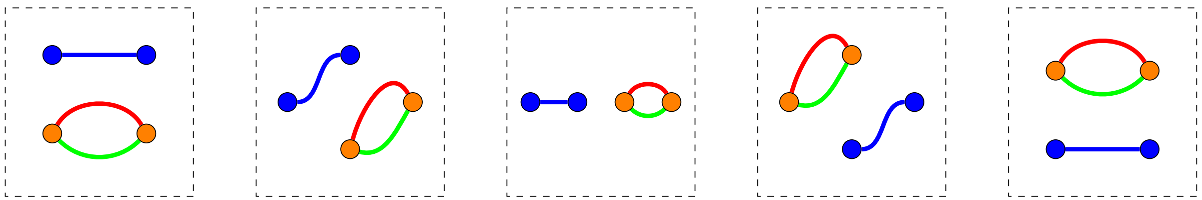


Figure 2.13: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n-categorical signature. The initial tree 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.

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

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



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

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.

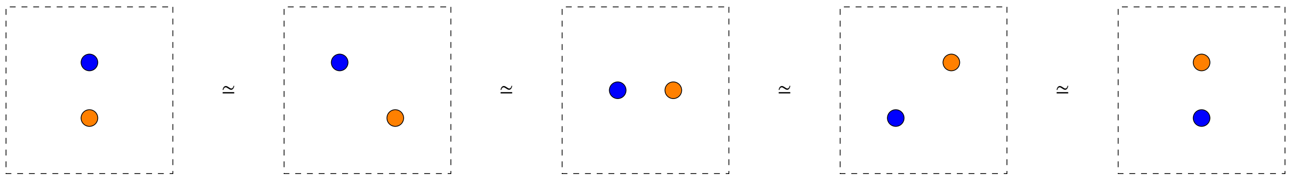


Figure 2.15: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument CITE is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting 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.

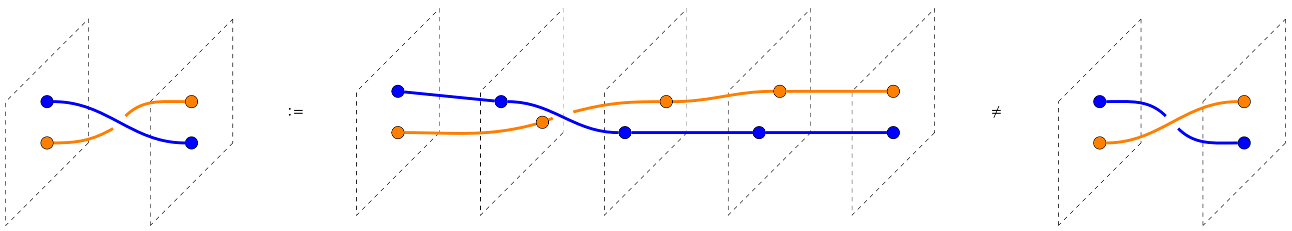


Figure 2.16: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies CITE , which are continuous bijective transformations that don't let wires double back on themselves, we can identify 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).

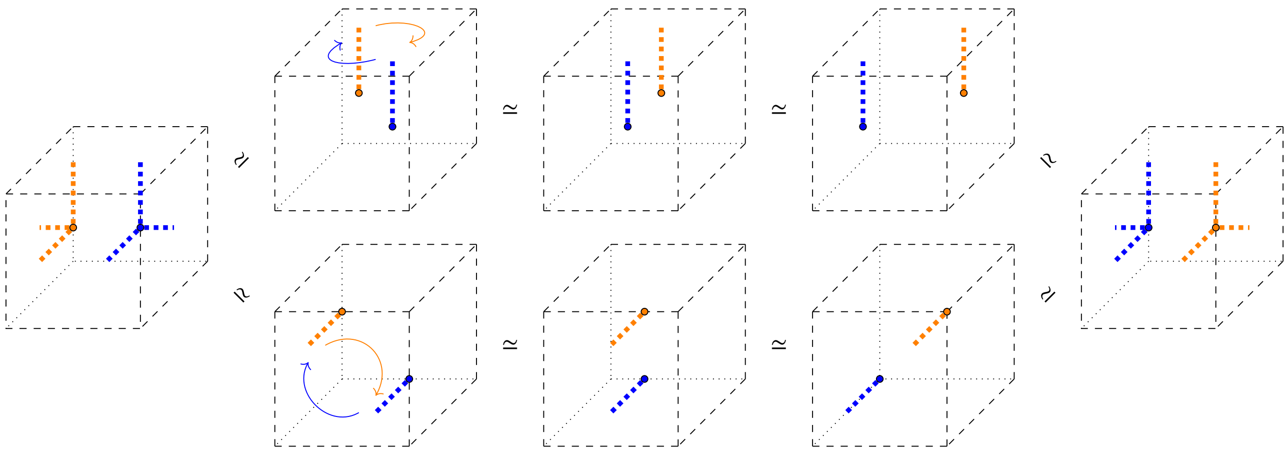


Figure 2.17: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise 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:

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

To recap: We follow the convention that object dimensions start at 0. In a 1-object-2-category we obtain planar string diagrams for monoidal categories, which are equivalent up to processive isotopies in an ambient 2-dimensional space. In a 1-object-3-category, we obtain string diagrams for braided monoidal categories that are equivalent up to processive isotopy in an ambient 3-dimensional space – the page with depth. In a

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

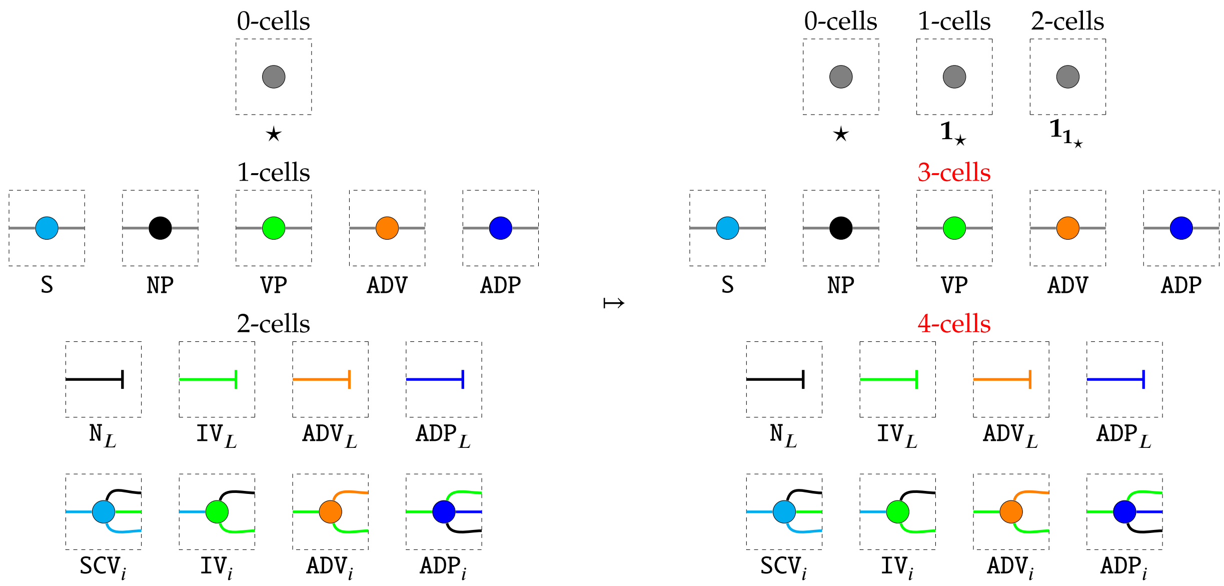


Figure 2.18: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with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

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.

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.

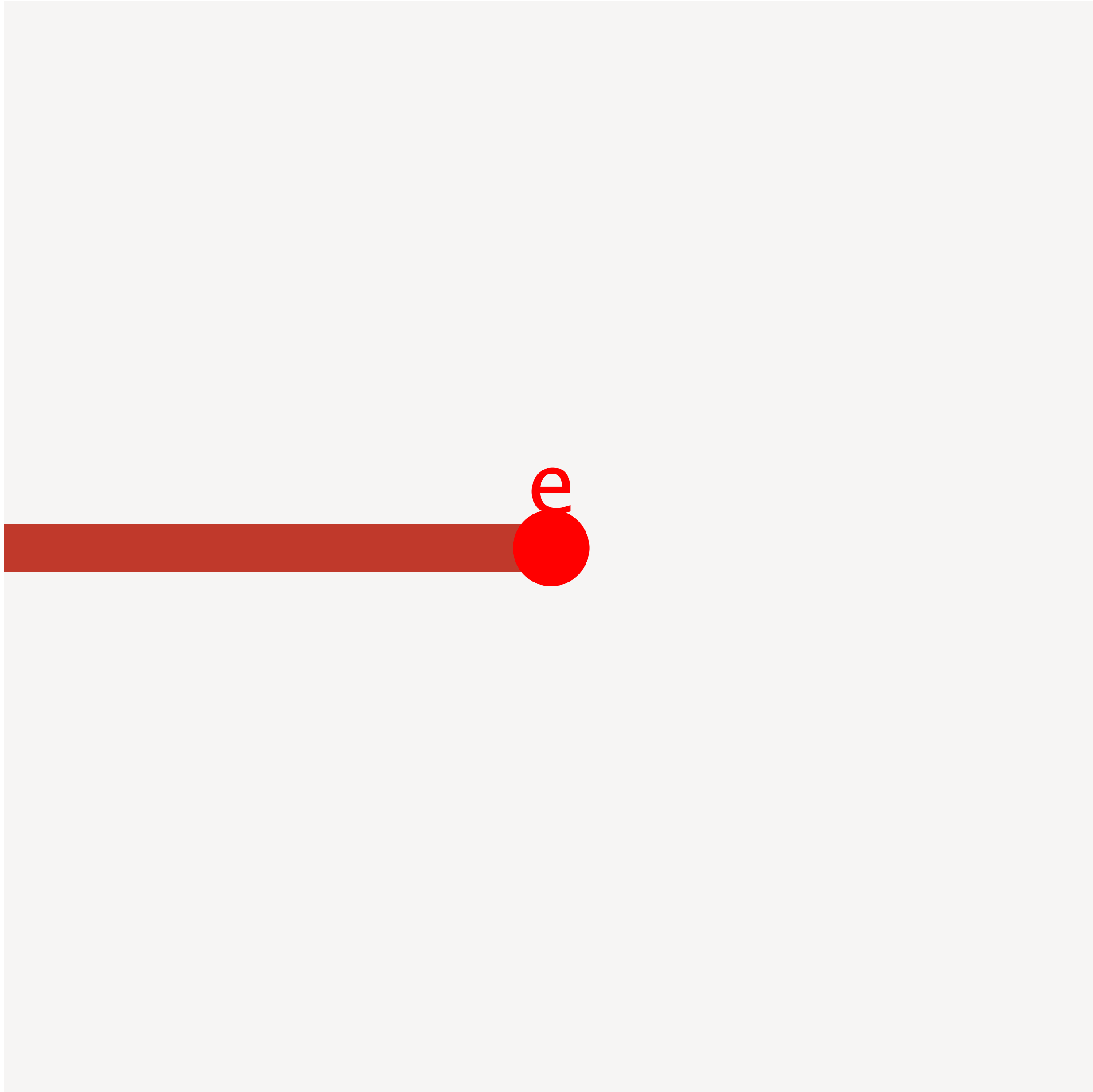
Example 2.1.8. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n-categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out. The trees given in Examples 2.4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

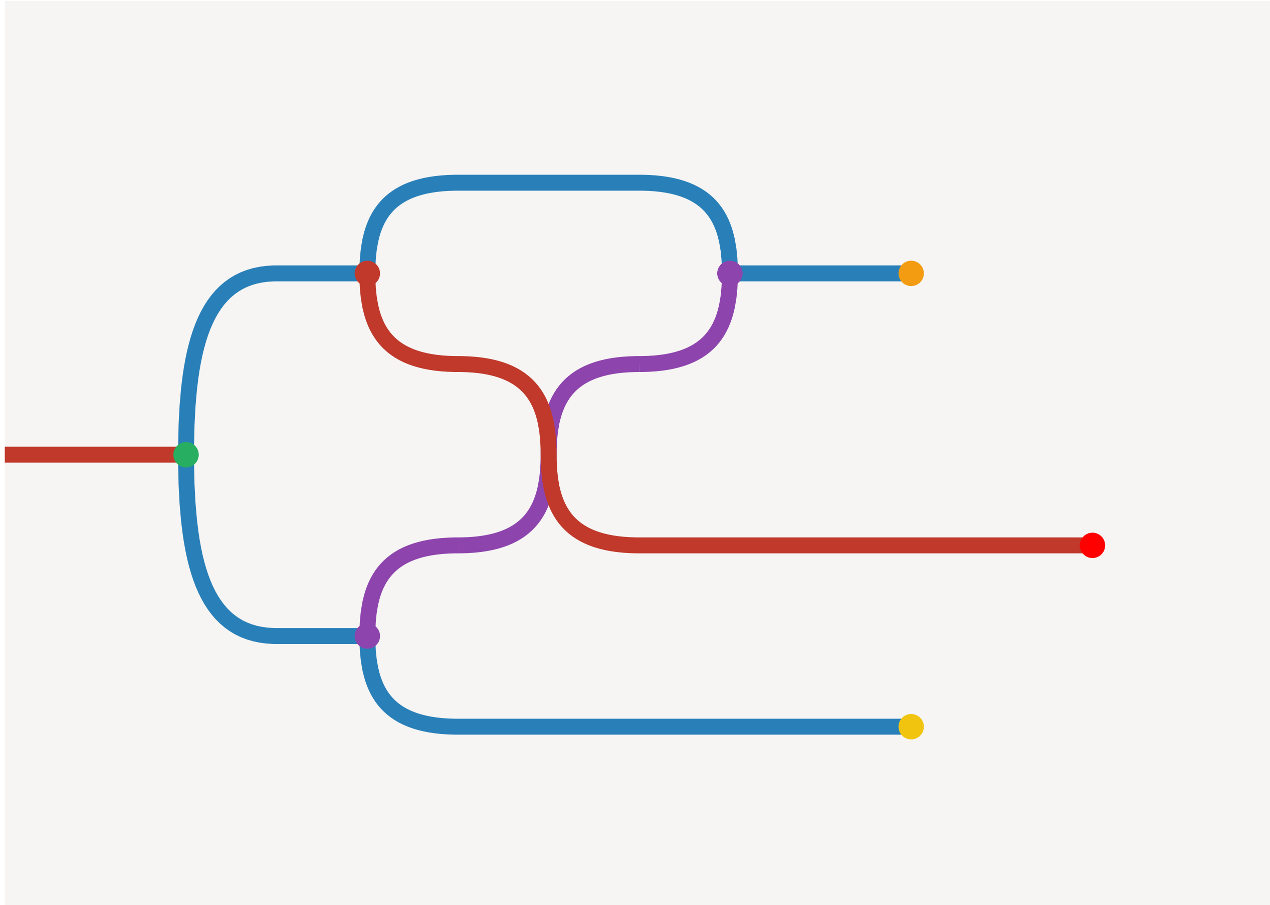
$$placeholder$$

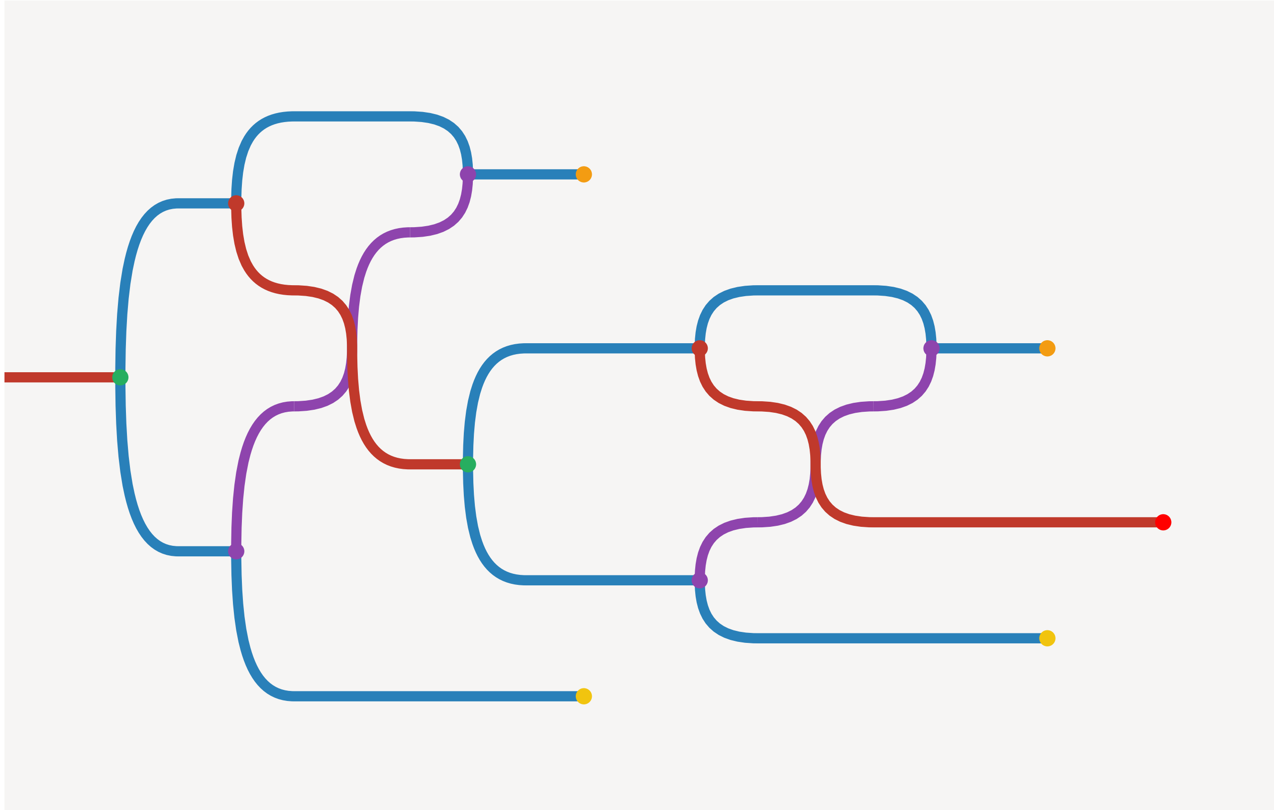
Which we interpret as expressions comprised of the following signature:

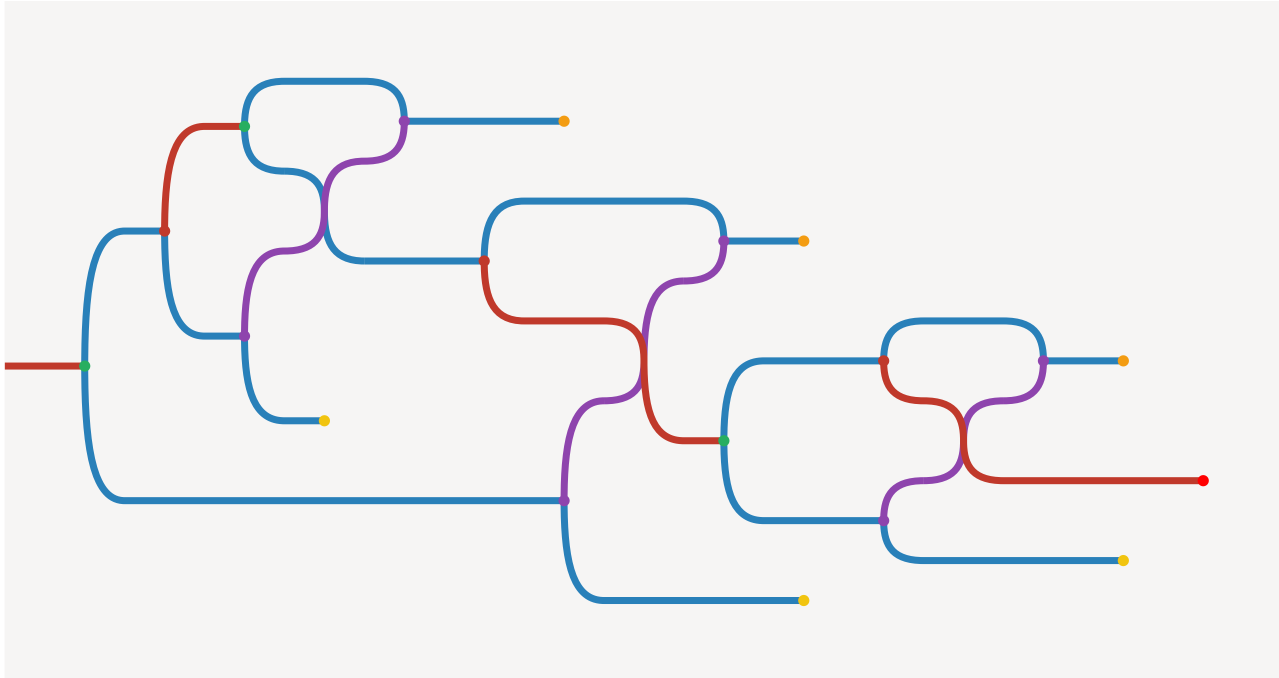
$$placeholder$$

In the process of interpretation, we introduce a link wire-type (in purple), and include directed link generation and elimination morphisms for the T wire-type (in blue). A necessary step in the process of interpretation (which for us involves taking a Poincaré dual to interpret nodes as wires) is a typing assignment of the tree-branches connected to terminal nodes, which we have opted to read as sharing a T-type for minimality, though we could just as well have introduced a separate label-type wire.


e







N.B. In practice when using `homotopy.io` for the symmetric monoidal setting, it is simpler to suspend symmetric monoidal signatures to begin at 4-cells rather than 3-cells. The reason for this is that under- and over-braids still exist in the symmetric monoidal setting, and while sequentially composed braids are homotopically equivalent to the pair of identities, they are not uniquely so, thus these homotopies must be input manually. By beginning at 4-cells (or higher, due to the stabilisation hypothesis CITE), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

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

Definition 2.1.9 (Tree Adjoining Grammars with local constraints and links in `homotopy.io`)**.** A *Linguistic Tree Adjoining Grammar* is given by the following data:

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

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

- $\mathcal{I}$ is a nonempty set of *initial* constrained-linked-trees.
- $\mathcal{A}$ is a nonempty set of *auxiliary* constrained-linked-trees.
- $\mathfrak{S}$ is a set of sets of *select* auxiliary trees.
- $\square, \lozenge$ are fresh symbols. $\square$ marks *obligatory adjoins*, and $\lozenge$ 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 $\mathcal{N} \times \mathfrak{S} \times \{\square, \lozenge\} \times \{*, \bar{*}\}$, and each leaf is an element of $\mathcal{N} \times \mathfrak{S} \times \{\square, \lozenge\} \times \{*, \bar{*}\} \cup \Sigma$. 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.

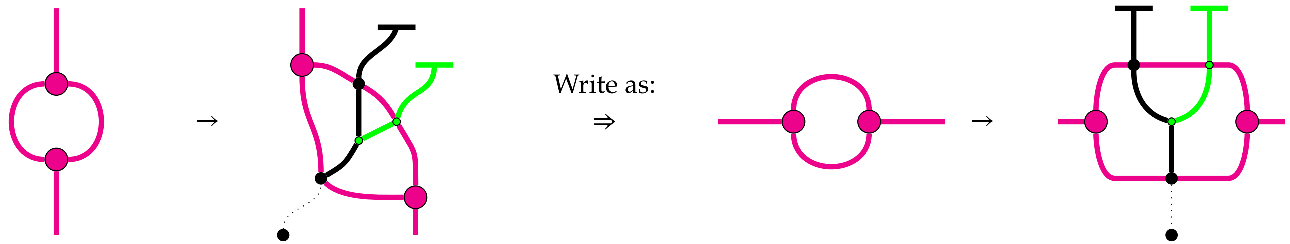
2.2 A generative grammar for text circuits

2.2.1 A circuit-growing grammar

There are many different ways to write an n-categorical signature that generates circuits. Mostly as an illustration of expressive capacity, I will provide a signature where the terms "surface" and "deep" structure are taken literally as metaphors; the generative grammar will grow a line of words in (roughly) syntactic order, and like mushrooms on soil, the circuits will appear as the mycelium underneath words.

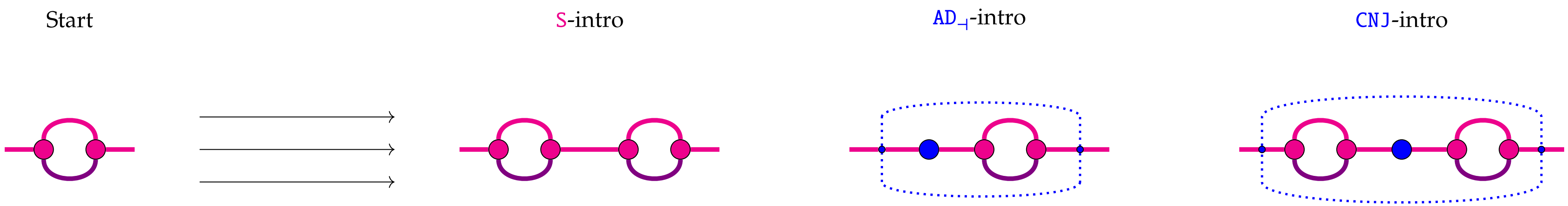
Simplifications: Propositions only, no determiners, only one tense, no morphological agreement between nouns and their verbs and referring pronouns, and we assume that adverbs, adverbial adjunctions, and adjectives stack indefinitely and without further order requirements; e.g. `Yesterday yesterday yesterday Alice happily secretly finds red big toy shiny car that he gives to Bob.` we consider grammatical enough. For now, we consider only the case where adjectives and adverbs appear before their respective noun or verb. Note that all of these limitations can principle be overcome by the techniques we developed in Section 2.1.5 for restricted tree-adjoining and links.

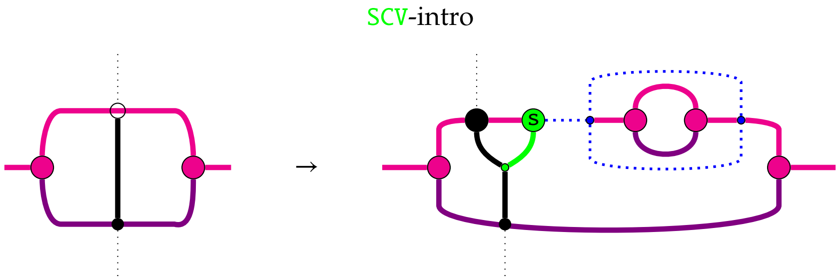
How to read the following diagrams: We work in a dimension where wires behave symmetric monoidally by homotopy, and the signature still works if interpreted in a compact closed setting. We start each derivation with a pink sentence bubble, which we depict for aesthetic purposes as horizontal, which amounts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



2.2.2 Sentences

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

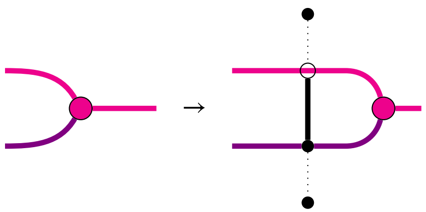




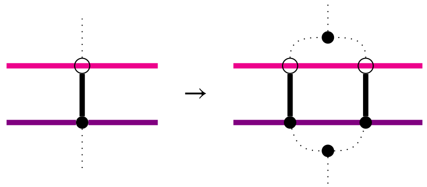
2.2.3 *Simple sentences*

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

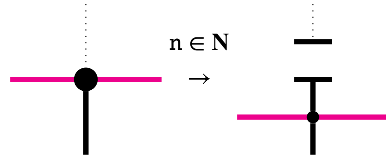
`N`-intro



`N`-split

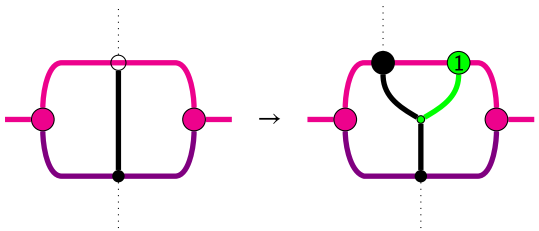


`N`-label

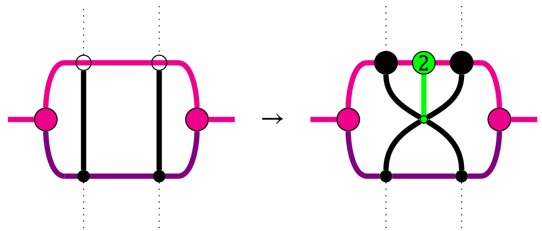


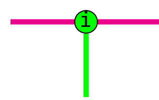
Nouns require verbs in order to be saturated. From left-to-right; if there is precisely one unlabelled noun, we may introduce an unlabelled intransitive verb and saturate the noun so that it is now ready to grow a label; or if there are two unlabelled nouns, we may introduce an unlabelled transitive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

`IV`-intro



`TV`-intro

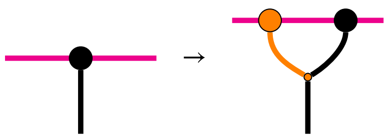




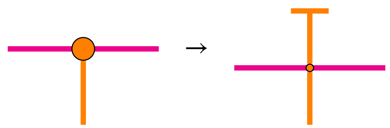
2.2.4 *Modifiers*

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

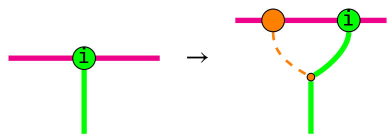
`ADJ`-intro



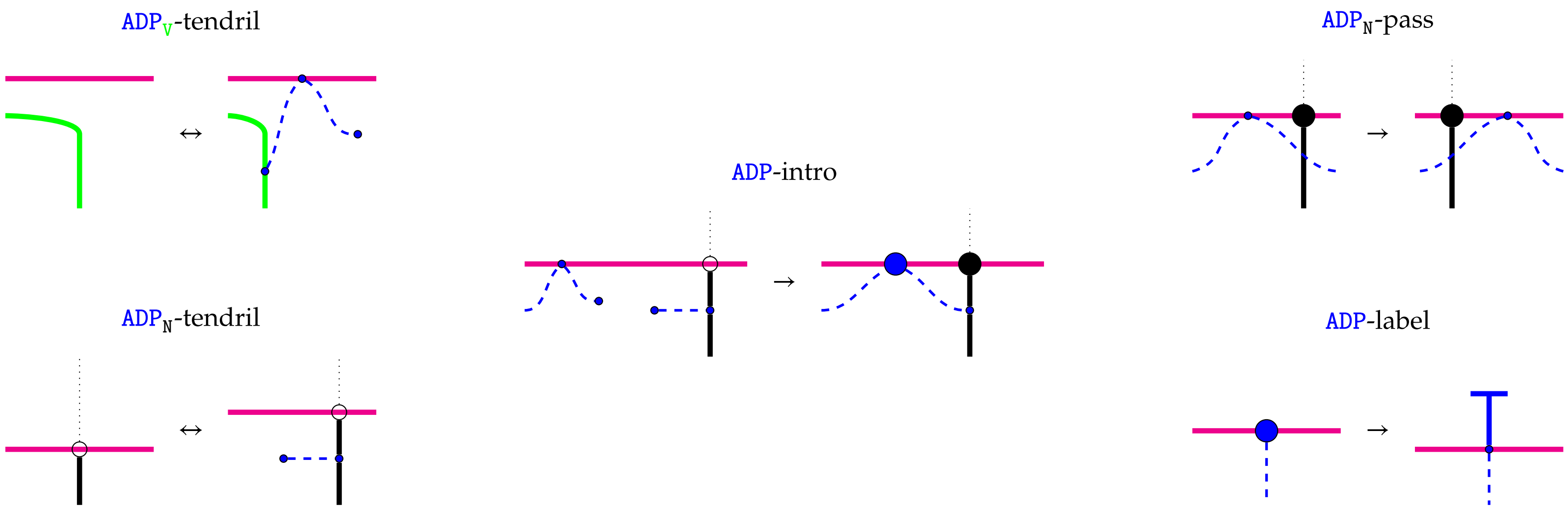
`ADJ`-label



`ADV`-intro



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



2.2.5 *Rewriting to circuit-form*

Resolving references

Connecting circuits

Example 2.2.1.

2.2.6 *Extensions I: relative and reflexive pronouns*

Subject relative pronouns

Example 2.2.2.

Object relative pronouns

Example 2.2.3.

Reflexive pronouns

Example 2.2.4.

2.2.7 Extensions II: grammar equations

Attributive vs. predicative modifiers

Example 2.2.5.

Copulas

Example 2.2.6.

Possessive pronouns

Example 2.2.7.

2.2.8 Extensions III: higher-order modifiers

Intensifiers

Example 2.2.8.

Comparatives

Example 2.2.9.

2.2.9 Equivalence to internal wirings

2.2.10 Text circuit theorem

2.2.11 Related work

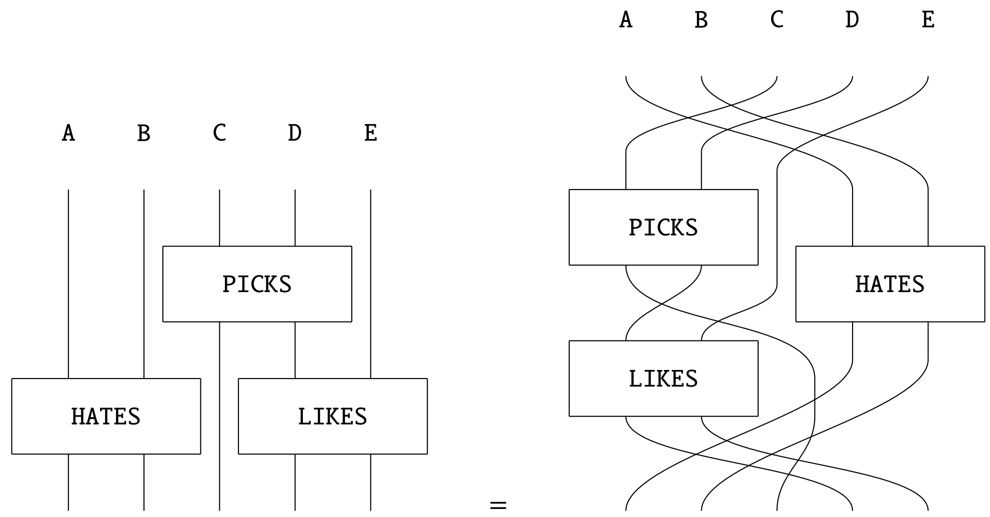
2.3 Text circuits: details, demos, developments

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. While there is no improvement in capabilities by solving bAbi using text circuits, the bAbi tasks have been used as a dataset to learn word gates from data, in a conceptually compliant and compositional manner. Surprisingly, despite the low-data, low-compute regime, the tasks for which the current theory has the expressive capacity to cover are solved better by text circuits than by LSTMs; a proof-of-concept that with the aid of appropriate mathematics, not only might fundamental linguistic considerations help rather than hinder NLP, but also that explainability and capability are not mutually exclusive. Experimental details are elaborated in a forthcoming report CITE . While there are expressivity constraints contingent on theoretical development, this price buys a good amount of flexibility within the theoretically established domain: text circuits leave room for both learning-from-data and "hand-coded" logical constraints expressed process-theoretically, and naturally accommodate previously computed vector embeddings of words.

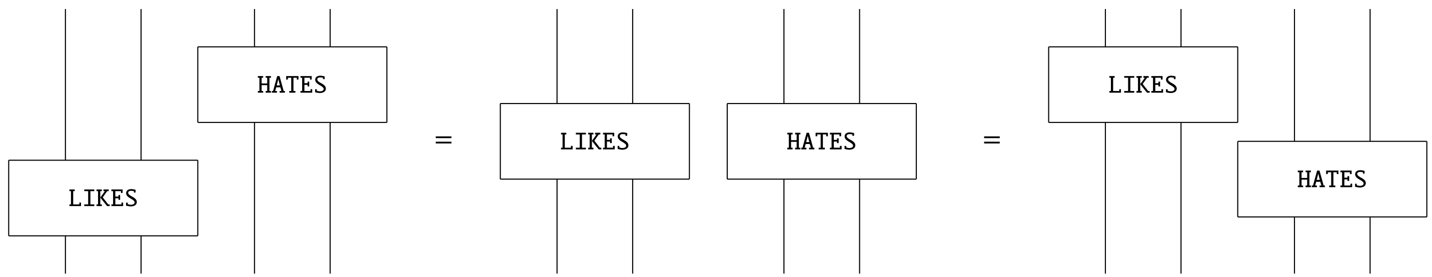
In practice, the process of obtaining transparently computable text goes through two phases. First, one has to obtain text circuits from text, which is conceptually simple: typelogical parsers for sentences can be modified to produce circuit-components rather than trees, and a separate pronomial resolution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

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

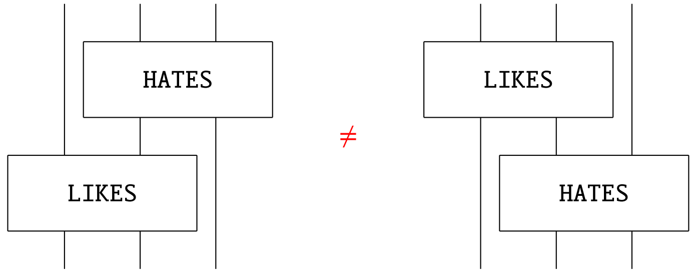
Convention 2.3.1. Sometimes we allow wires to twist past each other, and we consider two circuits the same if their gate-connectivity is the same:



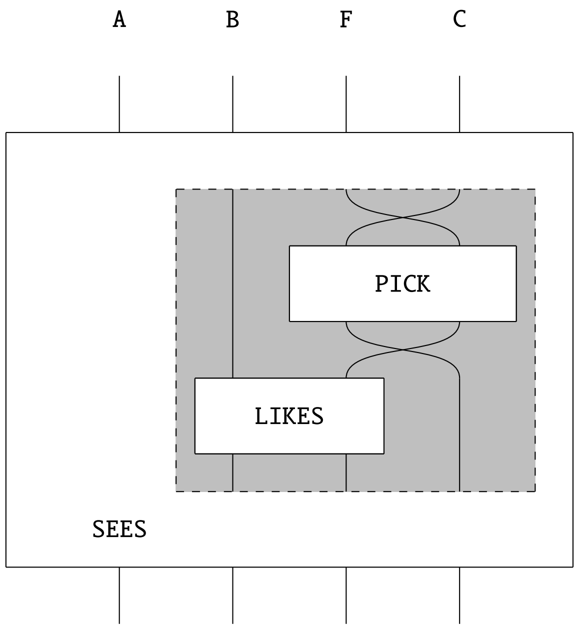
Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel:



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



Example 2.3.2. The sentence ALICE SEES BOB LIKES FLOWERS THAT CLAIRE PICKS can intuitively be given the following text circuit:



3

A palette for toy models

3.1 Continuous Relations

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. 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.

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 – a relative newcomer in terms of mathematical technique – we are able to find and make use of the rich structure of **ContRel** with a different toolkit.

In the end, we want to formalise doodles, so perhaps there is some virtue in proceeding by elementary means.

Recall that functions are relations, and the inverse image used in the definition of continuous maps is equivalent to the relational converse when functions are viewed as relations. So we can naïvely extend the notion of continuous maps to continuous relations between topological spaces.

Definition 3.1.4 (Continuous Relation)**.** A continuous relation $R : (X, \tau) \to (Y, \sigma)$ is a relation $R : X \to Y$ such that

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

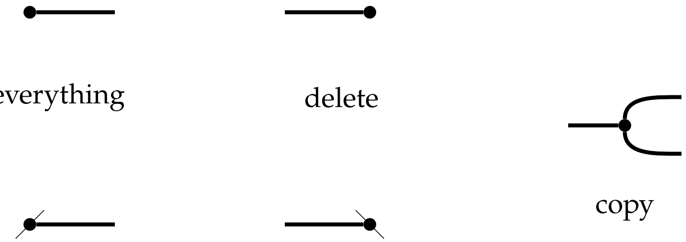
where $\dagger$ denotes the relational converse.

Notation 3.1.5. For shorthand, we denote the topology (X, τ) as X^{τ}. As special cases, we denote the discrete topology on X as $X^{\star}$, and the indiscrete topology X°.

The symmetric monoidal structure is that of product topologies on objects, and products of relations on morphisms.

inuous

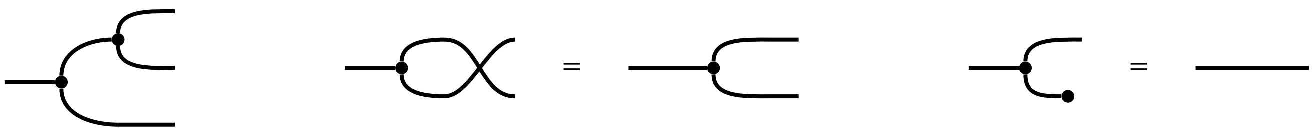
NS FOR ANY X^{τ}:



everything delete copy

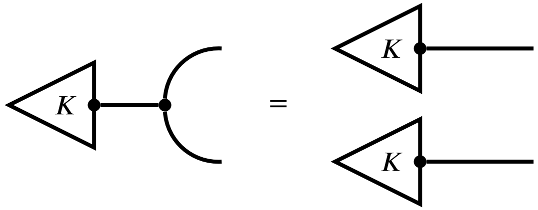
nothing (state) nothing (test)

NG EQUALITIES:

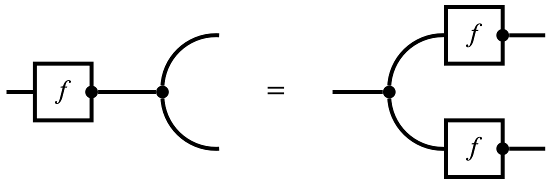


ivity cocommutativity counitality

DISTINGUISH THE DETERMINISTIC MAPS – POINTS AND FUNCTIONS – WHICH WE NOTATE WITH AN EXTRA DOT.

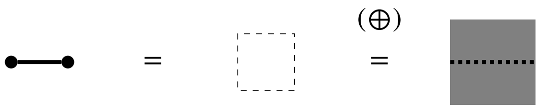


point

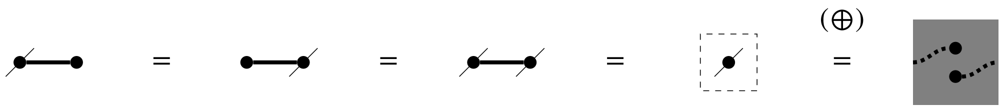


function

Everything, delete, nothing-states and nothing-tests combine to give two numbers, one and zero. There are extra expressions in grey squares above: they anticipate the tape-diagrams we will later use to graphically express another monoidal product of **ContRel**, the direct sum $\oplus$.

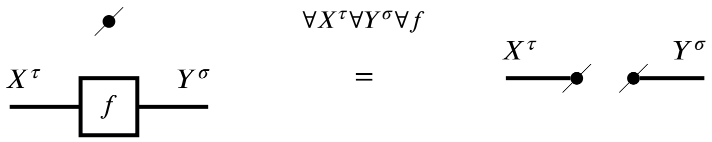


(b)one



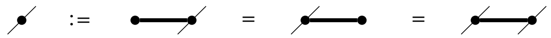
zero

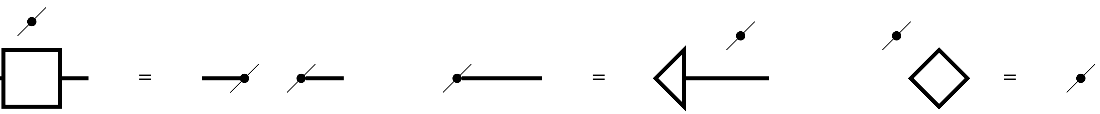
Zero scalars turn entire diagrams into zero morphisms. There is a zero-morphism for every input-output pair of objects in **ContRel**.



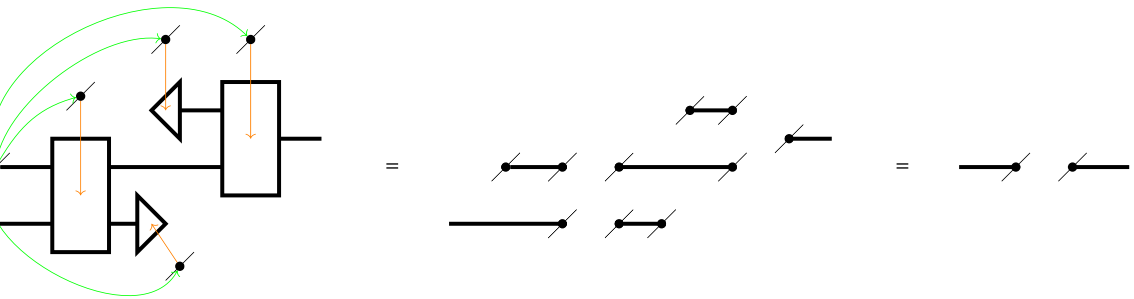
zero

ase in **Rel**, **Vect**, and any category with biproducts and zero-objects []) is that the zero scalar is an absorbing element of multipli-
ess by a zero scalar sends it to its corresponding zero-morphism.





rs in a diagram, it spawns zero scalars which infect all other processes, turning them all into zero-processes. The same holds for
kes copies of itself to infect all other processes.

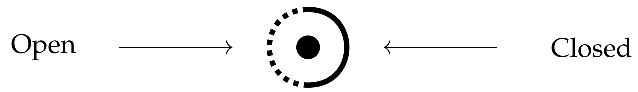


3.3 *Continuous Relations by examples*

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.

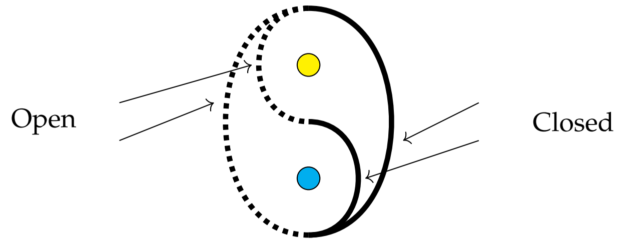
The **singleton space** consists of a single point which is both open and closed. We denote this space $\bullet$. Concretely, the underlying set and topology is

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

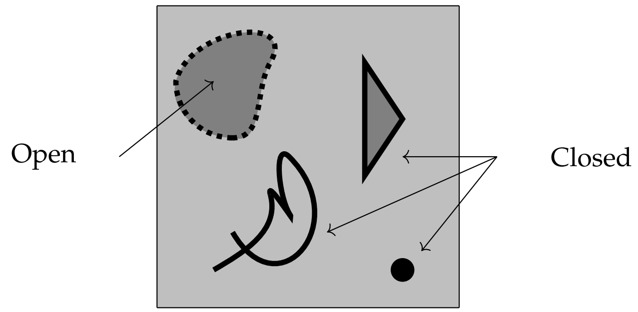


The **Sierpiński space** consists of two points, one of which (in yellow) is open, and the other (in cyan) is closed. We denote this space $\mathcal{S}$. Concretely, the underlying set and topology is:

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



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.

WHICH RELATIONS $X^\tau \to Y^\sigma$ ARE ALWAYS CONTINUOUS?

THE EMPTY RELATION IS ALWAYS CONTINUOUS.

Proposition 3.3.6. *Proof.* The preimage of the empty relation is always $\varnothing$, which is open by definition. □

FULL RELATIONS ARE ALWAYS CONTINUOUS

Proposition 3.3.8. *Proof.* The preimage of any subset of Y – open or not – under the full relation is the whole of X, which is open by definition. □

FULL RELATIONS RESTRICTED TO OPEN SETS IN THE DOMAIN ARE CONTINUOUS.

Proposition 3.3.9. Given an open $U \subseteq X^\tau$, and an arbitrary subset $K \subset Y^\sigma$, the relation $U \times K \subseteq X \times Y$ is open.

Proof. Consider an arbitrary open set $V \in \sigma$. Either V and K are disjoint, or they overlap. If they are disjoint, the preimage of V is $\varnothing$, which is open. If they overlap, the preimage of V is U, which is open. □

CONTINUOUS FUNCTIONS ARE ALWAYS CONTINUOUS.

Proposition 3.3.10. If $f : X^\tau \to Y^\sigma$ is a continuous function, then it is also a continuous relation.

Proof. Functions are special cases of relations. The relational converse of a function viewed in this way is the same thing as the preimage. □

THE IDENTITY RELATION IS ALWAYS CONTINUOUS. The identity relation is also the "trivial" continuous map from a space to itself, so this also follows from Proposition 3.3.10.

Proposition 3.3.12. *Proof.* The preimage of any open set under the identity relation is itself, which is open by assumption. □

GIVEN TWO CONTINUOUS RELATIONS $R, S : X^\tau \to Y^\sigma$, HOW CAN WE COMBINE THEM?

Proposition 3.3.14. 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 3.3.15. 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

Definition 3.3.17 (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 3.3.18 (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 3.3.19 (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 3.3.20. 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 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 3.3.19. By Lemma 3.3.18, every $R \cap F$ term is a partial function, and by Corollary 3.3.15, continuous. □

$\mathcal{S} \to \mathcal{S}$: We can use Proposition 3.3.20 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 $[0,1]$.

Any painting is a continuous relation $[0,1] \to \blacksquare$. By colour-coding $[0,1]$ and controlling brushstrokes, we can do quite a lot. Now we would like to develop the abstract machinery required to *formally* paint pictures with words.

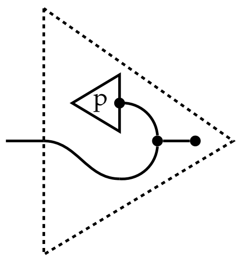
3.4 *Populating space with shapes using sticky spiders*

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

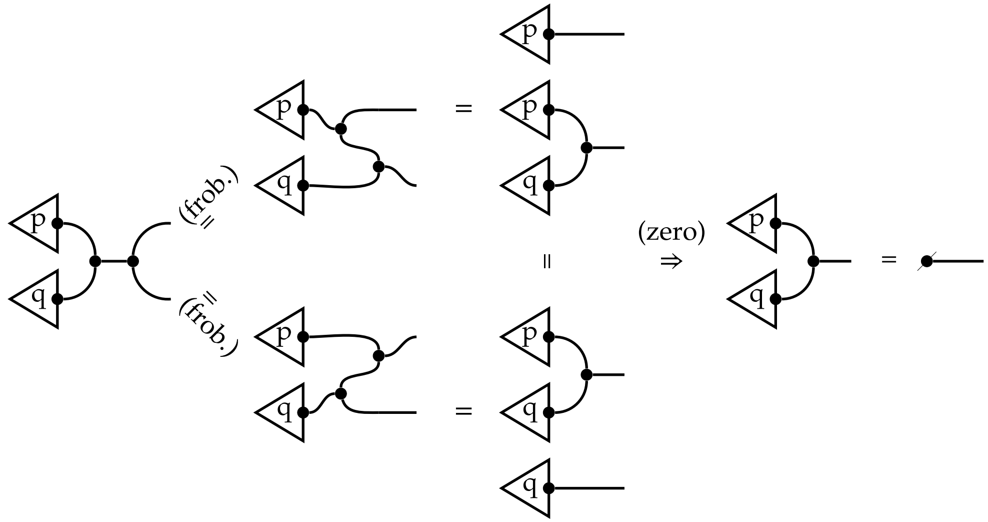
Example 3.4.2 (The copy-compare spiders of **Rel** are not always continuous)**.** The compare map for the Sierpiński space is not continuous, because the preimage of $\{0, 1\}$ is $\{(0, 0), (1, 1)\}$, which is not open in the product space of $\mathcal{S}$ with itself.

Proposition 3.4.3. The copy map is a spider iff the topology is discrete.

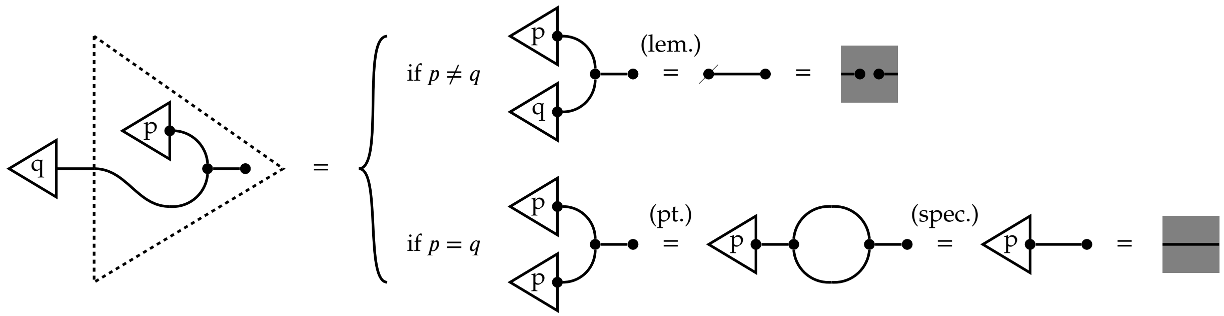
Proof. Discrete topologies inherit the usual copy-compare spiders from **Rel**, so we have to show that when the copy map is a spider, the underlying wire must have a discrete topology. Suppose that some wire has a spider, and construct the following open set using an arbitrary point p:



It will suffice to show that this open set is the singleton $\{p\}$ – when all singletons are open, the topology is discrete. As a lemma, using frobenius rules and the property of zero morphisms, we can show that comparing distinct points $p \neq q$ yields the $\varnothing$ state.



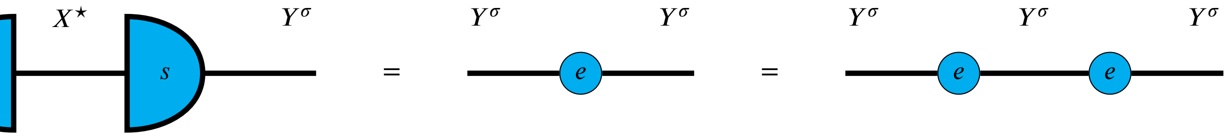
The following case analysis shows that our open set only contains the point p.

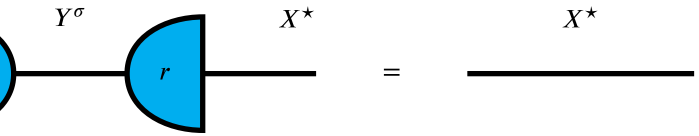


□

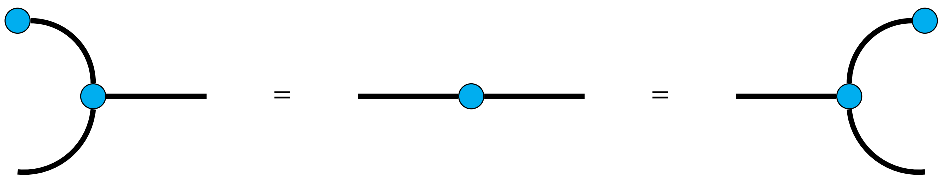
It will be more aesthetic going forward to colour processes and treat the colours as variables instead of labelling them.

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

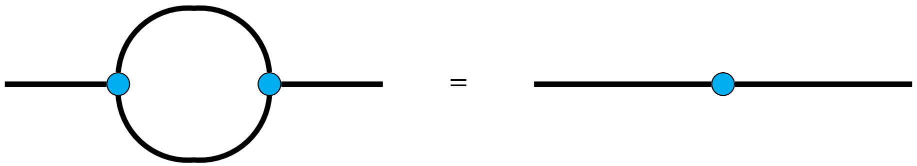




ticky spider (or just an *e*-spider, if we know that *e* is a split idempotent), is a spider *except* every identity wire on any side of an ne idempotent *e*.

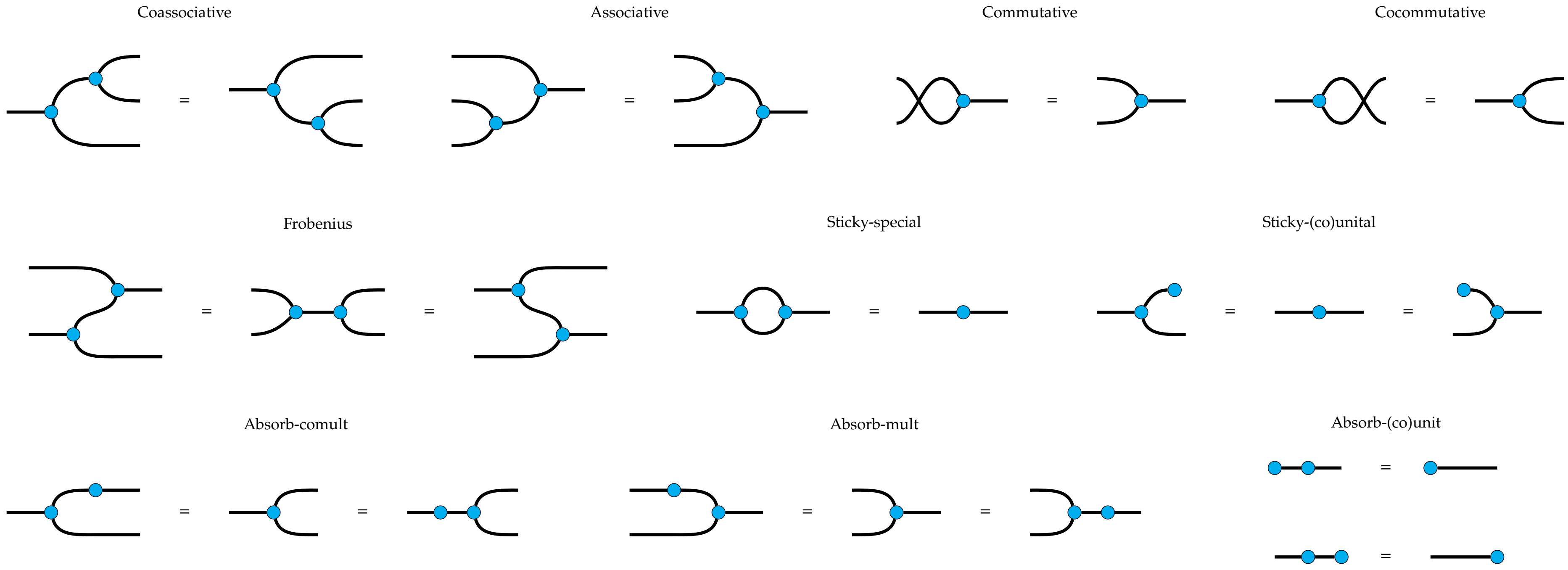


e-(co)unitality



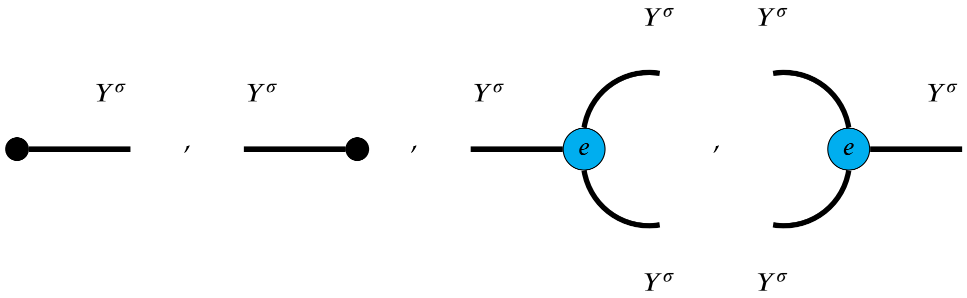
e-special

f a sticky spider is that one can still coalesce all connected spider-bodies together, but the 1-1 spider "sticks around" rather than chieved by the following rules that cohere the idempotent *e* with the (co)unit and (co)multiplications; they are the same as the frobenius algebra with two exceptions. First, where an identity wire appears in an equation, we replace it with an idempotent. omponents freely emit and absorb idempotents. By these rules, the usual proof [] for the normal form of spiders follows, except the idempotent becomes an explicit 1-1 spider, rather than the identity.



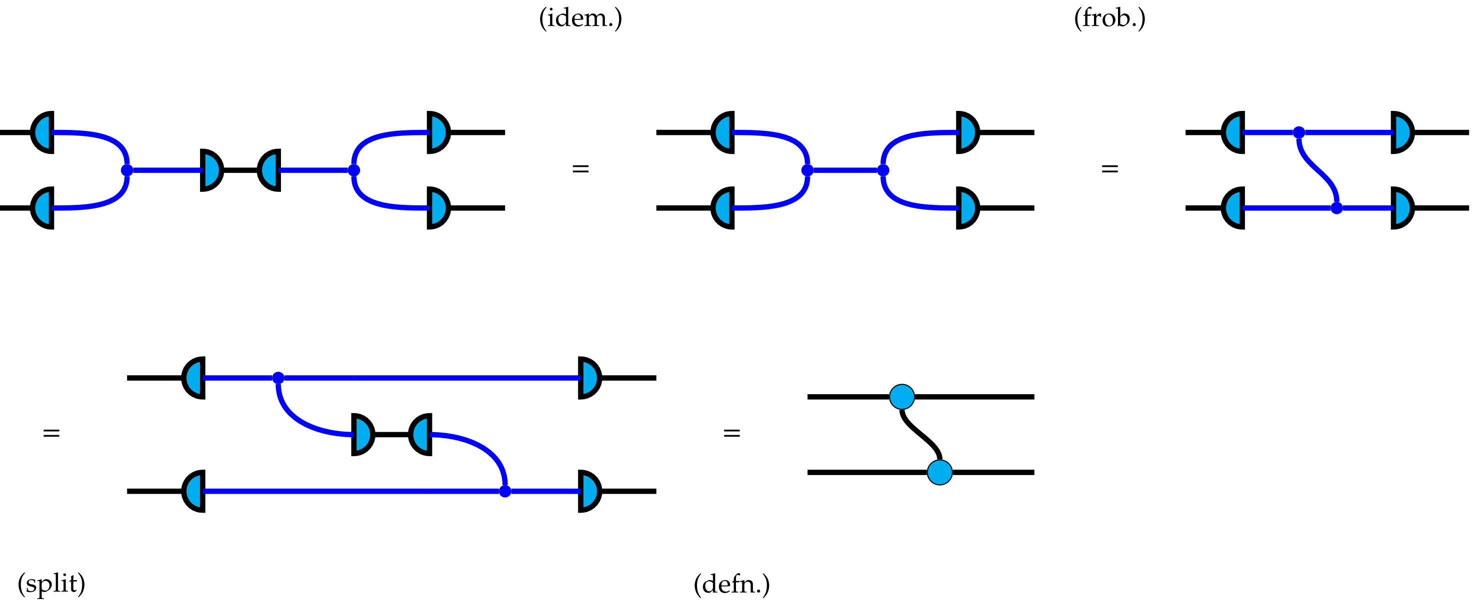
Construction 3.4.6 (Sticky spiders from split idempotents)**.** Given an idempotent $e : Y^\sigma \to Y^\sigma$ that splits through a discrete topology $X^\star$, we construct a new (co)multiplication as follows:

that splits through a discrete topology gives a sticky spider).



is a sticky spider

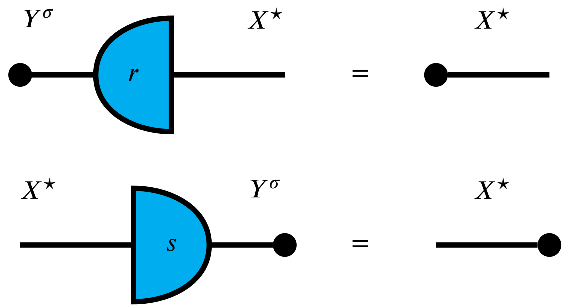
ction satisfies the frobenius rules as follows. We only present one equality; the rest follow the same idea.



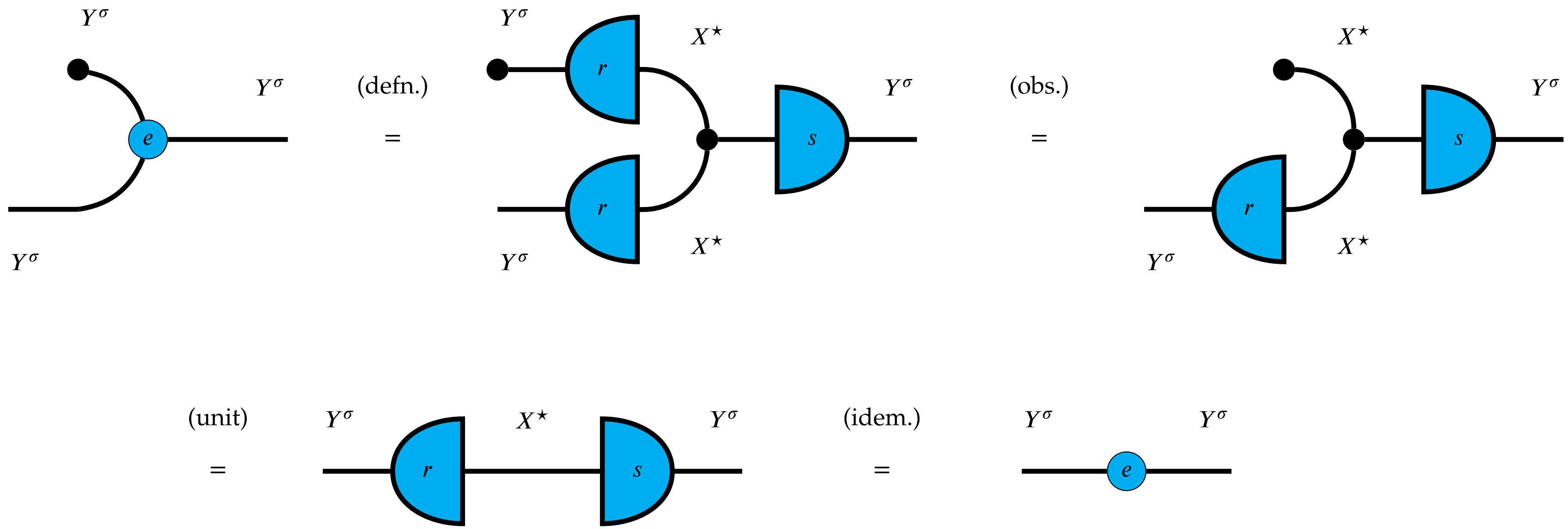
rst observe that since

$$X^\star \xrightarrow{s} Y^\sigma \xrightarrow{r} X^\star = X^\star \xrightarrow{id} X^\star$$

r must have all of $X^\star$ in its image, and s must have all of $X^\star$ in its preimage, so we have the following:



Now we show that e-unitality holds:

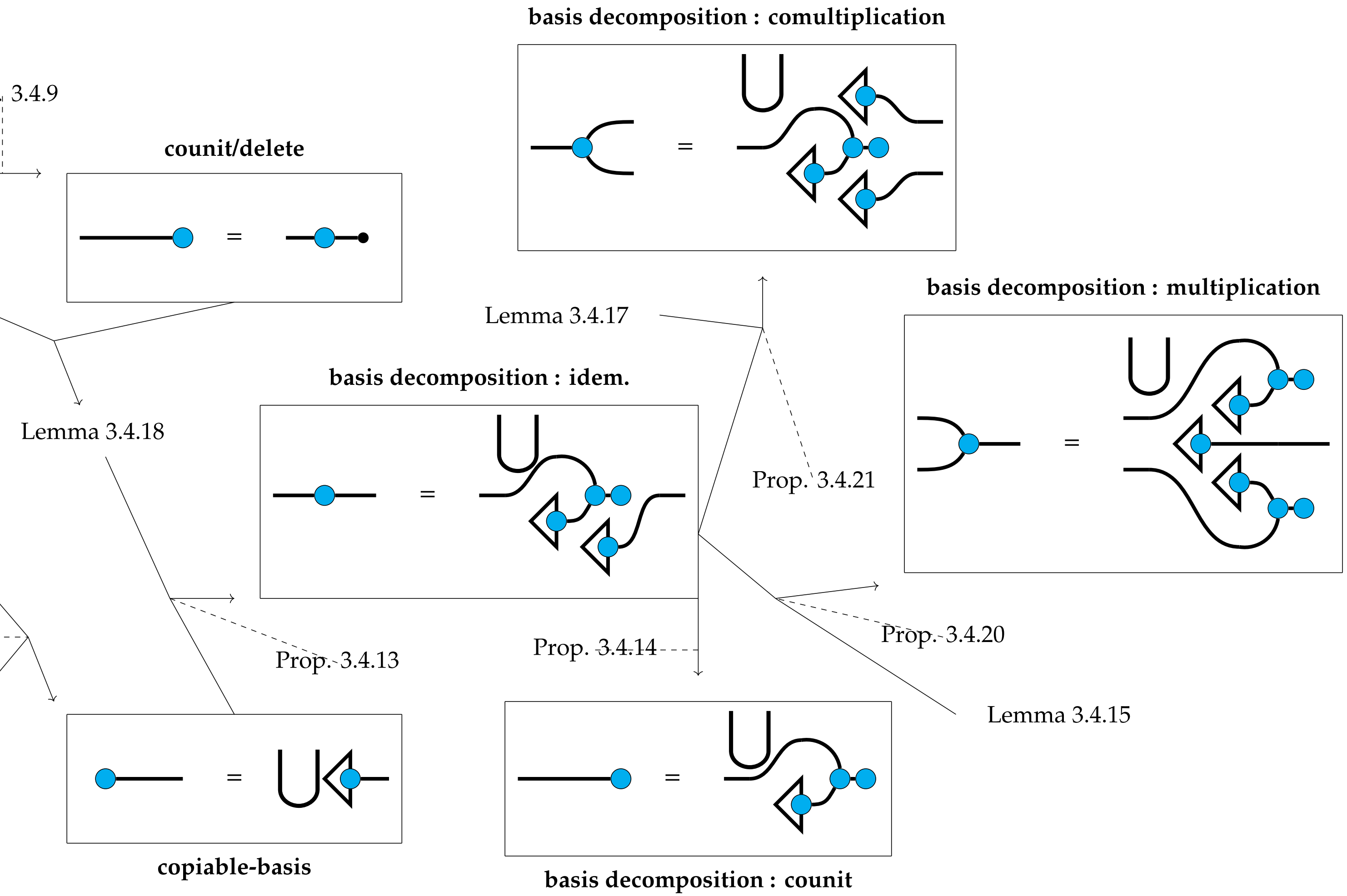


The proofs of e-counitality, and e-speciality follow similarly. □

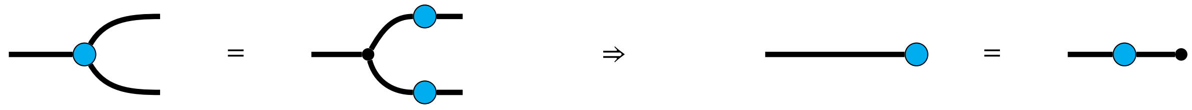
We can prove a partial converse of Proposition 3.4.7: we can identify two diagrammatic equations that tell us precisely when a sticky spider has an idempotent that splits though some discrete topology.

Theorem 3.4.8. A sticky spider has an idempotent that splits through a discrete topology if and only if in addition to the sticky spider equalities, the following rela-

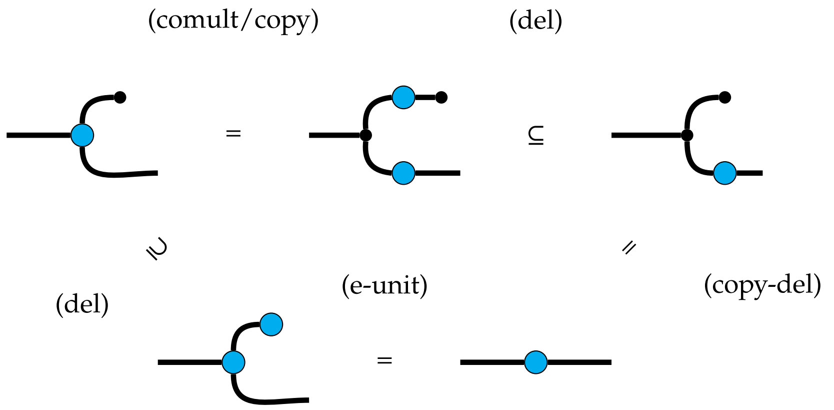
provide a map below of the various lemmas and propositions that will yield the claim.



Proposition 3.4.9 (comult/copy implies counit/delete)**.**



Proof.

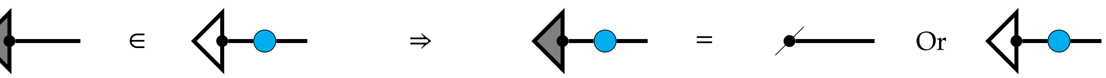


So:

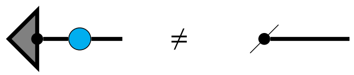
So:

∎

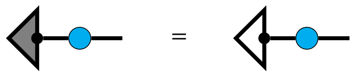
sider the set $e(\{x\})$ obtained by applying the idempotent e to a singleton $\{x\}$, and take an arbitrary element $y \in e(x)$ of this set.
iagrammatically:



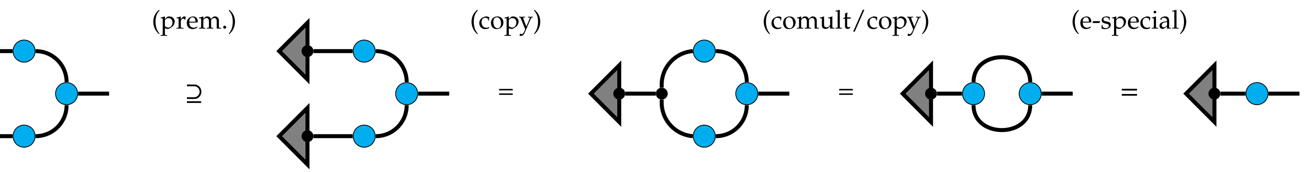
e



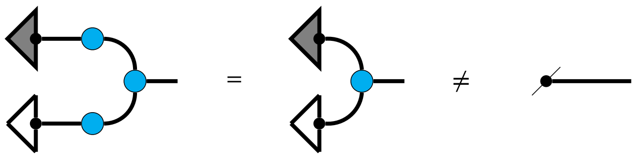
ve seek:



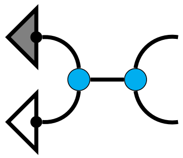
ng inclusion:

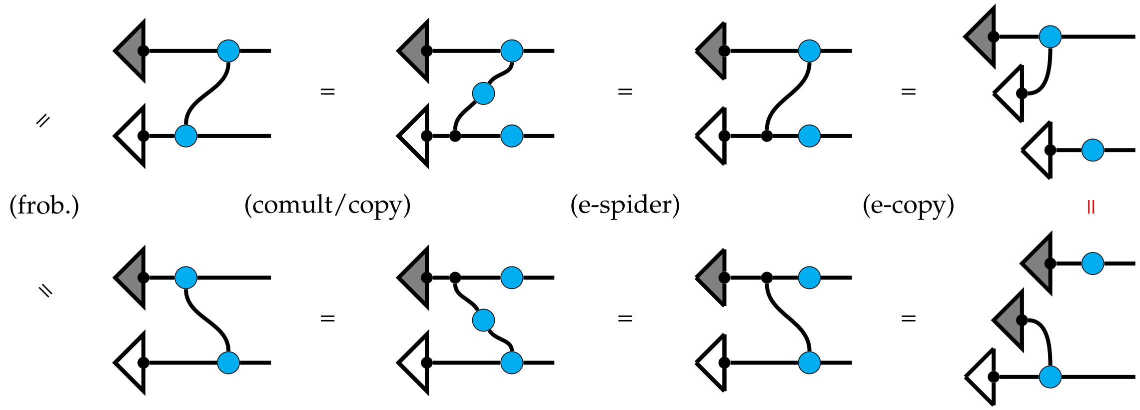


e:

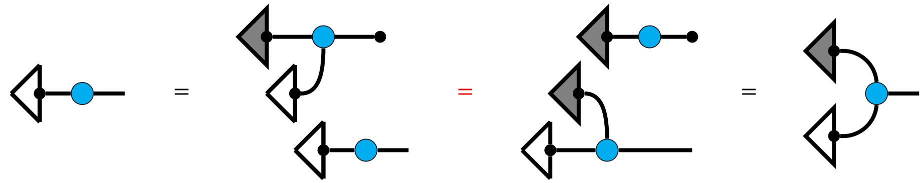


So we have the following equality:

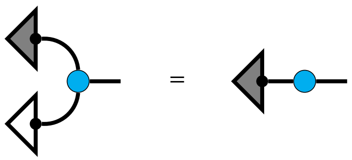




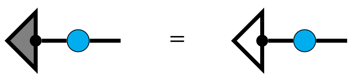
Which implies:



and symmetrically,

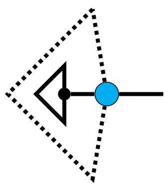


So we have the claim:

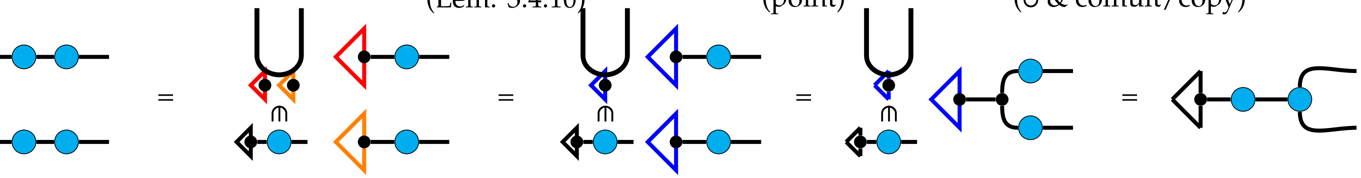


□

-copiable).

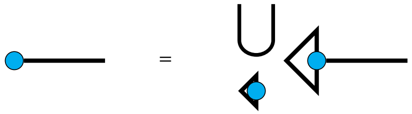


(Lem. 3.4.10) (point) (∪ & comult/copy)



□

Proposition 3.4.12 (The unit is the union of all *e*-copiables)**.**

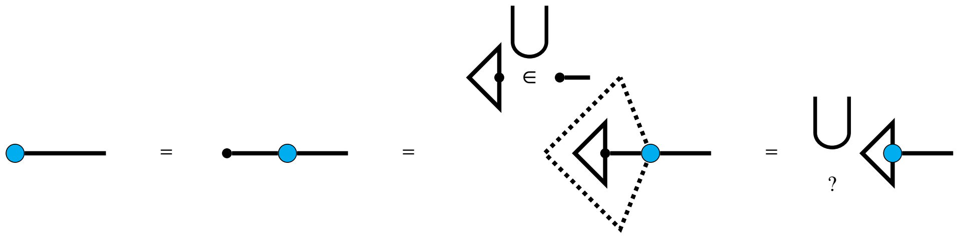


Proof.

The union of *all* *e*-copiables is a subset of the unit.

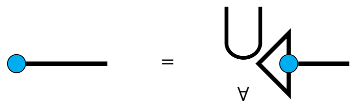
(Lem. 3.4.18) (evr.) (unit/evr.)

The unit is *some* union of *e*-copiables.



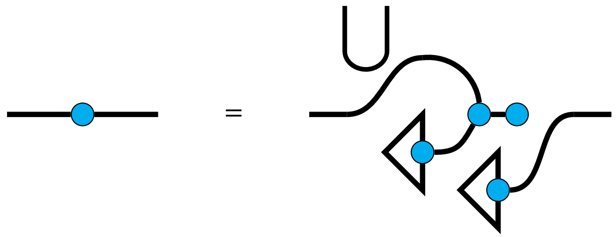
(unit/evr.) (Prop. 3.4.11)

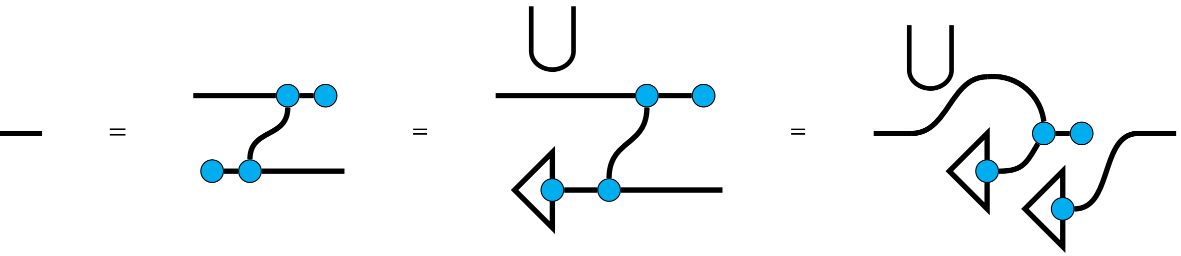
So the containment must be an equality.



□

position of e).

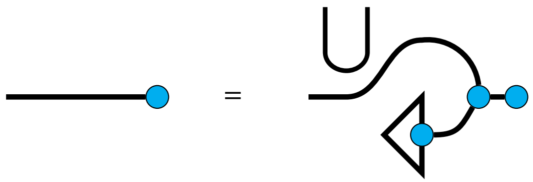




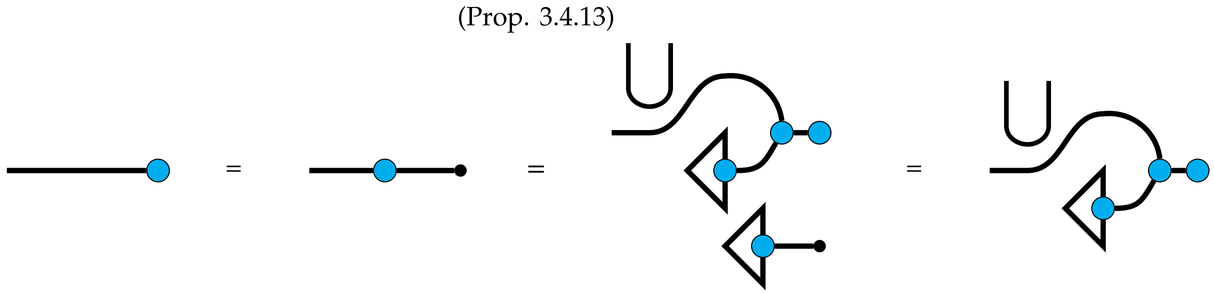
(Prop. 3.4.12) (e-copiable)

□

Proposition 3.4.14 (*e*-copiable decomposition of counit)**.**



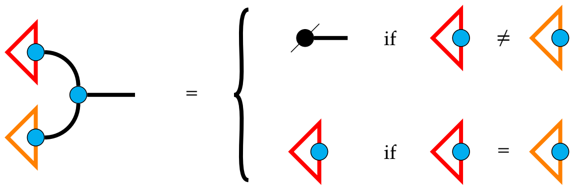
Proof.

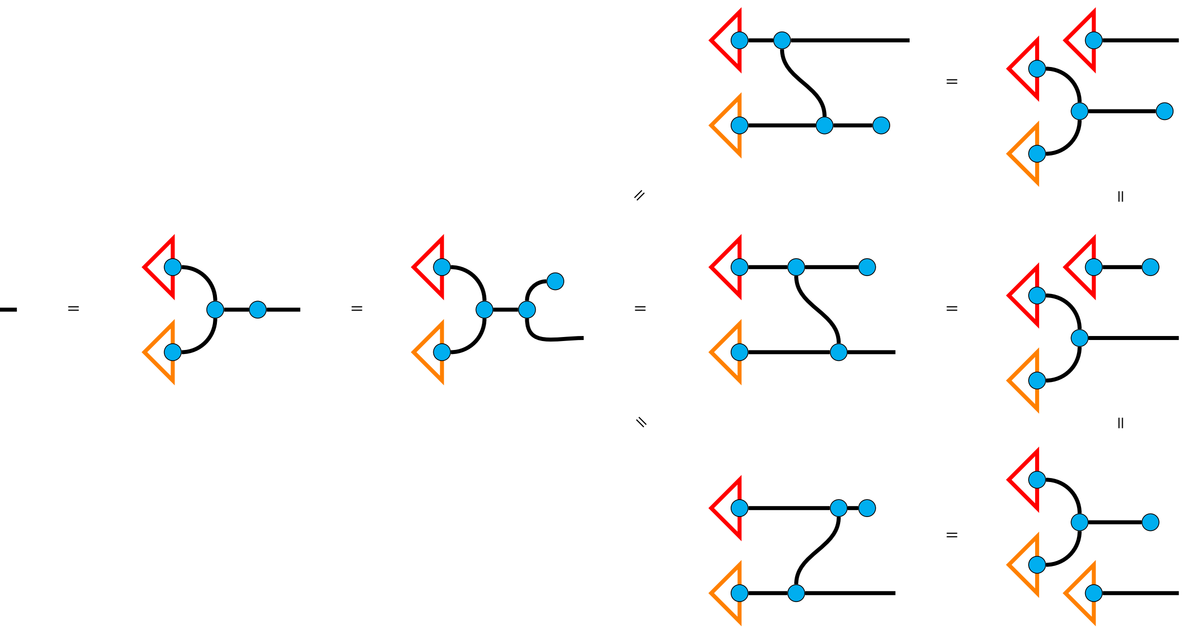


□

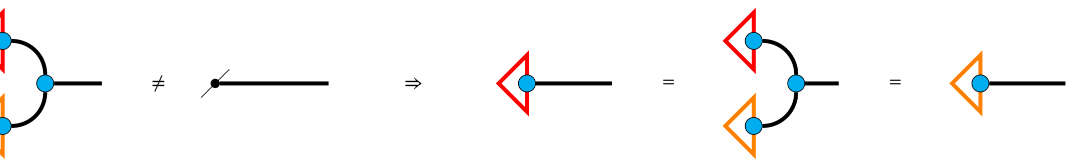
HAVE LIKE AN ORTHONORMAL BASIS, AS THE FOLLOWING LEMMAS SHOW.

onal under multiplication).



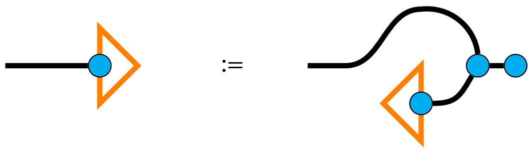


So:

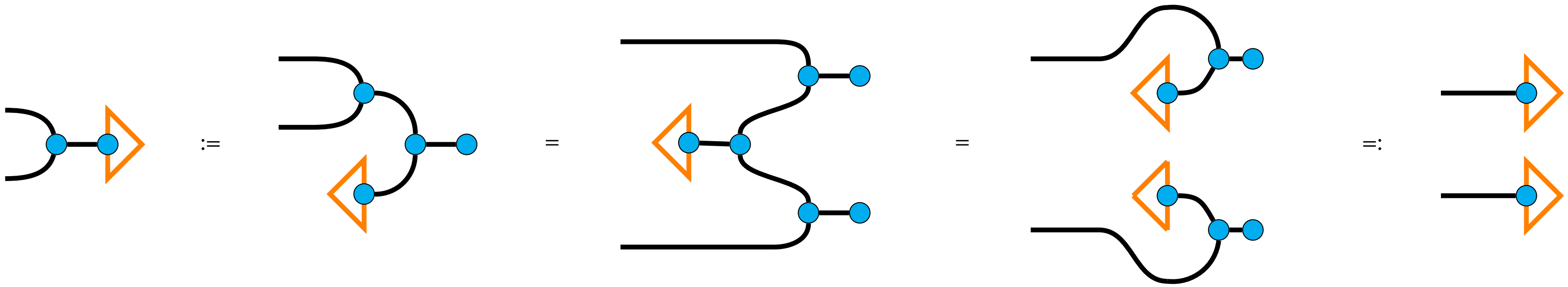


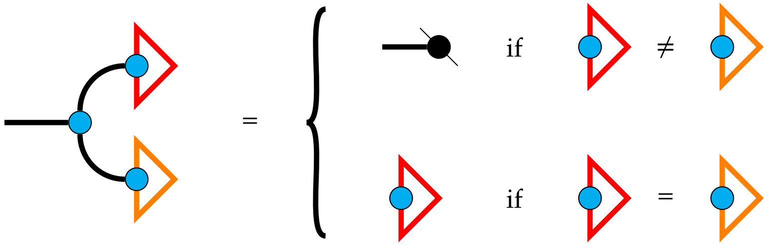
□

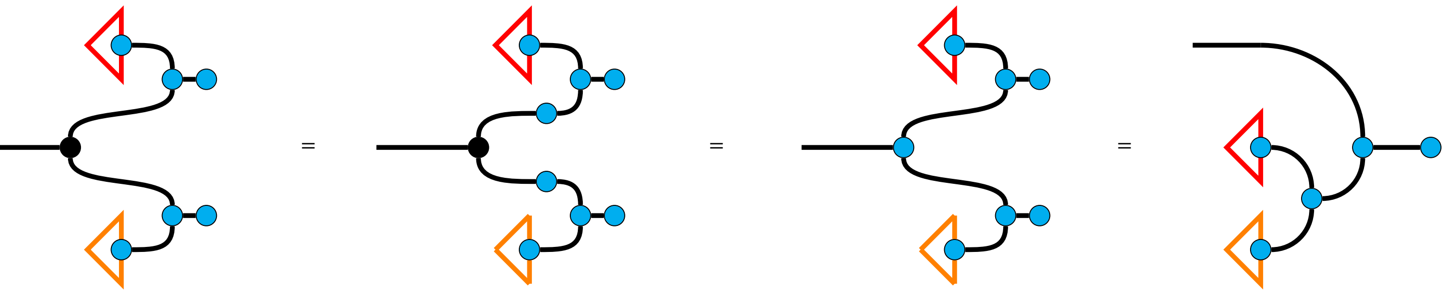
Convention 3.4.16 (Shorthand for the open set associated with an e-copiable)**.** We introduce the following diagrammatic shorthand.



Including the coloured dot is justified, because these open sets are co-copiable with respect to the multiplication of the sticky spider.

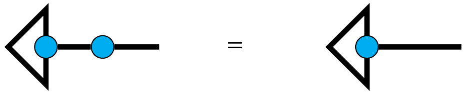






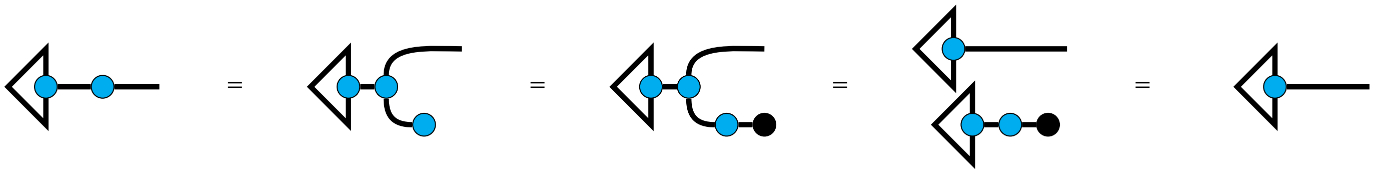
emma 3.4.15 to the final diagram. □

Lemma 3.4.18 (e-copiables are e-fixpoints)**.**



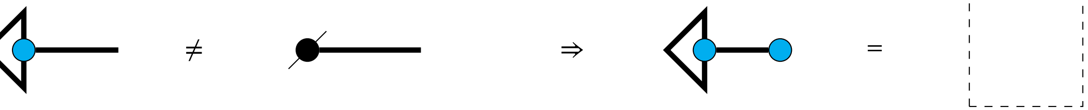
Proof.

(e-counit) (coun/del) (e-copy)

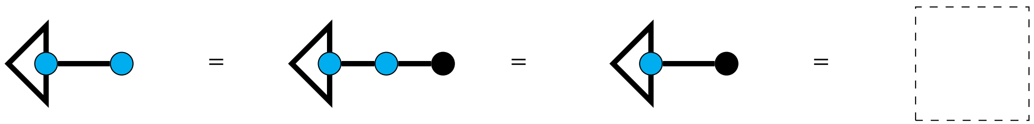


Observe that the final equation of the proof also holds when the initial e-copiable is the empty set. □

l).

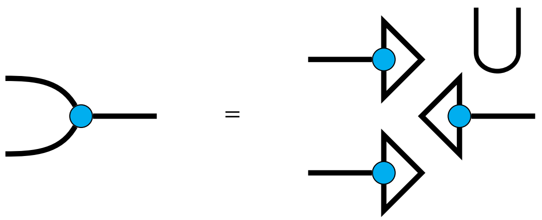


(coun/del) (Lem. 3.4.18) (Prem.)

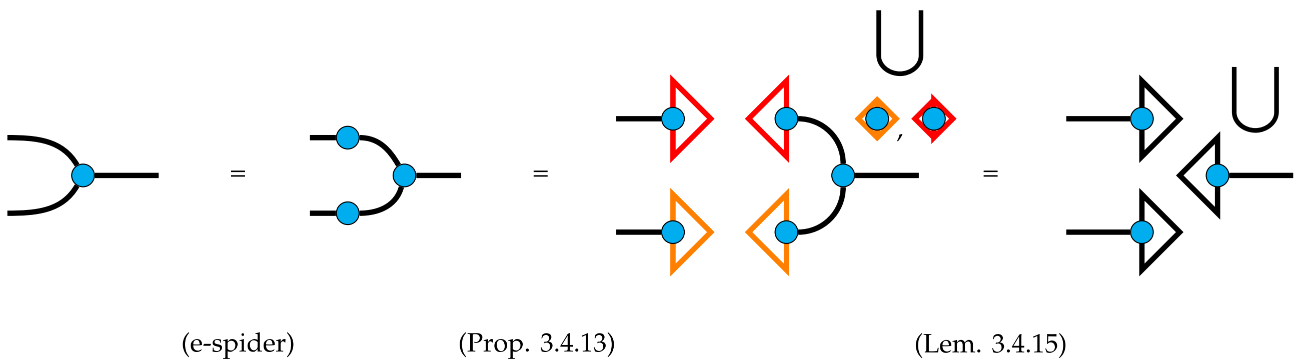


□

Proposition 3.4.20 (*e*-copiable decomposition of multiplication)**.**

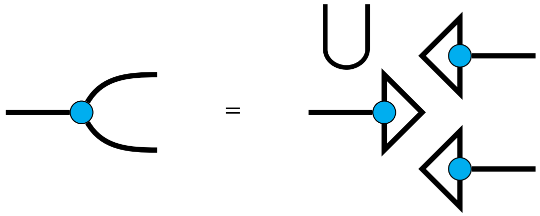


Proof.

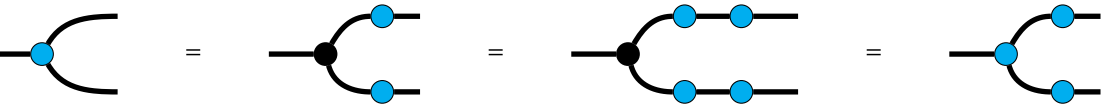


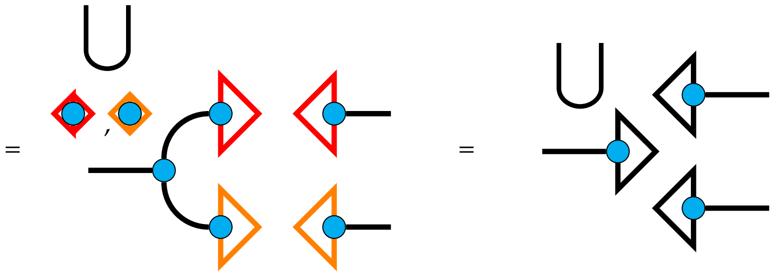
□

position of comultiplication).



(comult/copy) (idem.) (comult/copy)





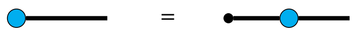
(Prop. 3.4.13) (Lem. 3.4.17)

□

Now we can prove Theorem 3.4.8.

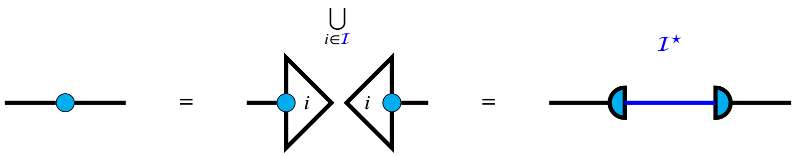
Proof. First a reminder of the claim; we want to show that when given a sticky spider, the following relations hold if and only if the idempotent splits through a discrete topology.

Unit/everything

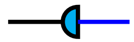


Comult/copy

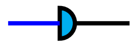
The crucial observation is that the e-copiable decomposition of the idempotent given by Proposition 3.4.13 is equivalent to a split idempotent though the set of e-copiables equipped with discrete topology.



(Prop. X)



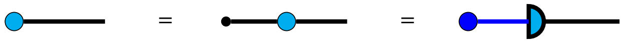
$:= \{(x, i) \mid i \in \mathcal{I}\ ,\ x \in |i>\}$



$:= \{(i, x) \mid i \in \mathcal{I}\ ,\ x \in< i|\}$

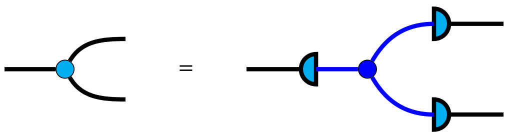
By copiable basis Proposition 3.4.12 and the decompositions Propositions 3.4.14, 3.4.20, 3.4.21, we obtain the only-if direction.

(unit/evr.) (Prop. 3.4.12)

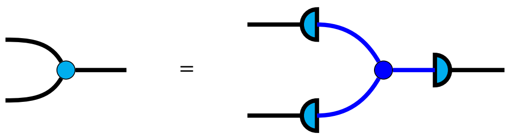


(Prop. 3.4.9) (Prop. 3.4.14)

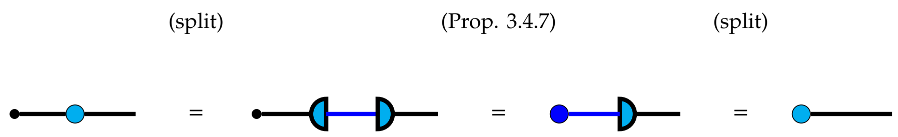
(Prop. 3.4.21)



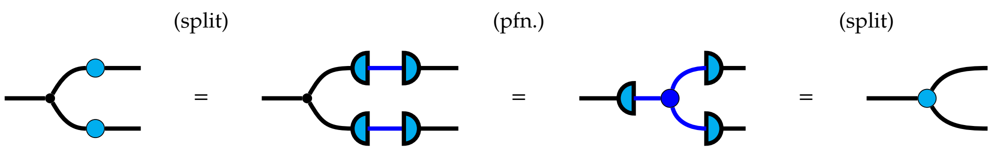
(Prop. 3.4.20)



the unit/everything relation, we have:



serve that for any split idempotent, the retract must be a partial function. To see this, suppose the split idempotent $e = r; s$ is on . Seeking contradiction, if the retract is not a partial function, then there is some point $x \in X$ such that $x \in e(x)$, and the image e point, which we denote and discriminate $a, b \in r(x) \subseteq Y$ and $a \neq b$. Because the composite $s; r = 1_Y$ of the section and retract ection s must be total – i.e. the image $s(X) = Y$. So $x \in s(a) \cap s(b)$. Now we have that $(a, x), (b, x) \in s$, and $(x, a), (x, b) \in r$, therefore radicts that $s; r$ is the identity relation 1_Y.



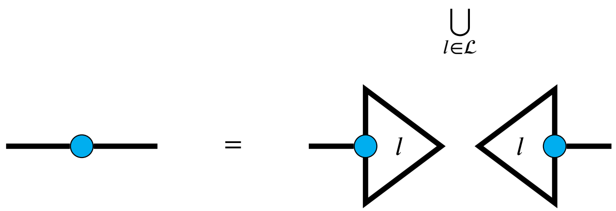
□

3.5 Topological concepts in flatland via **ContRel**

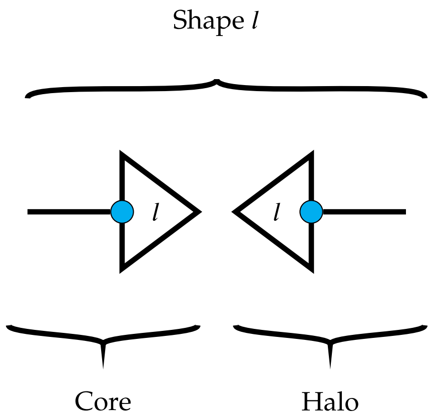
The goal of this section is to demonstrate the use of sticky spiders as formal semantics for the kinds of schematic doodles or cartoons we would like to draw. Throughout we consider sticky spiders on $\mathbb{R}^2$. In Section 3.5.1, we introduce how sticky spiders may be viewed as labelled collections 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

Definition 3.5.1 (Labels, shapes, cores, halos)**.** Recall by Proposition 3.4.13 that we can express the idempotent as a union of continuous relations formed of a state and test, for some indexing set of *labels* $\mathcal{L}$.



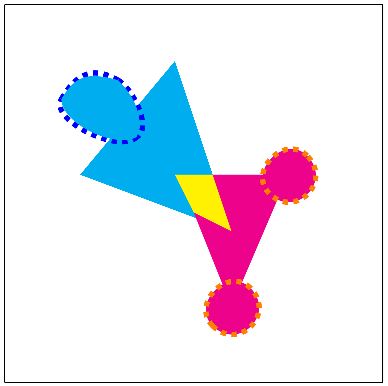
A *shape* is a component of this union corresponding to some arbitary $l \in L$. So we refer to a sticky spider as a labelled collection of shapes. The state of a shape is the *halo* of the shape. The halos are precisely the copiables of the sticky spider. The test of a shape is the *core*. The cores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



Proposition 3.5.2 (Core exclusion: Distinct cores cannot overlap)**.** *Proof.* A direct consequence of Lemma 3.4.17. □

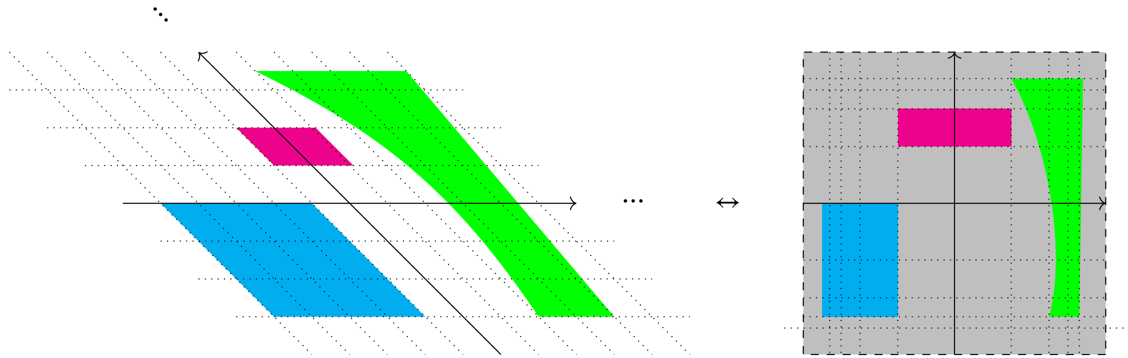
Proposition 3.5.3 (Core-halo exclusion: Each core only overlaps with its corresponding halo)**.** *Proof.* Seeking contradiction, if a core overlapped with multiple halos, Lemma 3.4.18 would be violated. □

Proposition 3.5.4 (Halo non-exclusion: halos may overlap)**.** *Proof.* By example:

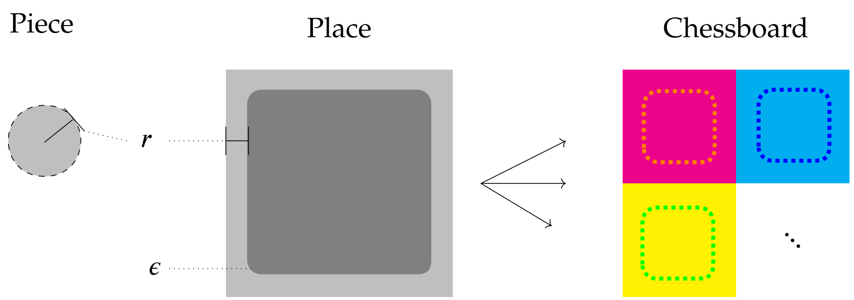


The two shapes are colour coded cyan and magenta. The halos are two triangles which overlap at a yellow region, and partially overlap with their blobby cores. The cores are outlined in dotted blue and orange respectively. Observe that cores and halos do not have to be simply connected; in this example 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. □

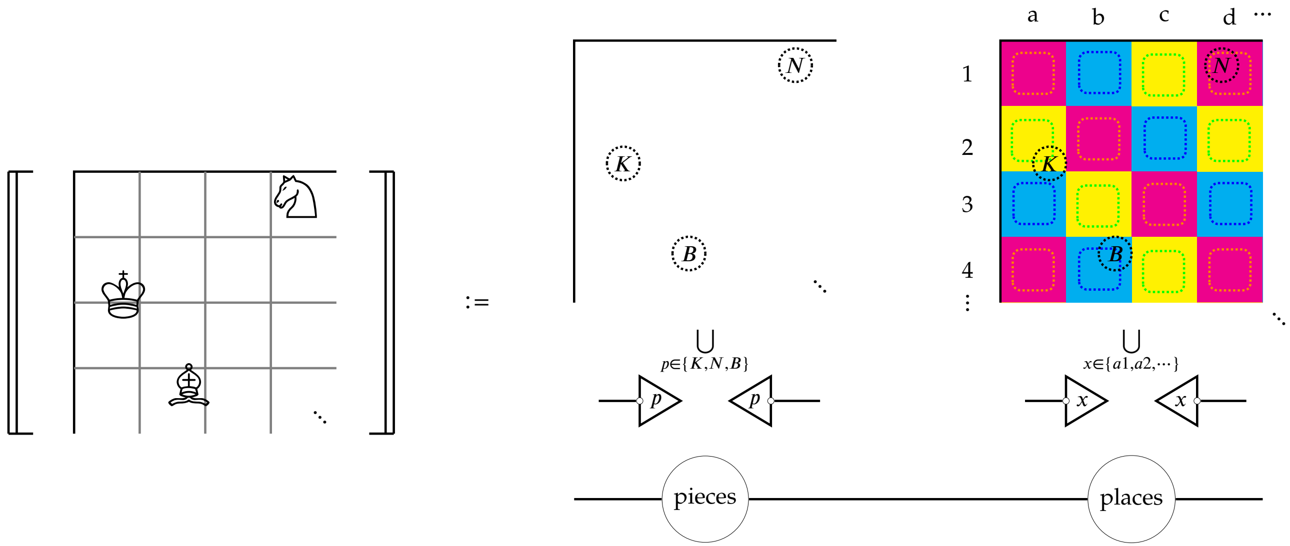
Remark 3.5.5. When we draw on a finite canvas representing all of euclidean space, properly there should be a fishbowl effect that relatively magnifies shapes close to the origin and shrinks those at the periphery, but that is only an artefact of representing all of euclidean space on a finite canvas. 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 3.5.6 (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.

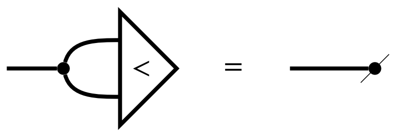
The reals There are many spaces homeomorphic to the real line. How do we know when we have one of them? The following theorem provides an answer:

Theorem 3.5.7 (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 $\mathbb{R}$.

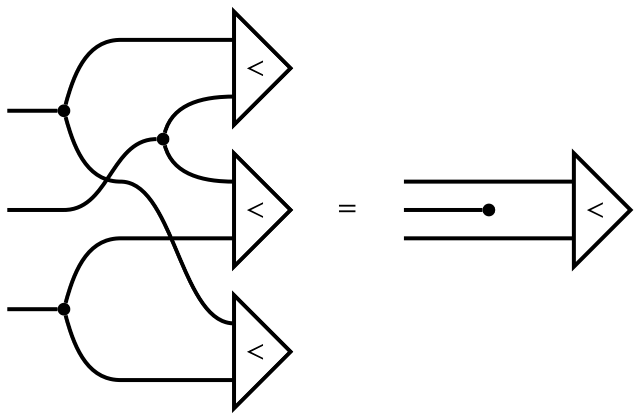
We can define all of these pieces using diagrammatic equations.

Less than We define a total order $<$ as an open set – i.e. a test – on $X \times X$ that obeys the usual axiomatic rules:

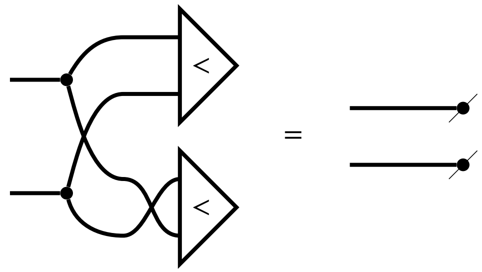
Antireflexive ($\forall x : x \not< x$)



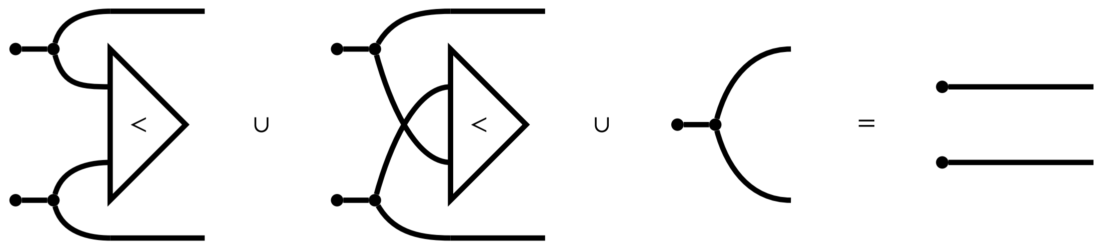
Transitive ($\forall xyz : x < y \;\&\; y < z \Rightarrow x < z$)



Antireflexive $\forall xy \neg(x < y \;\&\; y < x)$



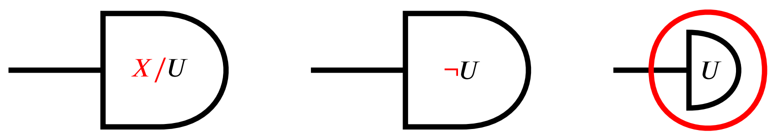
Trichotomy ($\forall xy : x < y \lor y < x \lor x = y$)



Endpoints We can introduce endpoints for open intervals directly by asking for the space X to have points that are less than or greater than all other points. Another method, which we will use here for primarily aesthetic reasons, is to use endocombinators to define suprema. Endocombinators are like functional expressions applied to diagrams. For a motivating example, consider the case when we have a locally indiscrete topology:

Definition 3.5.8 (Locally indiscrete topology)**.** (X, τ) is *locally indiscrete* when every open set is also closed.

If we know that a topology is locally indiscrete and we are given an open U, we would like to notate the complement X/U – which we know to be open – as any of the following, which only differ up to notation.

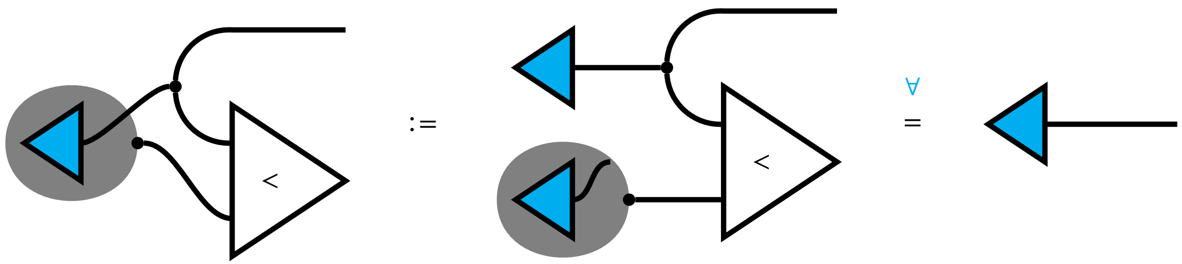


Unfortunately, the complementation operation $X/-$ is not in general a continuous relation, hence in the lattermost expression above we resort to using bubbles as a syntactic sugar. Formally, these bubbles are *endocombinators*, the semantics and notation for which we borrow and modify from [].

Definition 3.5.9 (Partial endocombinator)**.** In a category C, a *partial endocombinator* on a homset $(C)(A, B)$ is a function $(C)(A, B) \to (C)(A, B)$

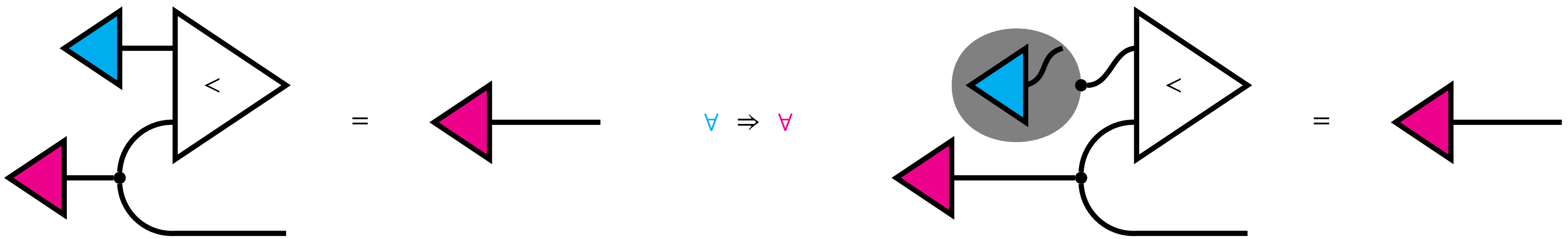
Using this technology, we can define:

Upper Bound



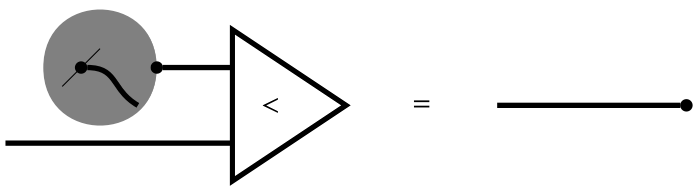
And we can add in further equations governing the upper bound endocombinator to turn it into a supremum, where the lower endpoint is obtained as the supremum of the empty set, and the upper endpoint is the supremum of the whole set.

Supremum

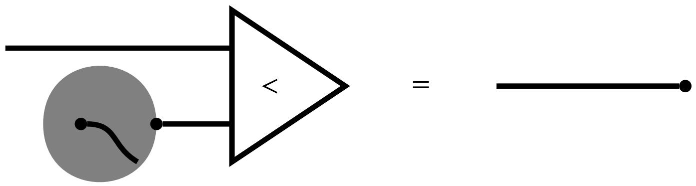


Now we can define endpoints purely graphically:

Lower endpoint



Upper endpoint

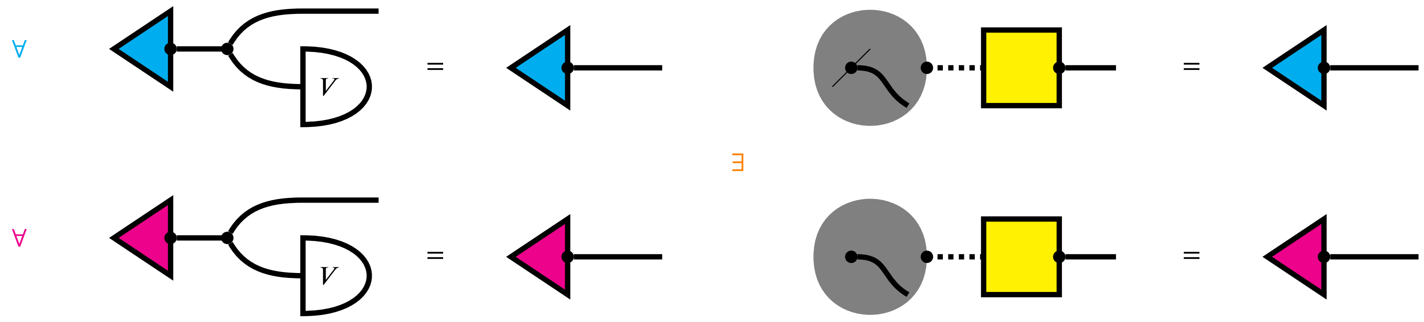


Going forward, we will denote the unit interval using a thick dotted wire.

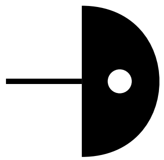
Simply connected spaces

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.

V is *simply connected* when:



This 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.

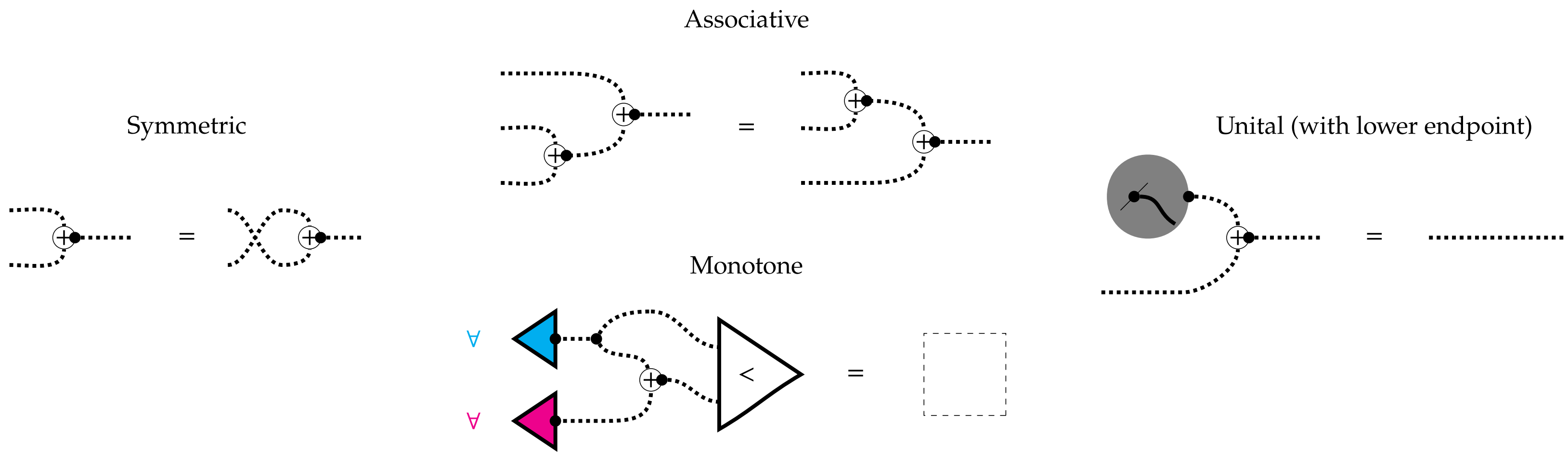


3.5.3 Displacing shapes

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.

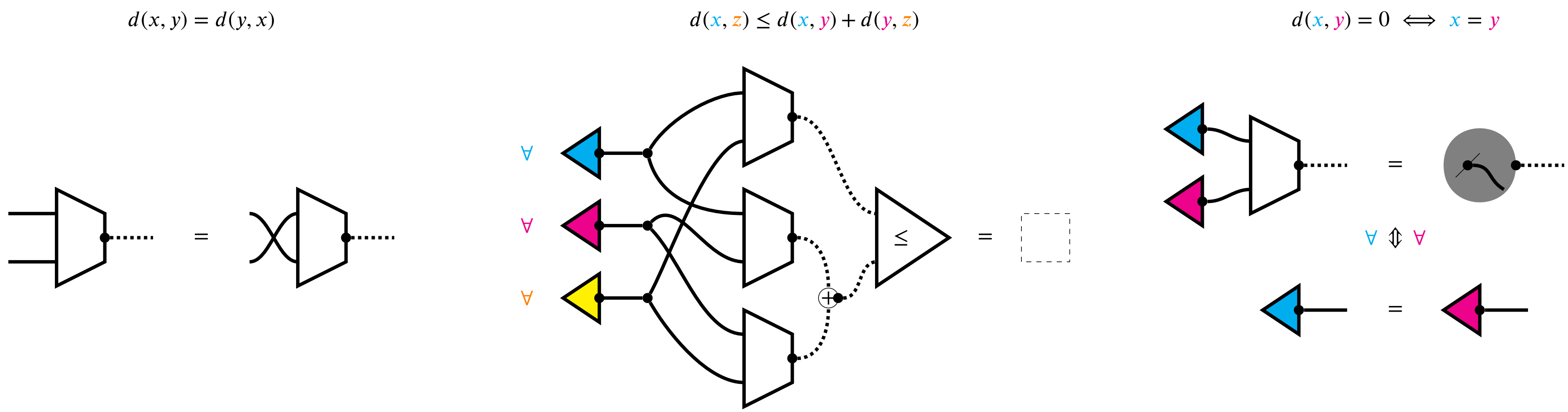
Addition

More precisely, we only need an additive monoid structure on the unit interval. We do not care about obtaining precise values from our metric, and we will not need to subtract distances from each other. All we need to know is that the lower endpoint stands in for "zero distance" – as the unit of the monoid – and that adding positive distances together will give you a larger positive distance deterministically.



Metrics

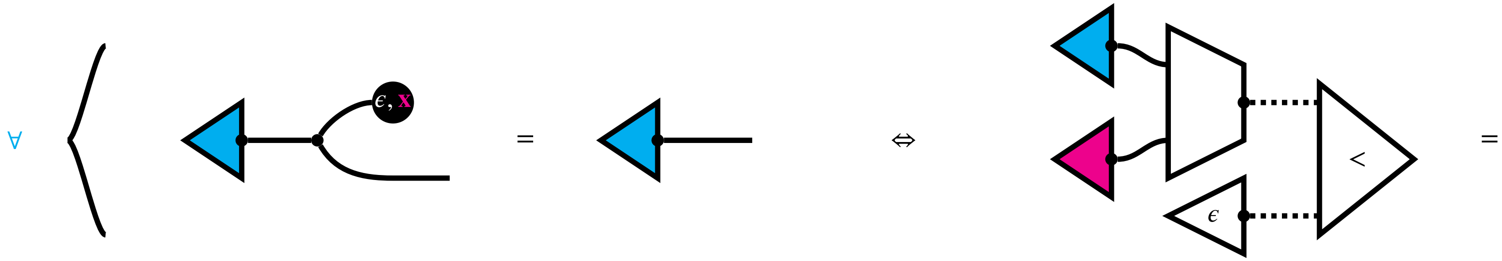
A metric on a space is a continuous map $X \to \mathbf{R}^+$ to the positive reals that satisfies the following axioms.



Open balls

Once we have metrics, we can define the usual topological notion of open balls. Open balls will come in handy later, and a side-effect which we note but do not explore is that open balls form a basis for any metric space, so in the future whenever we construct spaces that come with natural metrics, we can speak of their topology without any further work.

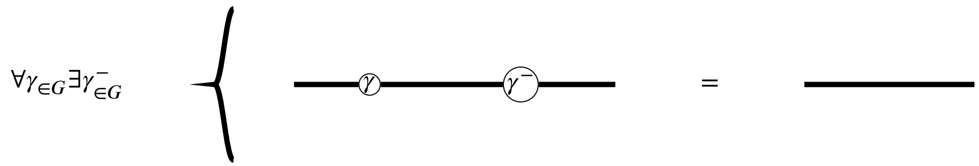
Open ball of radius ϵ at a point **x**



TOPOLOGICAL GROUP

It is no trouble to depict collections of invertible transformations of spaces $X \to X$:

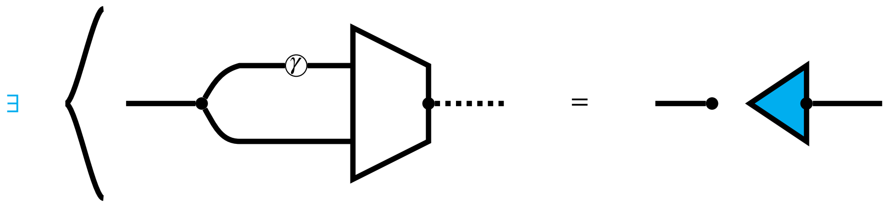
A topological group G



ISOMETRY

But recall that the collections of invertible transformations we are really interested in are the *rigid* ones, the ones that move objects in space without deforming them. We can identify when a transformation is rigid by the following criterion:

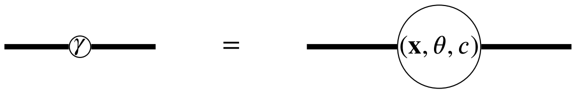
γ is an *isometry*



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 $\mathbf{R}^2$



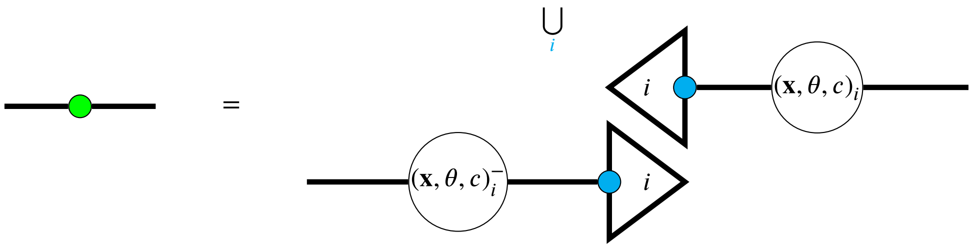
$\mathbf{x} \in \mathbf{R}^2$

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

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

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

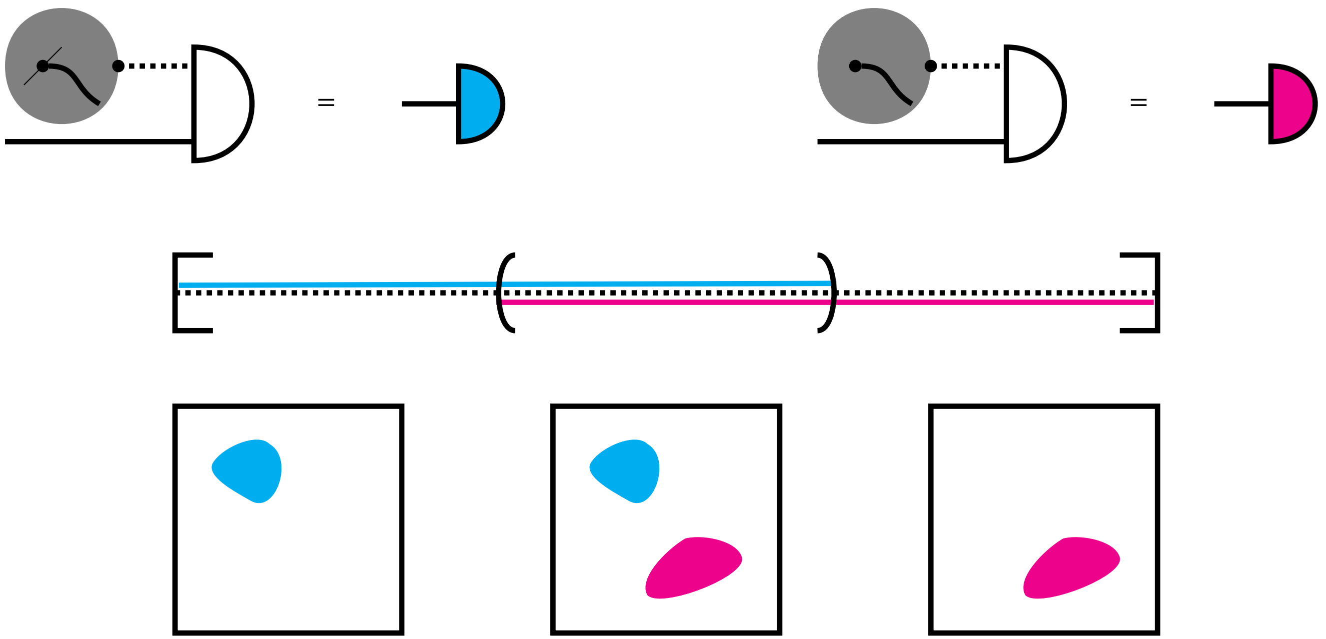
If we want continuous transformations in the plane from the configuration of shapes in one spider to end at the configuration of shapes in another, we ought to define an analogue of *homotopy*: the continuous deformation of one map to another. However, we will have to massage the definition a little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

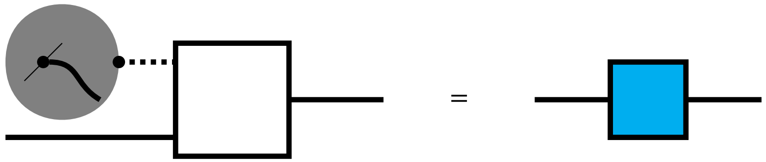
Definition 3.5.10 (Homotopy in **Top**). where f and g are continuous maps $A \to B$, a *homotopy* $\eta : f \Rightarrow g$ is a continuous function $\eta : [0,1] \times A \to B$ such that $\eta(0,-) = f(-)$ and $\eta(1,-) = g(1,-)$.

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.

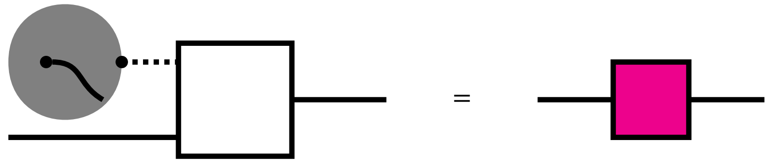


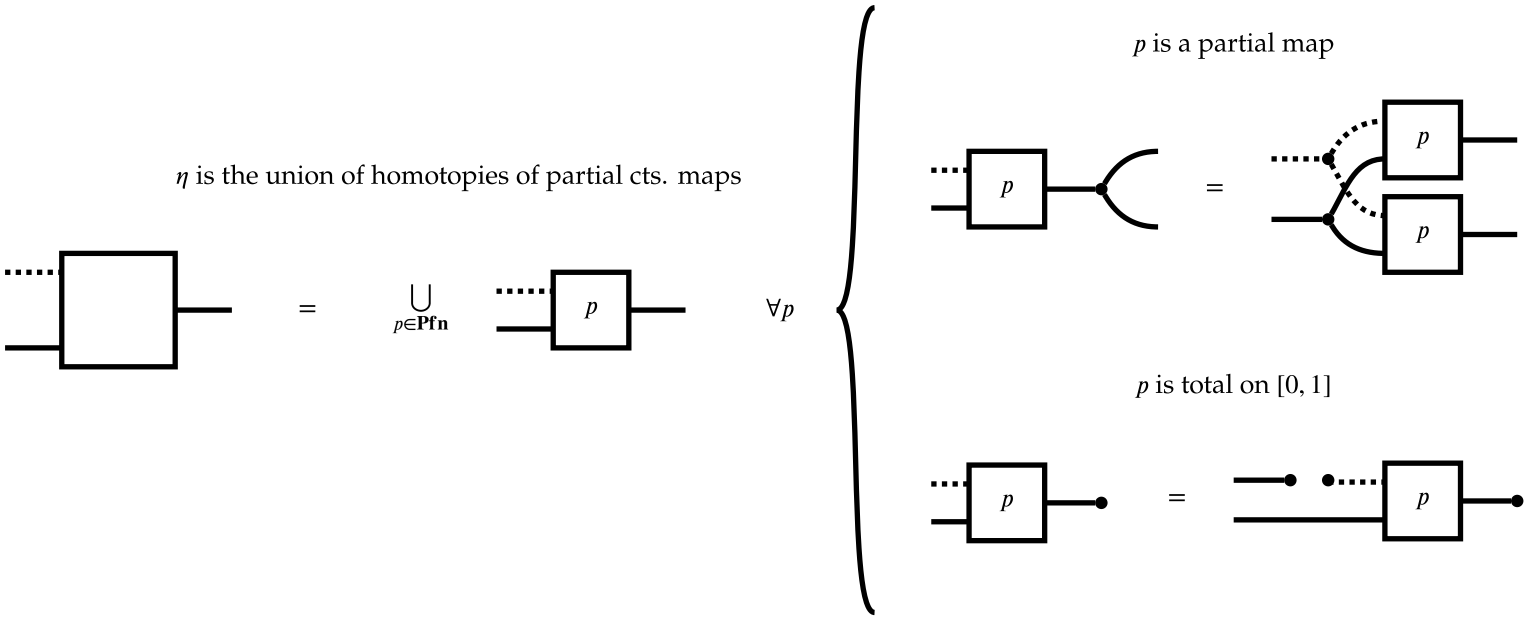
What is happening in the above film is that we have our starting open set, which stays constant for a while. Then suddenly the ending open set appears, the starting open disappears, and we are left with our ending; while *technically* there was no discontinuous jump, this isn't the notion of sliding 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.

$\eta(0,-) = f(-)$

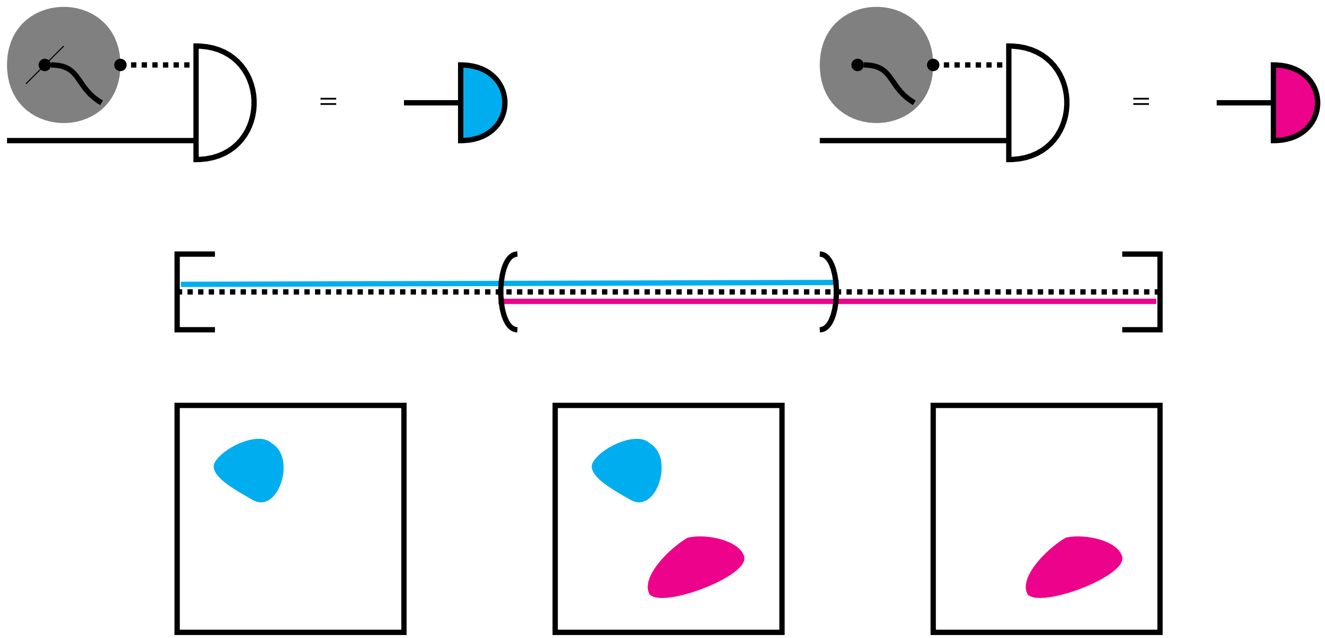


$\eta(1,-) = g(-)$

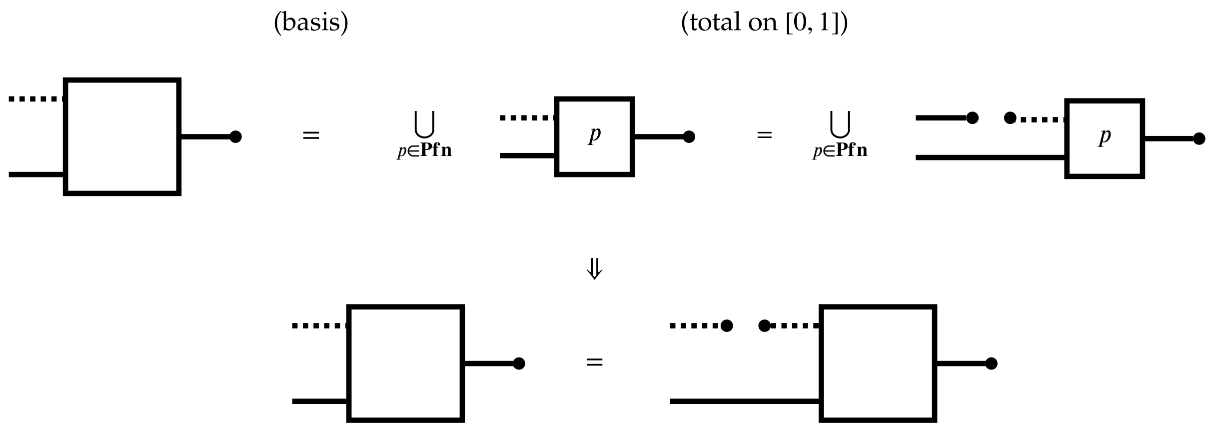




Observe that the second condition asking for decomposition in terms of partial comes for free by Proposition 3.3.20; the constraint of the definition is provided by the first condition, which is a stronger condition than just asking that the original continuous relation be total on I:



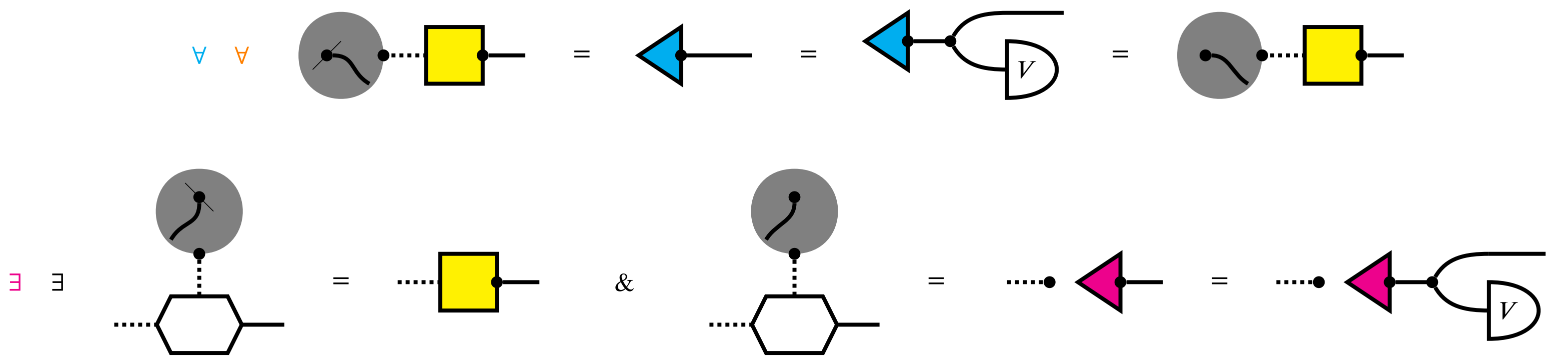
This definition is "natural" in light of Proposition 3.3.20, that the partial continuous functions $A \to B$ form a basis for $\mathbf{ContRel}(A, B)$: we are just asking that homotopies between partial continuous functions – which can be viewed as regular homotopies with domain restricted to the subspace topology induced by an open set – form a basis for homotopies between continuous relations.



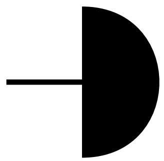
CONTRACTIBLE SPACES

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.

V is *contractible* when:



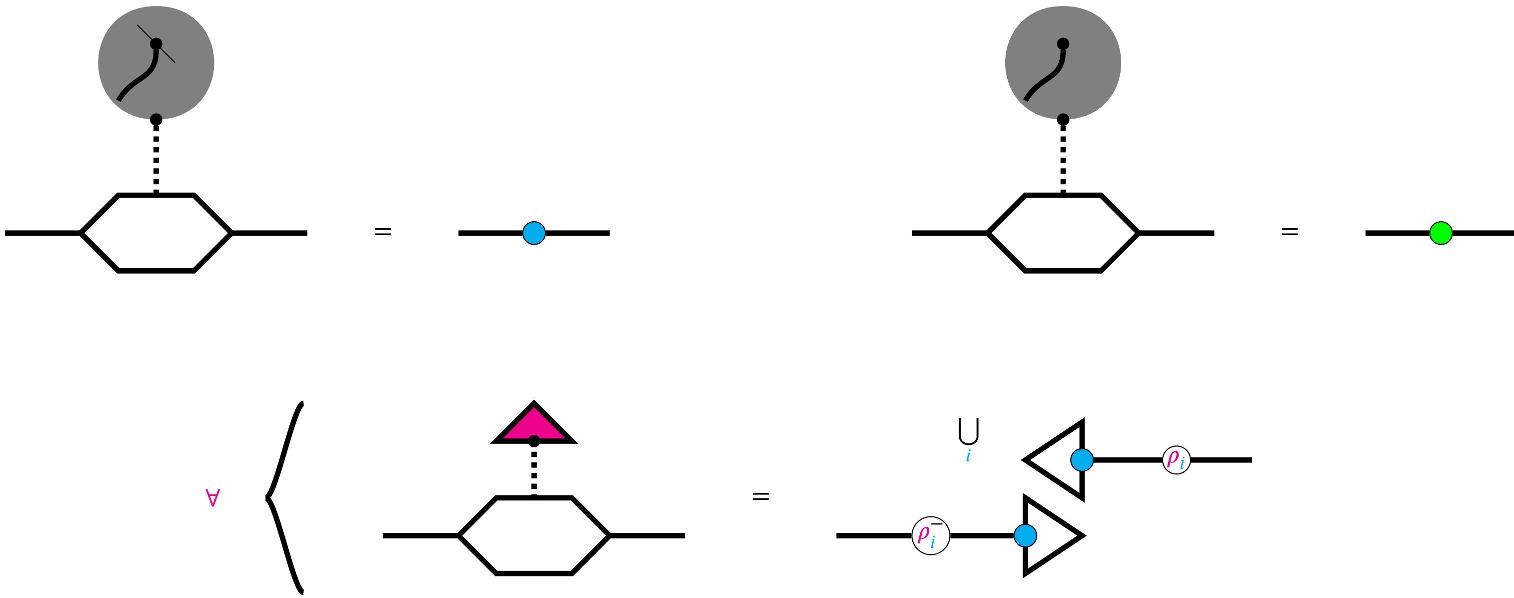
Contractible open sets are worth their own notation too; a solid black effect, this time with no hole.



3.5.5 *Rigid motion*

Now at last we can define sliding shapes. What we mean by two sticky spiders being relatable by sliding shapes is that we have a homotopy that begins at one and ends at the other, such that every point in between is itself a sticky spider related to the first by rigid displacement.

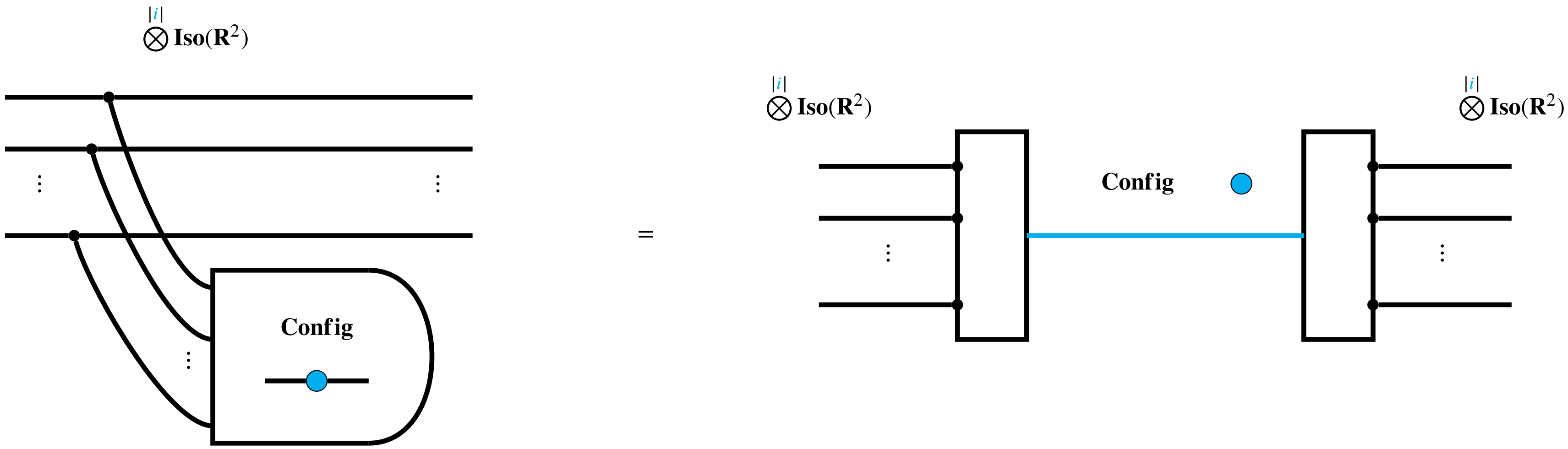
Rigid motion



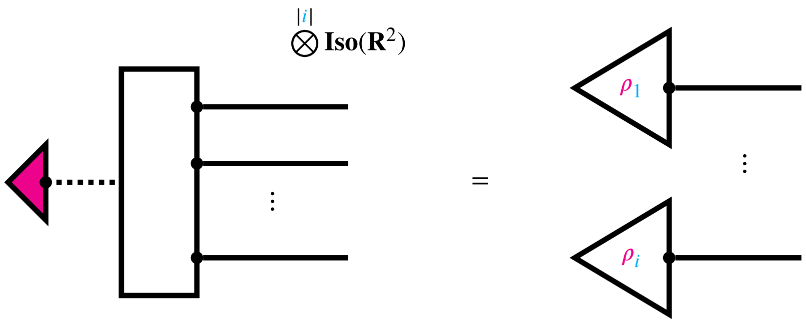
Configuration spaces

We can depict the *configuration space* of shapes that are obtainable by displacing the shapes of a given spider by a split idempotent through the n-fold tensor of rigid transformations – a restriction to the subspace of the largest open set contained in the subset of all valid (with correct chirality) combinations of displacements that yield another spider.

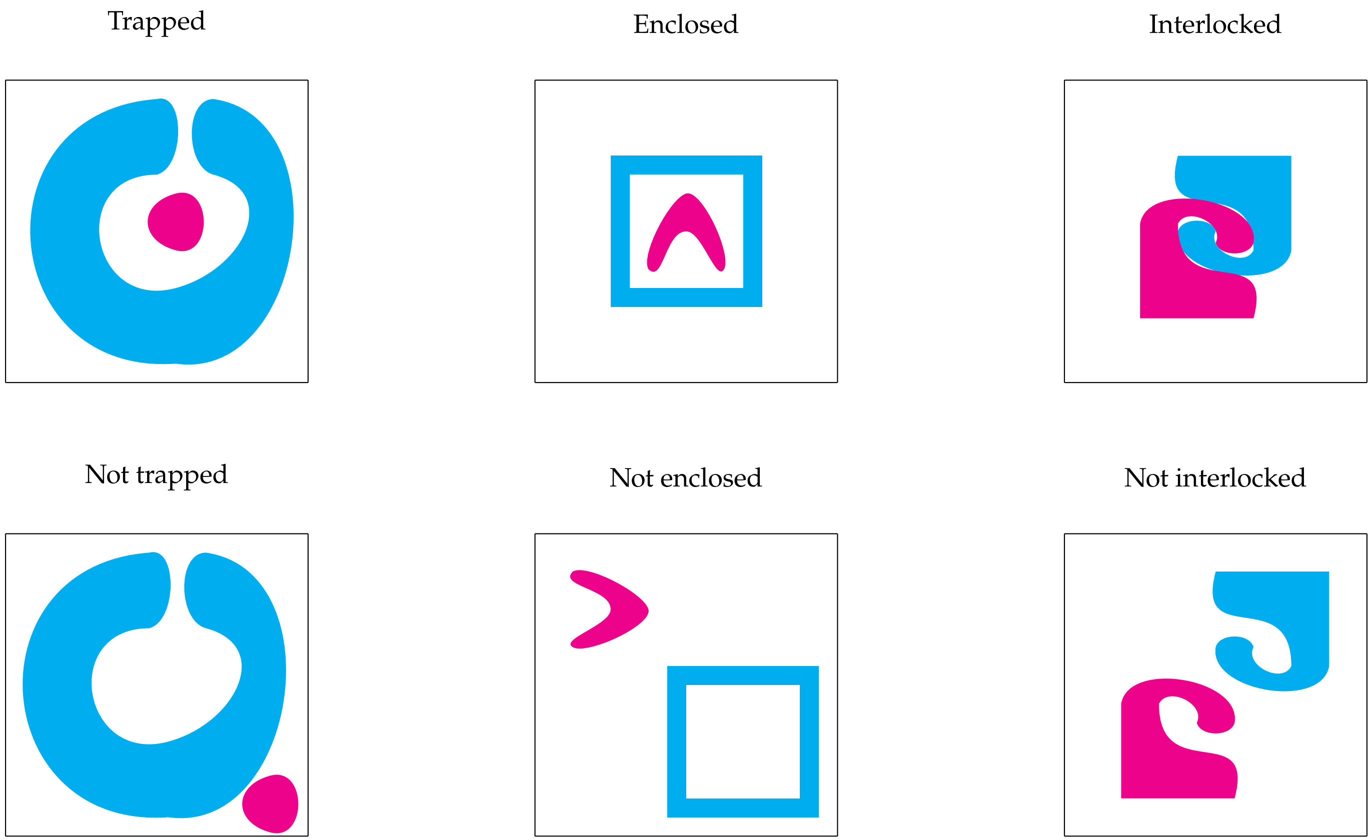
Configuration space of a sticky spider



Observe that the data of rigid motion on a sticky spider as we have defined above can be captured as a continuous map from the unit interval to rigid transformations: one for each shape in the spider. This is precisely a continuous path in configuration space.



What are the connected components of configuration space? Evidently, there are pairs of spiders that are both valid displacements, but not mutually reachable by rigid motion. For example, shapes might *enclose* or *trap* other shapes, or shapes might be *interlocked*. Depicted below are some pairs of configurations that are mutually unreachable by rigid transformations:



Now we have the conceptual toolkit to begin modelling these concepts in the configuration space of a sticky spider.

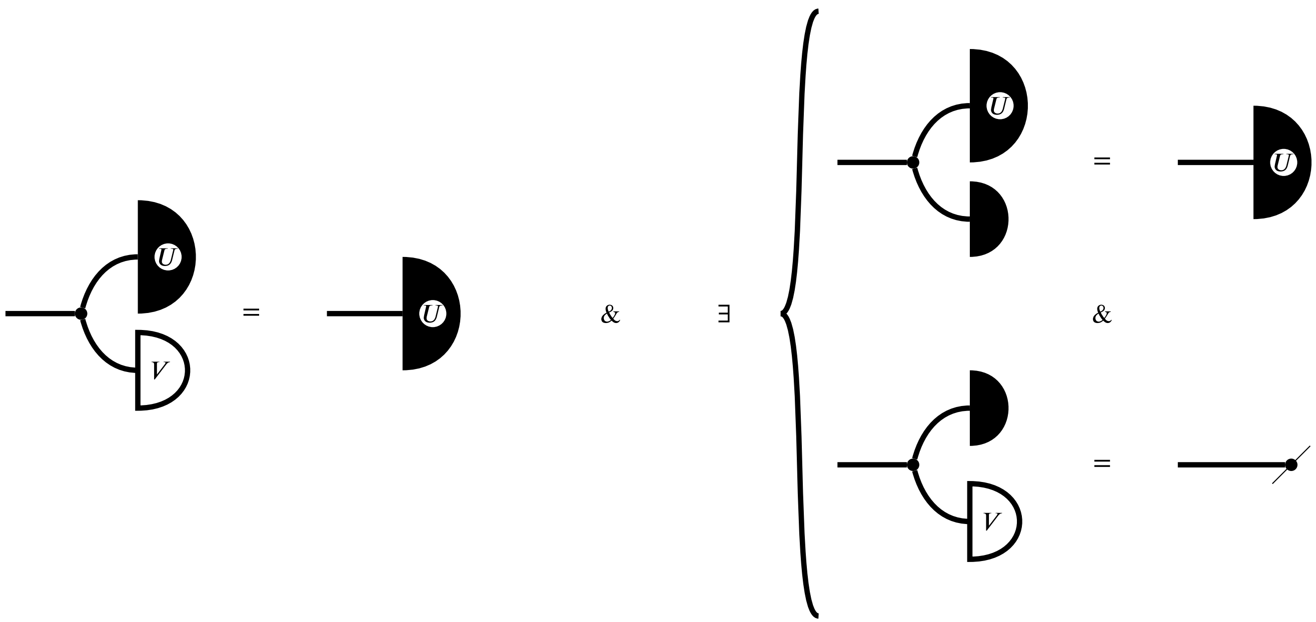
3.5.6 *Modelling linguistic topological concepts*

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:

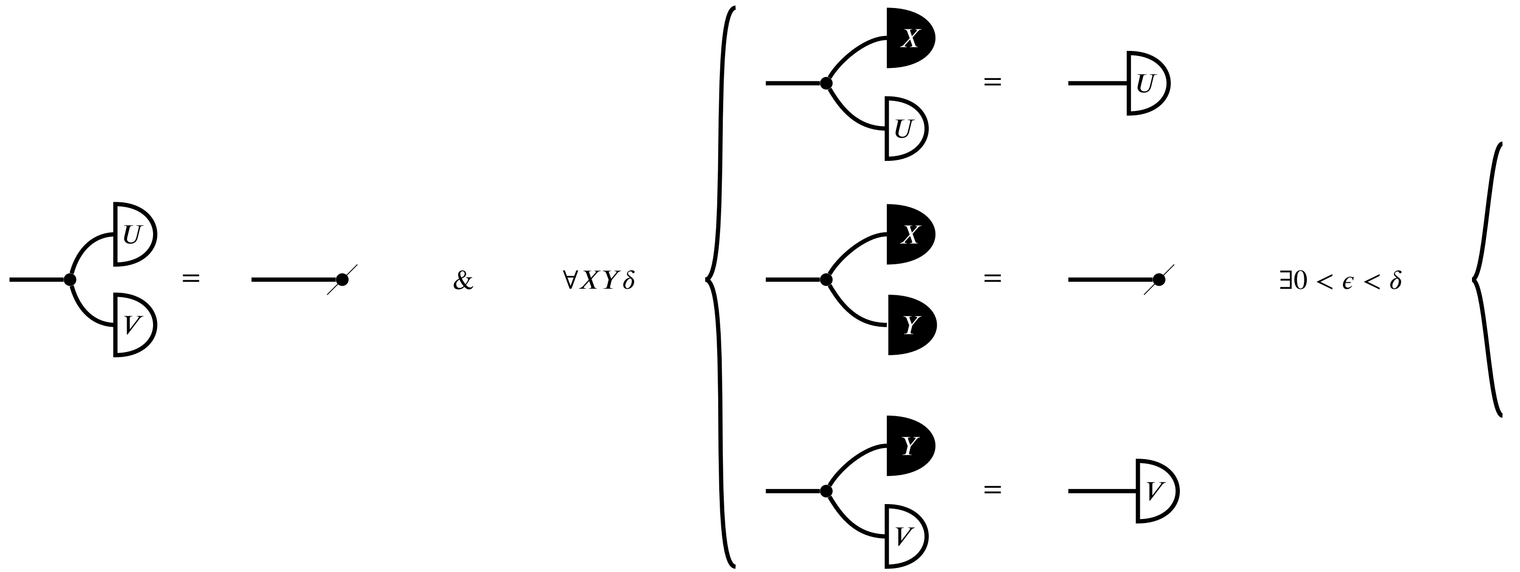
U is *a part of* V



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:

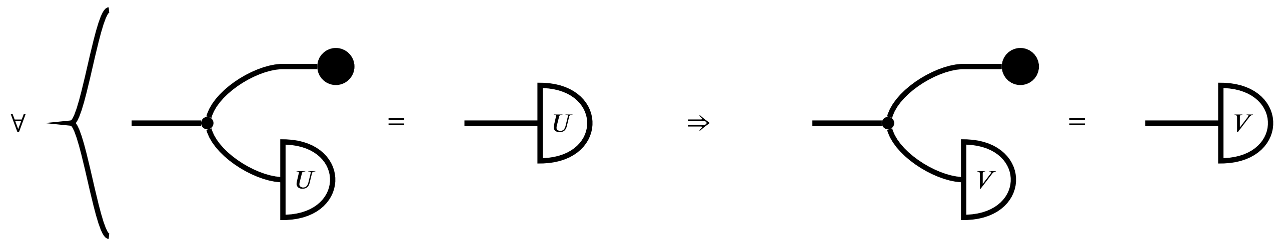
U and V are *touching*



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.

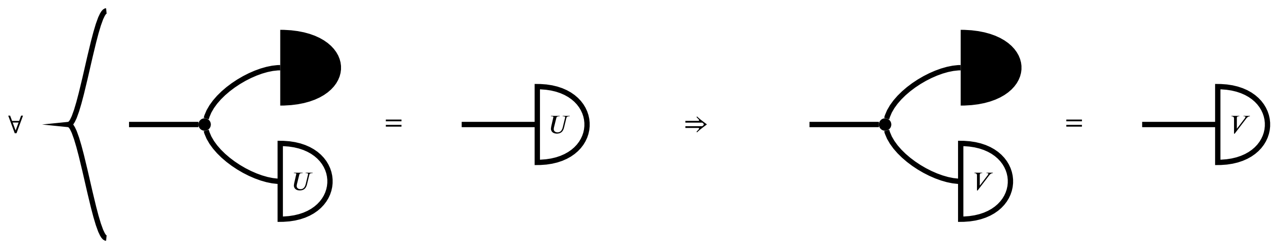
V is *within* U, or U *surrounds* V



Containers and enclosure

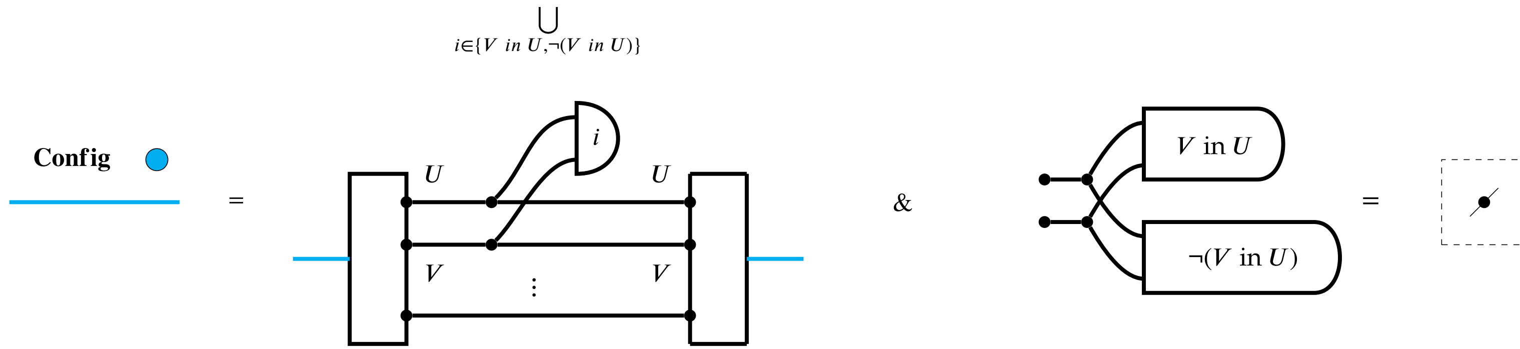
There is a strong version of within-ness, which we will call enclosure. As in when we are in elevators and the door is shut, nothing gets in or out of the container. Intuitively, there is a hole in the container surrounded on all sides, and the contained shape lives within the hole. To give a real-world 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



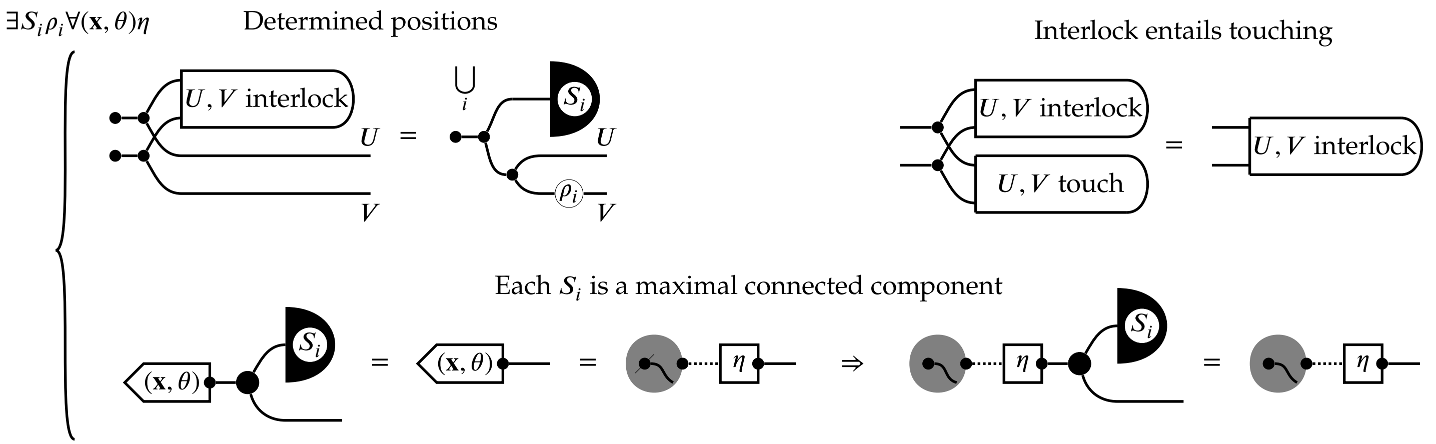
Trapped

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.



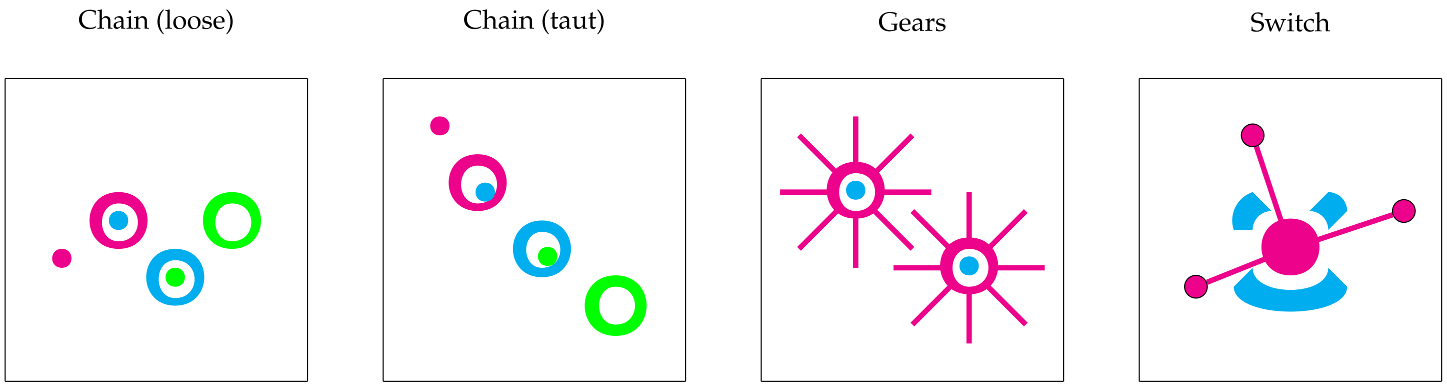
Interlocked

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

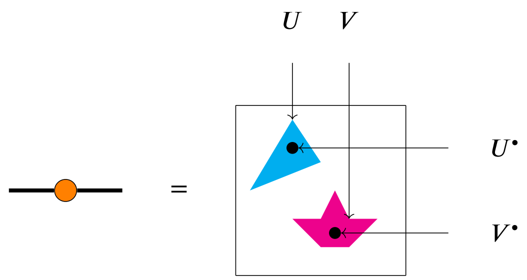
A weaker notion of interlocking is when shapes only imperfectly determine each other's potential displacements, by specifying an allowed range. Here is an understatement: there is some interest in studying how shapes mutually constrain each other's movements in this way.



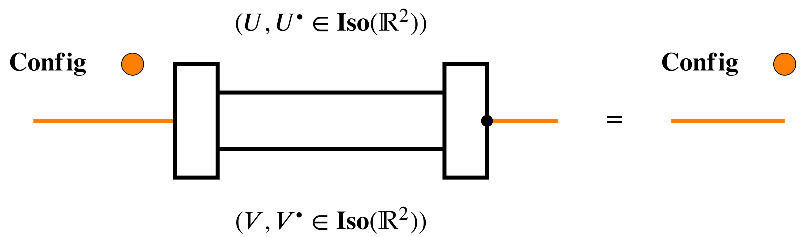
There are as many definitions to go through here as there are potential mechanical models, and among other things, there are mechanically realised clocks [], computers [], and analogues of electric circuits []. So instead, we will allow ourselves to additionally specify open sets as concepts in configuration 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.

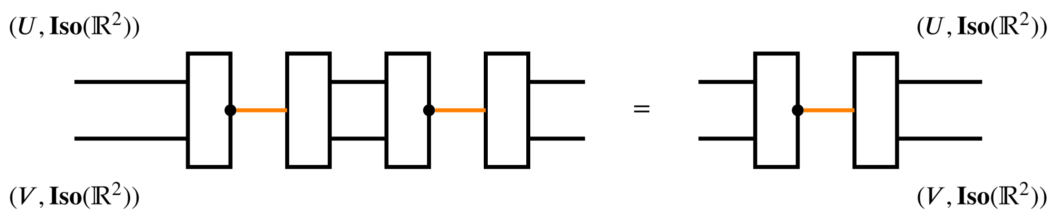
3.5.7 States, actions, manner

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.

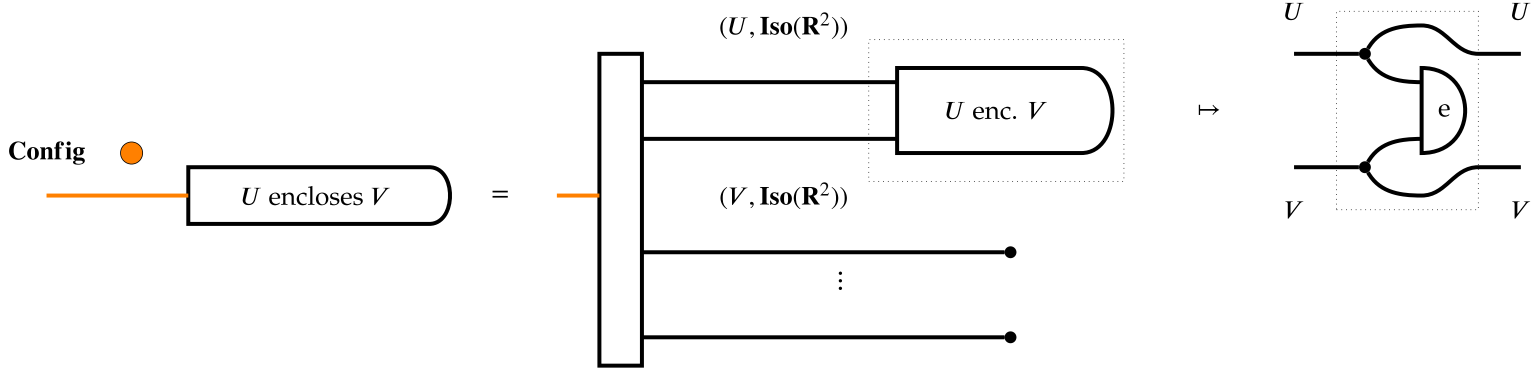


Concretely, for each shape in a sticky spider,
in order to reconstruct the shape,
the data of configuration space
only has to remember a basepoint
(which the isometries act upon)
paired with a label naming the shape
(so that the extent of the shape is reconstructible)

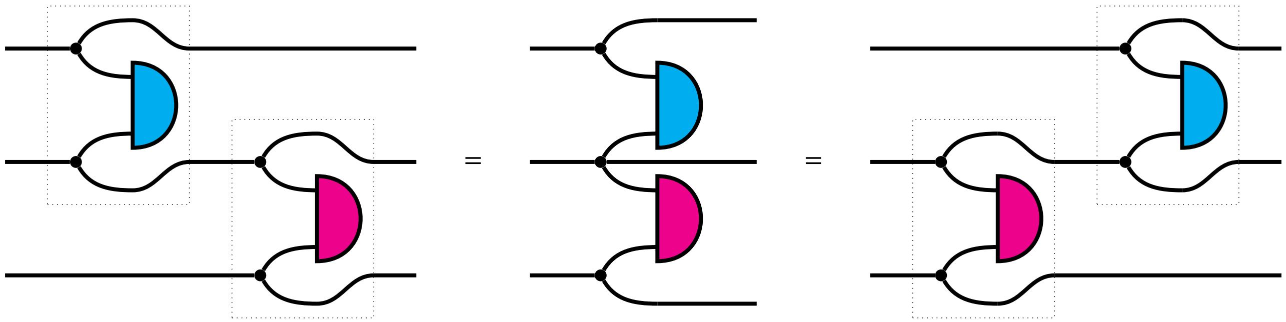




All of the concepts we have defined so far are open sets in configuration space – and for any concept that isn't, we are always free to take the interior of the set; the largest open set contained within the concept. Passing through the split idempotent, we can recast each as a circuit gate using copy maps.

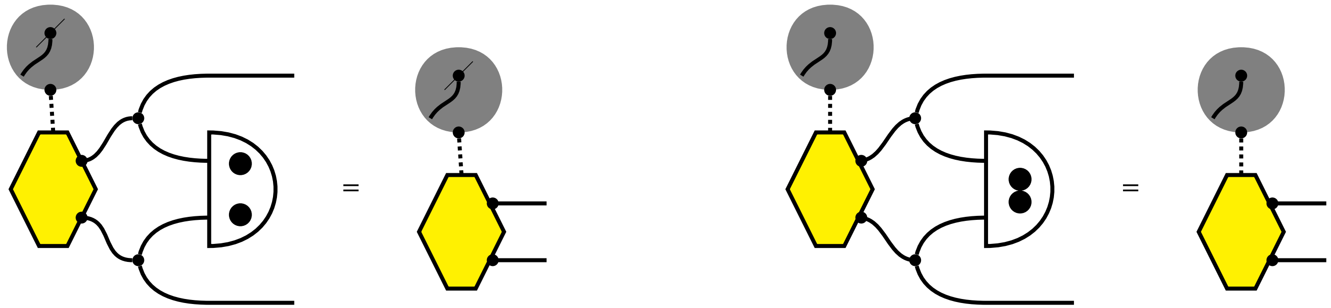


Going forward, we will just label the wires with the names of each shape when necessary. We notice that one feature of this procedure to get gates from open sets is that all gates commute, due to the commutativity of copy.



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.

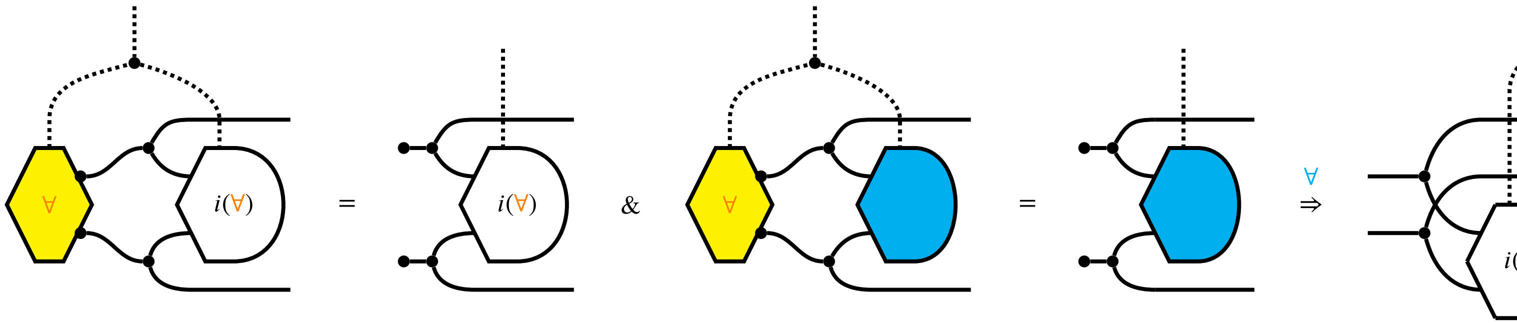
A particular collision trajectory



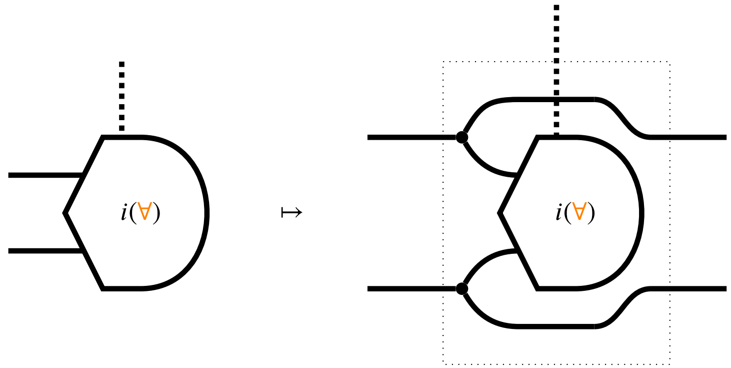
At $t = 0$, the shapes do not touch

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.

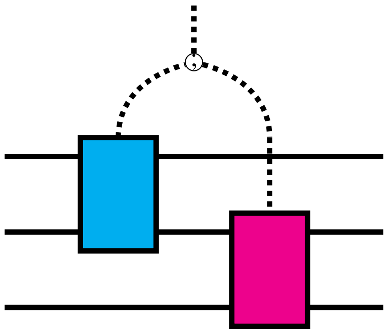


Once we have the open set $i(\forall)$ that corresponds to all expressible collisions, we have a homotopy-parameterised gate. Following a similar procedure, we can construct gates of motion that satisfy whatever pre- and postconditions we like.

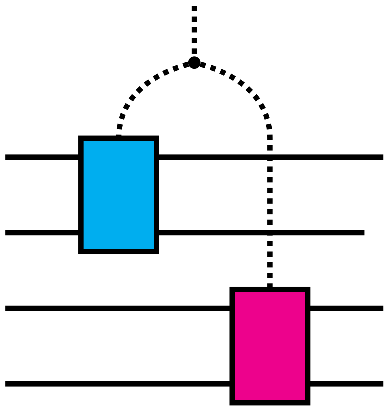


We can compose multiple rigid motions sequentially by a continuous function ; that splits a single unit interval into two: $; := x \mapsto \begin{cases} (2x, 0) \text{ if } x \in [0, \frac{1}{2}) \\ (1, 2x-1) \text{ if} x \in [\frac{1}{2}, 1] \end{cases}$. The effect of the map is to splice two vignettes 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 3.5.7 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.

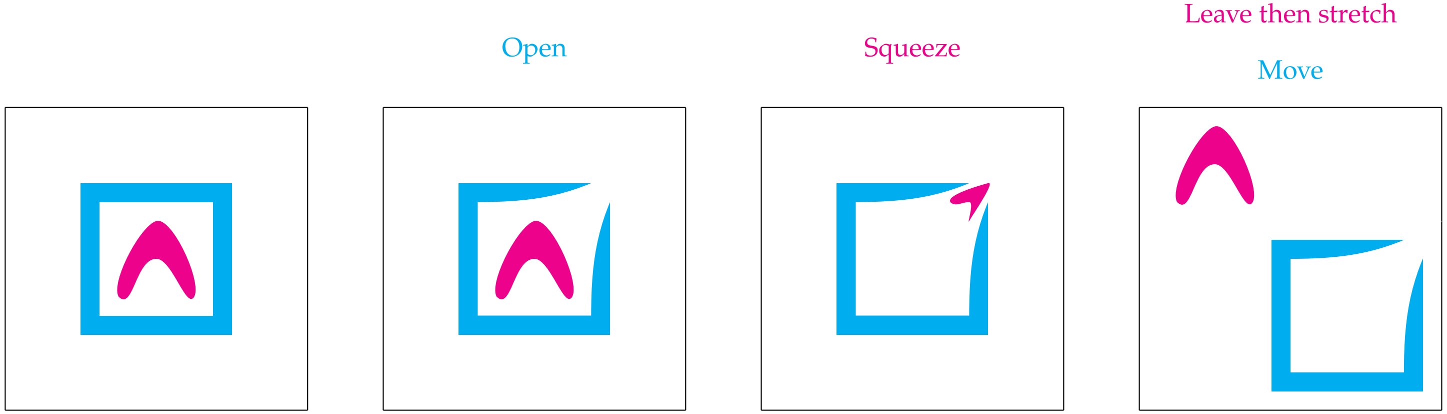
Sequential composition of motions



Parallel composition of motions

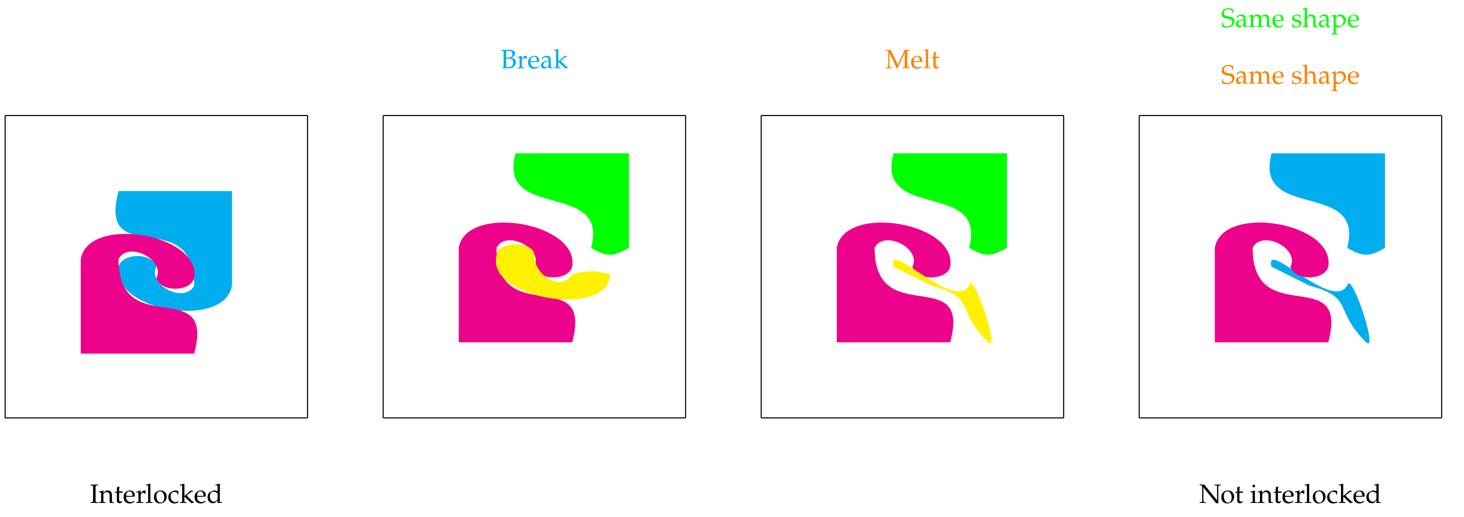


It is easy to see that the gates can always be rewritten to respect the composition order given by ; and copy, since for any input point at the unit interval the gates behave as restrictions to open sets. These new gates do not generally commute; consider comparing the situation where a tenant moves 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.



Enclosed

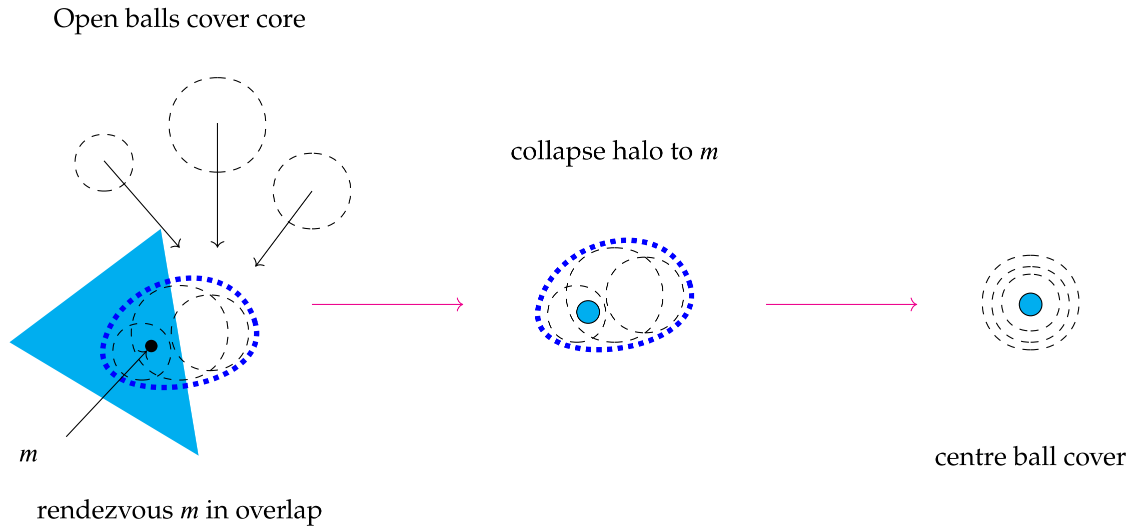
We can further generalise by noting that completely different spiders can be related by homotopy, so we can model a situation where there is a permanent bend, or how a rigid shape might shatter.



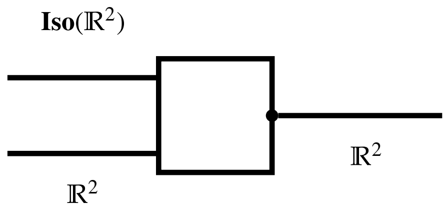
We provide the following construction as a general recipe to construct homotopies between spiders.

Construction 3.5.11 (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:

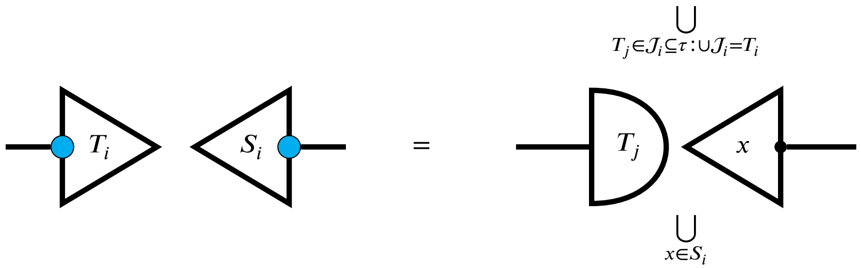


Second, in order to perform the sliding of open balls, we observe that, given a basepoint to act as origin (which we assume is provided by the data of the split idempotent of configuration space) we can express the group action of rigid isometries $\mathbf{Iso}(\mathbf{R}^2)$ on $\mathbf{R}^2$ as a continuous function:



$$\big((\mathbf{a},\theta),\mathbf{b}\big) \mapsto \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \mathbf{b} + \mathbf{a}$$

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 3.5.6, 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.

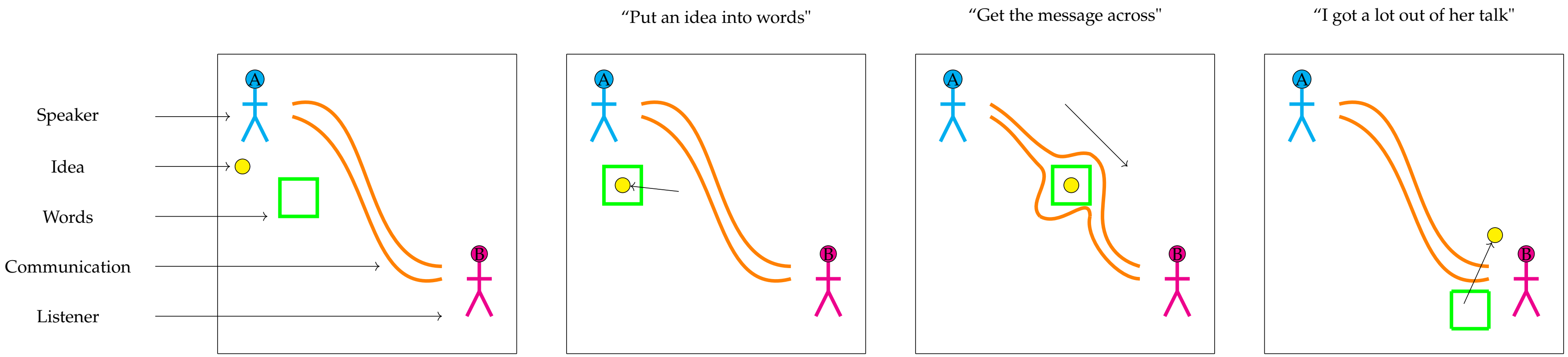
OBJECTION: THAT IS WAY OUTSIDE THE SCOPE OF FORMAL SEMANTICS. Insofar as semantics is sensemaking, we certainly are capable of making sense of things in terms of mechanical models and games by means of metaphor, the mathematical treatment of which is concern of Section **??**, so I claim that I am, definitionally, *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 Continuous Relations: A concept-compliant setting for text circuits

In this chapter, we introduce *continuous relations*, which are a naïve extension of the category **Top** of topological spaces and continuous functions towards continuous relations. We choose this category (as opposed to plain **Rel**, the category of sets and relations) because it satisfies several requirements arrived at by introspection of some of the demands of modelling language. These justifications involve basic reflections upon language use, and the consequent mathematical constraints those affordances impose on any interpretation of text circuits in a symmetric monoidal category; A priori it could well be that there is no non-trivial process theory that satisfies these constraints, so the onus is on us to show that there exists such a process theory. I outline these justifications – mostly intended for readers interested in how any of this relates to the cognitive aspect of text – after this subsection. The justifications can also be skipped and optionally revisited after the reader is more familiar with what **ContRel** is.

Second, we introduce **ContRel** diagrammatically. In the appendix for this chapter we do the bookwork demonstrating that it is a symmetric monoidal category, and we relate it to the well studied categories **Rel** and **Loc**. To the best of my knowledge, the study of this category is a novel contribution, for reasons I list prefacing the bookwork.

Third, once we have defined a stage to perform calculations in **ContRel**, our aim is to introduce some actors. We seek to make formal the kinds of informal schemata we might doodle on paper to animate various processes occurring in space. One good reason for doing this is to establish formal foundations for the semantics of metaphor, some of the most commonly used of which involve spatial processes in a way that is fundamentally topological []. For example, in the *conduit metaphor* [], `words` are considered *containers* for `ideas`, and `communication` is considered a *conduit* along which those containers are sent.



If you are already happy to treat such doodles as formal, then I think you're alright, you can skip the rest of this chapter. If you are a category theorist or just curious, hello, how are you, please do write me to share your thoughts if you care to, and please skip the rest of this paragraph. If you are still reading, I assume you are some kind of smelly epistemic-paranoiac Bourbaki-thrall sets-and-lambdas math-phallus-worshipping truth-condition-blinded symbol-pusher who takes things too seriously. I hope you choke on the math. I will begin the intimidation immediately.

We provide a generalisation (Definition 3.4.5) of special commutative frobenius algebras in **ContRel** that cohere with idempotents in the category. The relation of (†)-special commutative algebras in **FdHilb** to model observables in quantum mechanics is well-studied [], as is the role of idempotents in generalisations of quantum logic to arbitrary categories [], therefore this generalisation may be viewed as a unification of these ideas to define doodles in **ContRel**. N.B. we are not quite taking the Karoubi envelope of **ContRel**, as we are restricting the idempotents we wish to consider to only those that also behave appropriately as observables. The reason

for this restriction lies in Theorem 3.4.8 which provides a diagrammatic characterisation result that allows us to precisely identify
gory splits though a discrete topology. The discrete topology thus acts as a set of labels, where the (pre)images of each element u
the kind of shapes we wish to consider. As an interlude, we demonstrate how we may construct families of such idempotent-cohe
algebras on $\mathbb{R}^2$ to provide a monoidal generalisation (c.f. the monoidal computer framework in []) of **FinRel** equipped with a Tur
tion **??**. Then we proceed to define configuration spaces of collections of shapes up to rigid displacement, and we develop a relati
motions as paths in configuration space, along with appropriate extensions to accommodate nonrigid phenomena. If you are still
theorist nor just curious, I will drop the jargon now, because you are probably either questioning or deeply committed to your fal
just to spite the latter, I will proceed to do everything diagrammatically as much as possible.

3.6.1 *Why not use something already out there?*

Justification 1: We want a symmetric monoidal category. We want to work process-theoretically as much as possible, becau
allows whatever we construct in this setting to generalise to other process theories. Also, we want an excuse to build up and play
from scratch, just to see what formal work is involved in doing so.

Justification 2: We want to model concepts. Second, we have some cognitive considerations: how do we move between conc
– and symbolic representation and manipulation? Here we sidestep the debate around what concepts actually *are*, aligning ourse
sume that there are *conceptual spaces* that organise concepts of similar domain – such as colour, taste, motion – and that regions of
cepts. Gärdenfors' stance is backed by empirical data [], but even if he is wrong, he is at least interestingly so for our purposes. A
of colourspace is also one of the best studied and implemented: there are many different embeddings of the space of visible colou
ting we can mathematically model the action of categorising a particular point in colour space as `blue` by checking to see whether
colourspace that the symbol `blue` is associated with. So we find that this view is conducive to modelling concepts as spatial entit
sive framework, which will allow us to calculate interesting things – whilst also having the ability to handle them using symbolic
do we model this association between space and sets of labels? This leads us to the following consideration: In a category for con
conceptual space object Γ should possess a split idempotent through a set of concept-labels, thus encoding the association betwe
qua-symbols.

A further complication arises. Once we have a stock of symbols referring to entities in space, we can start talking about pairs o
sets of sets of entities (e.g. `autumnal colours`), arbitrary relations between the set of entities in one space and entities of another (
relates to its probable textures and tastes). Given enough time and patience, we can linguistically construct – at least – any finite r
will elaborate in Section 4, the real challenge is that every time we define a new concept in this way, we can again treat it as a sym
noun. Since nouns are first-class citizens that travel along wires in our framework, we are asking that any putative process theory
about concepts has to have some object, some wire Ξ for nouns, such that all finite relations fit into it. This leads us to the followi
concept-compliant text-circuits $\mathcal{T}$ ought to have an object Ξ such that all of **FinRel** can be encoded within Ξ somehow. But we alr
idempotents, so we can translate the two considerations of the last two paragraphs into one requirement. In prose; the first item g
$\mathcal{T}$ to associate concepts-qua-spaces to concepts-qua-symbols; the second item gives us a noun-wire in $\mathcal{T}$ that permits us to encode

we would like to treat finite relations.

Requirement 3.6.1. We call a text circuit category $\mathcal{T}$ *concept-compliant* if:

1. Any wire Γ used to model a conceptual space possesses a split idempotent through a set of concept-symbols $\mathfrak{L}$
2. Anything one can do with sets of concept-symbols in **FinRel** must be doable in $\mathcal{T}$; in particular, there exists a noun-wire Ξ such that **FinRel** embeds as a category into the subcategory of $\mathcal{T}$ generated by the split idempotents on Ξ

JUSTIFICATION 3: WE THINK TOPOLOGY IS A GOOD SETTING TO MODEL CONCEPTS SPATIALLY.

Where we differ from Gärdenfors is that we only ask for topological spaces, rather than his stronger requirements for metric spaces and convex concepts, so we are following the spirit but not the letter. There are several reasons for this choice. First, topology is a primitive mathematical framework for space; all metric spaces are topological spaces, but not vice versa, so this is a conservative generalisation of Gärdenfors. Second, there are technical reasons that **ContRel** is desirable, For example, we can process-theoretically characterise continuous maps from the unit interval fairly easily to model things going on in space and time; without topology around, for instance in the setting of just **Rel**, I do not know if it is even possible to pick out the continuous maps from all the others purely process-theoretically. Third and perhaps most interestingly, there are also good reasons to think that this is the right way to think about conceptual spaces. We can view topology as a framework for conceptual spaces where we consider the open sets of a topology to be the concepts. Éscardo provides a the following correspondence [], which we extend with "Point" and "Subset", and an additional column for interpretation as conceptual space:

ContRel reflects the above correspondences diagrammatically. Open sets are precisely tests, and it is always possible to construct finite intersections of opens using copy-relations. Modelling concepts as open sets or tests aligns them with semi-decidability. Given any state-instance, we can test whether it overlaps with a concept graphically: success returns a unit scalar, failure returns the zero scalar. Moreover, another independent diagrammatic calculus for concepts by Tull [] agrees with ours; concepts are effects, copy maps take intersections of concepts, and so on. Tull's diagrams are interpreted in **Stoch**, the category of stochastic processes, and as a whole his design philosophy closely adheres to Gärdenfors. All this is to suggest that if you take Gärdenfors seriously, then there is something worth taking seriously about modelling concepts as effects in a monoidal category with copy-maps. However, taking diagrammatic conceptual spaces seriously also yields a no-go result for topological spaces – which includes Gärdenfors' metric spaces with convex concepts: you can only do first-order logic with equality on your concepts if your base space is discrete and finite. We explain this below.

The correspondence between spaces and type-theory extends to conceptual spaces as follows: "niceness conditions on your conceptual space correspond to the ability to form new concepts using logical operations." For example, this means that if we denote colourspace with $\mathfrak{C}$, we can only construct a concept `different colours` on $\mathfrak{C} \times \mathfrak{C}$ if we model $\mathfrak{C}$ using a Hausdorff space, such as Euclidean space. If we want to model `not X` as the complement of a colour `X` in colourspace, asking that `not X` also be a concept requires $\mathfrak{C}$ be locally indiscrete – i.e. every open set is also closed; Euclidean space is not locally indiscrete, so we cannot use set-complement as negation, we have to use something else, like the interior of the complement. If we want to have access to universal quantifiers in colourspace, so that we can sensibly construct concepts such as `the taste that apples of all colours have in common`, then we require $\mathfrak{C}$ to be compact, which Euclidean space is not, but bounded Euclidean space is. Here then is the conflict: if we take modelling concepts with spaces seriously, and we also care to do logic with concepts, there are tradeoffs to be made. In order to take equality as a concept `same colour` on $\mathfrak{C} \times \mathfrak{C}$ requires that $\mathfrak{C}$ have discrete topology, but discrete topologies on infinite sets are not compact, so you cannot also have universal quantification at the same time. Conversely, if you want universal quantification on colourspace, then you can at best have approximate equality on colours as a concept, never exact. Now we can summarise this tension. All topological spaces, including Gärdenfors conceptual spaces with metrics and convex concepts, start off with regular logic – $\exists, \wedge, \top$ – for free []. In order to obtain first-order logic with equality on the points of the space, the underlying space must be compact (to support universal quantification) and discrete (to support equality of points), and the latter condition implies locally indiscrete

(to support negation). Only finite sets with the discrete topology are compact; you can only do first-order logic with equality on y
discrete and finite.

This no-go just provides another perspective of the ancient observation that logical thought really does seem symbolic. It is als
idea of conceptual spaces is to circumvent this no-go; in our language, asking for split idempotents through (finite) discrete topolo
constructions to work on spatial representations of concepts, when those split idempotents exist. What this no-go does mean is th
looking for *precise* unifications of conceptual spaces and first-order logic-with-equality where every logical operator is viewed as a
that sends concepts to concepts and behaves properly on the space of points; there is only "good enough".

Summary of justifications. We want a symmetric monoidal category to keep the prospect of general, process-theoretic reasonin
gory to be topological spaces because we want to model conceptual spaces and calculate interesting things. While the category **To**
uous functions is already symmetric monoidal with respect to categorical product, for linguistic and concept-related consideratio
into play diagrammatically, and because **Top** is cartesian monoidal it doesn't work for our purposes. So we construct **ContRel**.

3.7 Mathematician's endnotes

This section has two aims. First is to formally demonstrate that **ContRel** is indeed a symmetric monoidal category. Second is to investigate the relationship between **ContRel** and what seem like they should be close cousins: **Rel**, **Top**, and **Loc**. We demonstrate that **ContRel** enjoys a free-forgetful adjunction with **Rel** as expected, but **ContRel** has no forgetful functor to **Loc**. We verify that **ContRel** cannot be viewed as "powering up topology with relations" in the usual ways. Specifically, **ContRel** does not arise as conservative generalisation of the Kleisli category of the powerset monad on **Set** to **Top**, nor is it equivalent to **Span**(**Top**). We provide a sketch involving display categories to attempt to explain where the topology is coming from. The failure of these (relatively sophisticated and general) techniques to modify **Top** to accommodate relations may explain why **ContRel** has no footprint in the literature, and suggests that the study of this category may be a novel contribution.

3.7.1 The category *ContRel*

Proposition 3.7.1 (**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**. □

3.7.2 Symmetric Monoidal structure

Proposition 3.7.2. $(\mathbf{ContRel}, \bullet, X^\tau \otimes Y^\sigma := (X \times Y)^{(\tau\times\sigma)})$ is a symmetric monoidal closed category.

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

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 $(R \times S)^\dagger = R^\dagger \times S^\dagger$:

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

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 ar
are done.

The natural isomorphisms are inherited from **Rel**. We will be explicit with the unitor, but for the rest, we will only check th
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 obtai
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 a
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:

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

To check the continuity of the associator, observe that product topologies are isomorphic in **Top** up to bracketing, and these isom
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

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

Monoidal Closure: Here is the evaluator.

$$placeholder$$

3.7.3 Rig category structure

Definition 3.7.3 (Biproducts and zero objects)**.** A *biproduct* is simultaneously a categorical product and coproduct. A *zero object* is
Rel has biproducts (the coproduct of sets equipped with reversible injections) and a zero object (the empty set).
Proposition 3.7.4. 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 3.7.5. 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 3.7.6. Biproducts yield another symmetric monoidal structure which the $\times$ monoidal product distributes over appropriately to yield a rig category. Throughout the chapter we have been using $\cup$, 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 two diagrammatic formalisms for rig categories that we could have used, [] and []. Neither case is perfectly suitable due to the fact that we sometimes took 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.

3.7.4 *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 3.7.7 (Any relation R between discrete topologies is continuous)**.** *Proof.* All subsets in a discrete topologies are open. □

Definition 3.7.8 (L: **Rel** → **ContRel**)**.** We define the action of the functor L:

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

On morphisms $L(X \xrightarrow{R} Y) := X^\star \xrightarrow{R} Y^\star$, the existence of which in **ContRel** is provided by Lemma 3.7.7.

Evidently identities and associativity of composition are preserved.

Definition 3.7.9 (R: **ContRel** → **Rel**)**.** We define the action of the functor R as forgetting the topological structure.

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

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

Evidently identities and associativity of composition are preserved.

Lemma 3.7.10 ($RL = 1_{\mathbf{Rel}}$)**.** 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, $FU(X \xrightarrow{R} Y) = F(X^\star \xrightarrow{R} Y^\star) = X \xrightarrow{R} Y$ □

Reminder 3.7.11 (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 3.7.12 (Coarsening is a continuous relation)**.** Let X^σ be coarser than X^τ. The identity relation on underlying points $X^\tau \xrightarrow{1_X}$

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 3.7.13 ($L \dashv R$)**.** *Proof.* We verify the triangular identities governing the unit and counit of the adjunction, which we f
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

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

First we evaluate $L \xrightarrow{L\eta} LRL \xrightarrow{\epsilon L} L$ at an arbitrary object (set) $X \in \mathbf{Rel}$. $L(X) = X^\star = LRL(X)$, where the latter equality holds b
functor on **Rel**. For the first leg from the left, $L(\eta_X) = L(1_X) = X^\star \xrightarrow{1_X} X^\star = 1_{X^\star}$. For the second, $\epsilon_{L(X)} = \epsilon_{X^\star} = X^\star \xrightarrow{1_X} X^\star = 1$
required.

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

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 fu
remembering topology". But we show that there is no such functor that forms a free-forgetful adjunction.

Reminder 3.7.14 (The category **Loc**)**.** [] 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 3.7.15. Here are informal intuitions to ease the definition. The lattice of open sets of a given topology ordered by inclusi
ogy "arbitrary unions" : "all joins" :: "finite intersections" : "finite meets". Closure under arbitrary joins guarantees a maximal elen
that is the whole space. So frames are a setting to speak of topological structure alone, without referring to a set of underlying po
serve that in the definition of continuous functions, open sets in the *codomain* must correspond (uniquely) to open sets in the *doma*
induces a frame homomorphism going in the opposite direction that the function does between spaces, hence, to obtain the categ
we reverse the arrows of **Frm**. Observe that continuous relations induce frame homomorphisms in the same way. These observat
struct the free and forgetful functors.

Definition 3.7.16 (U : **ContRel** $\to$ **Loc**)**.** On objects, U sends a topology X^τ to the frame of opens in τ, which we denote $\hat{\tau}$.

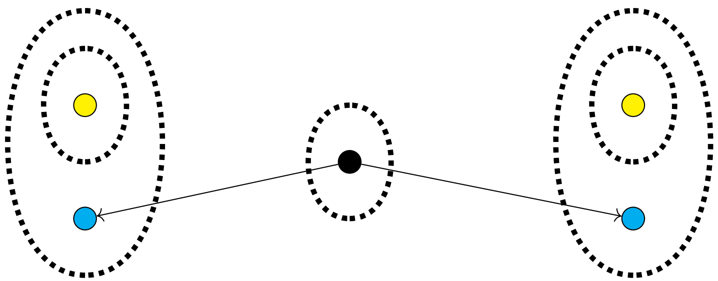
On morphisms $R : X^\tau \to Y^\sigma$, the corresponding partial frame morphism $\hat{\tau} \leftarrow \hat{\sigma}$ (notice the direction reversal for **Loc**), we define to be $\{(U_{\in\sigma}, R^\dagger(U)_{\in\tau}) \mid U \in \sigma\}$. We ascertain that this is (1) a function that is (2) a frame 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 3.7.17 (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 3.3.14, 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.

3.7.5 *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.



3.7.6 *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: $A \xrightarrow{f} \mathcal{P}B \xrightarrow{\mathcal{P}g} \mathcal{P}\mathcal{P}C \xrightarrow{\mu_{\mathcal{P}C}} \mathcal{P}C$. 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):

$$\begin{tikzcd}
\mathbf{Top} \arrow[r, "T"] \arrow[d, bend left] \arrow[d, phantom, "\dashv"] & \mathbf{Top} \arrow[d, bend left] \arrow[d, phantom, "\dashv"] \\
\mathbf{Set} \arrow[r, "\mathcal{P}"] \arrow[u, bend left] & \mathbf{Set} \arrow[u, bend left]
\end{tikzcd}$$

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 rea
isation of the notion of relations to accommodate topologies. As a consequence of this condition, we may observe that discrete to
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-el
We know from Proposition **??** that the continuous relations $X^\tau \to \bullet$ are precisely the open sets of τ, which correspond to continu
$X^\tau \to \mathbb{S}$, and $\mathbb{S} \neq \mathbf{2}$.

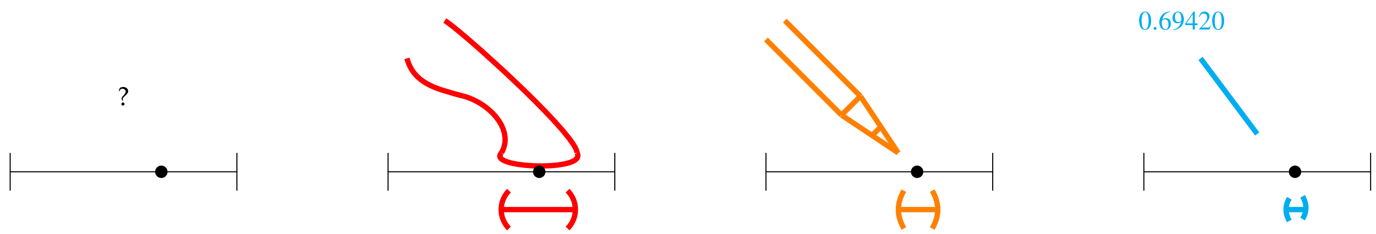
3.7.7 *Where is the topology coming from?*

It is category-theoretically natural to ask whether **ContRel** is "giving topology to relations" or "powering up topologies with relat
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 diffic
in the literature. However, we do have a free-forgetful adjunction between **ContRel** and **Rel**, and if we focus on this, it *is* possible
is coming from with enough machinery; here is one such sketch. Observe that the forgetful functor looks like it could be a kind o
the fibre over any set A in **Rel** correspond to all possible topologies on A. Moreover, these topologies may be partially ordered by
(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 c
crucial observation is that if R is continuous between τ and σ, then R will be continuous for any finer topology in the domain, τ ≤
the codomain $\sigma' \leq \sigma$; that is, the fibre over R displays a boolean-valued profunctor between preorders. So **ContRel** can be viewe
by a functor $\mathbf{Rel} \to \mathcal{C}$, where $\mathcal{C}$ is a category with preorders for objects and boolean-enriched profunctors as morphisms, and the
sending sets in **Rel** to preorders of all possible topologies, and relations to profunctors. I have deliberately left this a sketch becau
something so simple in such a complex way.

3.7.8 *Why are continuous relations worth the trouble?*

I'll have to refer you back to the introduction of this section. In short, because the opens of topological spaces crudely model how
points of a topological space crudely model instances of concepts. Why this is so is best demonstrated by an illustrated example.

Points in space are a mathematical fiction. Useful, but a fiction. Suppose we have a point on a unit interval. Consider how
where this point is. We could point at it with a pudgy appendage, or the tip of a pencil, or give some finite decimal approximatio



But in each case we are only speaking of a vicinity, a neighbourhood, an *open set in the borel basis of the reals* that contains the point. Identifying a true point on a real line requires an infinite intersection of open balls of decreasing radius; an infinite process of pointing again and again, which nobody has the time to do. In the same way, most language outside of mathematics is only capable of offering successively finer, finite approximations to whatever it is that occurs in the mind or in reality.

MAYBE THAT EXPLAINS THE ASYMMETRY OF WHY TESTS ARE OPEN SETS, BUT WHY ARE STATES ALLOWED TO BE ARBITRARY SUBSETS? Because 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 by boundedly finite language; 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.

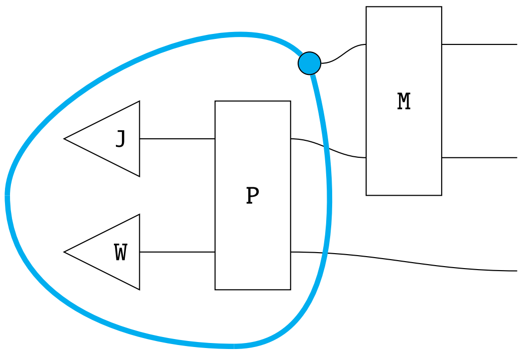
TOP IS SYMMETRIC MONOIDAL CLOSED WITH RESPECT TO PRODUCT, WHY DIDN'T YOU JUST WORK THERE FROM THE START? Because **Top** is cartesian monoidal, which in particular means that there is only one test (the map into the terminal singleton topology), and worse, all states are tensor-separable. The latter fact means that we cannot reason natively in diagrams about correlated states, which are extremely useful representing entangled quantum states [dodo], and for reasoning about spatial relations [talkspace]. I'll briefly explain the gist of the analogy in prose because it is already presented formally in the cited works and elaborated in [bobcomp]. The Fregean notion of compositionality is roughly that to know a composite system is equivalent to knowing all of its parts, and diagrammatically this amounts to tensor-separability, which arises as a consequence of cartesian monoidality. Schrödinger suggests an alternative of compositionality via a lesson from entangled states in quantum mechanics: *perfect knowledge of the whole does not entail perfect knowledge of the parts.* Let's say we have information about a composite system if we can restrict the range of possible outcomes; this is the case for the bell-state, where we know that there is an even chance both qubits measure up or both measure down, and we can rule out mismatched measurements. However, discarding one entangled qubit from a bell-state means we only know that the remaining qubit has a 50/50 of measuring up or down, which is the minimal state of information we can have about a qubit. So we have a case where we can know things about the whole, but nothing about its parts. A more familiar example from everyday life is if I ask you to imagine a `cup on a table in a room`. There are many ways to envision or realise this scenario in your mind's eye, all drawn from a restricted set of permissable positions of the cup and the table in some room. The spatial locations of the cup and table are entangled, in that you can only consider the positions of both together. If you discard either the cup or the table from your memory, there are no restrictions about where the other object could be in the room; that is, the meaning of the utterance is not localised in any of the parts, it resides in the entangled whole.

4

Sketches of the shape of language

4.1 *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:



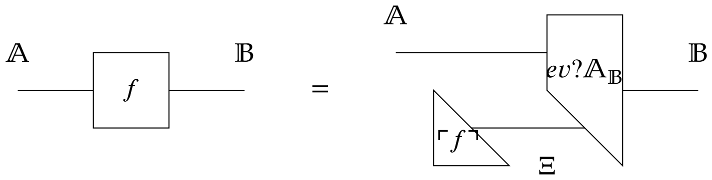
A mathematical modelling problem for semanticists arises when anything can be a noun wire. The problem at hand is finding the right mathematical setting to interpret and calculate with such lassos. In principle, any meaningful (possibly composite) part of text can be referred to as if it were a noun. 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

PECIAL IS THAT ON REAL COMPUTERS, THEY ARE SPECIFIED BY CODE. Code is just another format of data. Programs that are equiv-
y have many different implementations in code: for example, there are many sorting algorithms, though all of them map the
nversely, every possible program in a process theory of programs must have some implementation as code. Diagrammatically, we
his: 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
denote Ξ going forward) as an additional input, and for every possible program $f : \mathbb{A} \to \mathbb{B}$, there exists a state $\ulcorner f \urcorner : I \to \Xi$

$$\forall \mathbb{A}, \mathbb{B} \in Ob(C) \exists ev\{\mathbb{A}_{\mathbb{B}} : \mathbb{A} \otimes \Xi \to \mathbb{B} \forall f : \mathbb{A} \exists \ulcorner f \urcorner : I \to \Xi \tag{4.1}$$



(4.2)

erises computers as code-evaluators, provides a plan of attack for the semantic modelling problem of entification: if we take
monoidal computer, we have restricted the candidate symmetric monoidal categories to model text-circuits in a way that allows

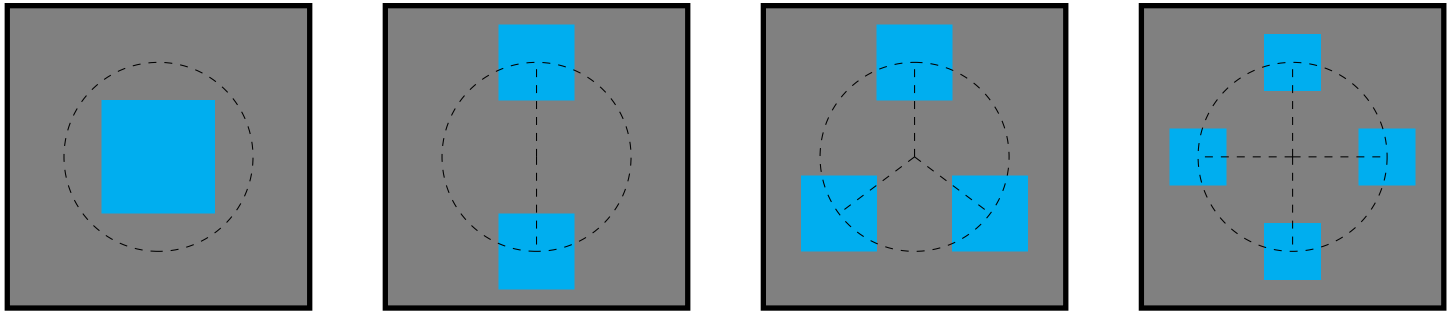
we could have made is that since computers really just manipulate code, every data format is a kind of restricted form of the
to be a mathematical consequence of the above equation (and the presence of a few other operations such as copy and compare
ra), demonstrated in Pavlovic's forthcoming monoidal computer book [], itself a crystallisation of three monoidal computer pa-
Cockett's work on Turing categories []. Both approaches to a categorical formulation of computability theory share the common
losure (monoidal closure in the case of monoidal computer and exponentiation in Turing categories) where rather than having
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
restriction categories, which generalise partial functions. I work in Pavlovics' formalism because I prefer string diagrams to com-

onclude that any programming language is a model for text circuits, using the code data format as the noun wire. Since semanti-
the following construction.
the open unit square model the category relations between countable sets equipped with a code object). Using the open unit
code object, there is a subcategory of **ContRel** which behaves as the category of countable sets and relations equipped with

hrough any countable set X). For any countable set X, the open unit square ▦ has a sticky spider that splits through $X^\star$.

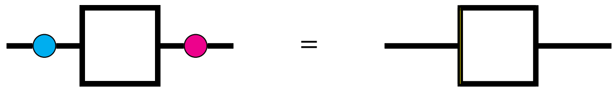
We'll assume the sticky-spiders to be mereologies, so that cores and halos agree. Then we only have to highlight the copiable
axis-aligned open squares evenly along them, one for each element of X. The centres of the open squares lie on the circumfer-

ence of the circle, and we may shrink each square as needed to fit all of them.

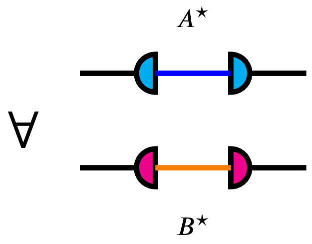


□

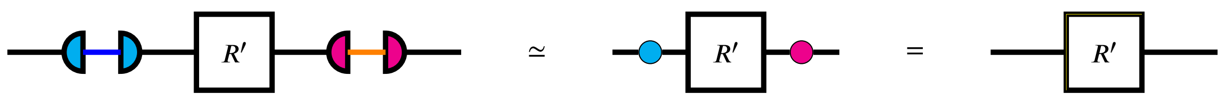
Definition 4.1.4 (Morphism of sticky spiders)**.** A morphism between sticky spiders is any morphism that satisfies the following equation.



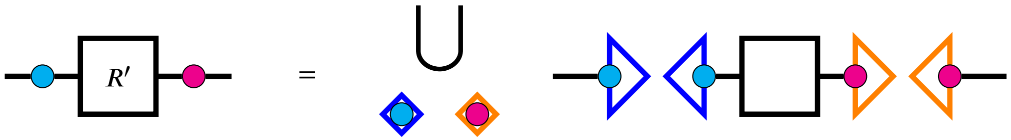
Proposition 4.1.5 (Morphisms of sticky spiders encode relations)**.** For arbitrary split idempotents through $A^\star$ and $B^\star$, the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \to B$.

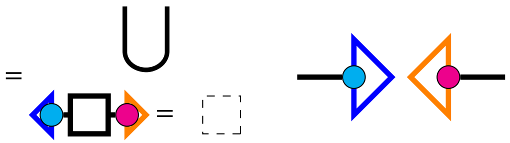


$: \mathbf{Rel}(A, B) \ni R \leftrightarrow$

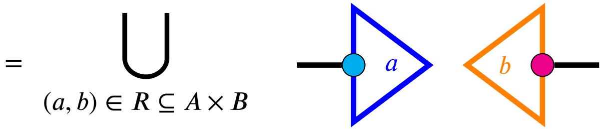


(⇐) : Every morphism of sticky spiders corresponds to a relation between sets.

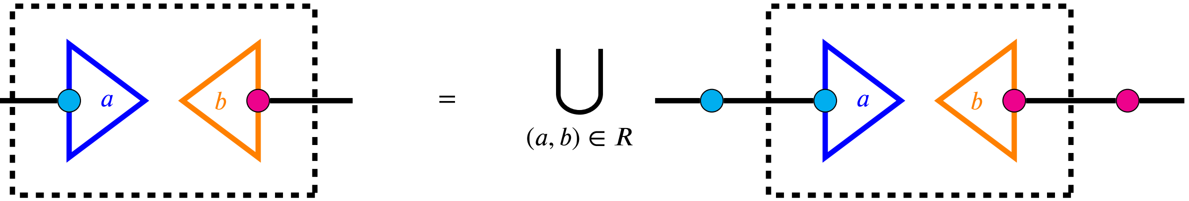




Since (co)copiables are distinct, we may uniquely reindex as:

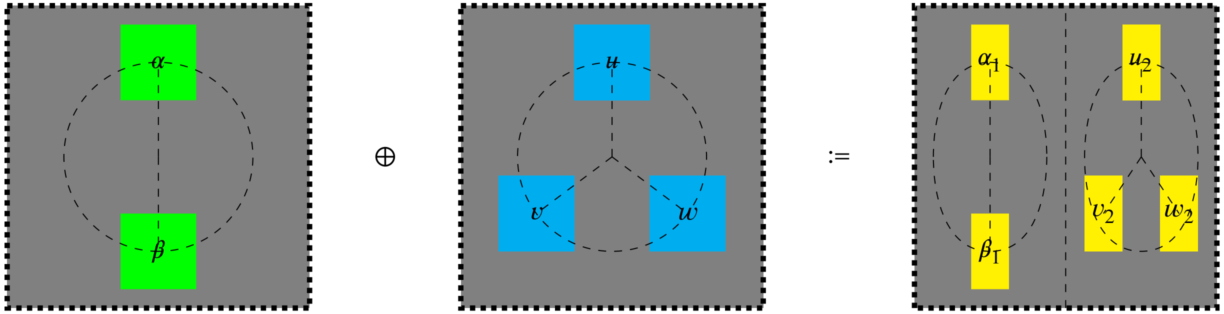


By idempotence of (co)copiables, every relation $R \subseteq A \times B$ corresponds to a morphism of sticky spiders.



□

Construction 4.1.6 (Representing sets in their various guises within ▦)**.** We can represent the direct sum of two ▦-representations of sets as follows.

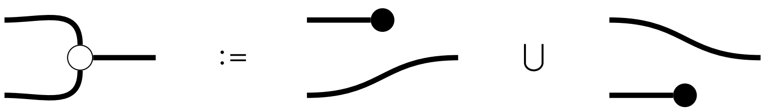


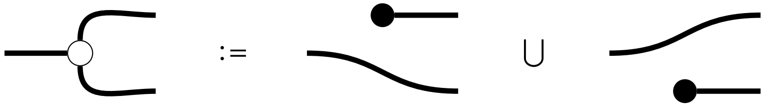
The important bit of technology is the homeomorphism that losslessly squishes the whole unit square into one half of the unit square. The decompressions are partial continuous functions, with domain restricted to the appropriate half of the unit square.

$(x, y) \mapsto (\frac{x}{2}, y)$ $\qquad$ $(x, y) \mapsto (\frac{x+1}{2}, y)$ $\qquad$ $(x, y)|_{x<\frac{1}{2}} \mapsto (2x, y)$ $\qquad$ $(x, y)|_{x>\frac{1}{2}} \mapsto (2x-1, y)$

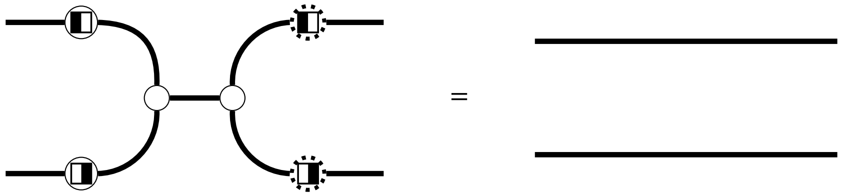
We express the ability of these relations to encode and decode the unit square in just either half by the following graphical equations.

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**.

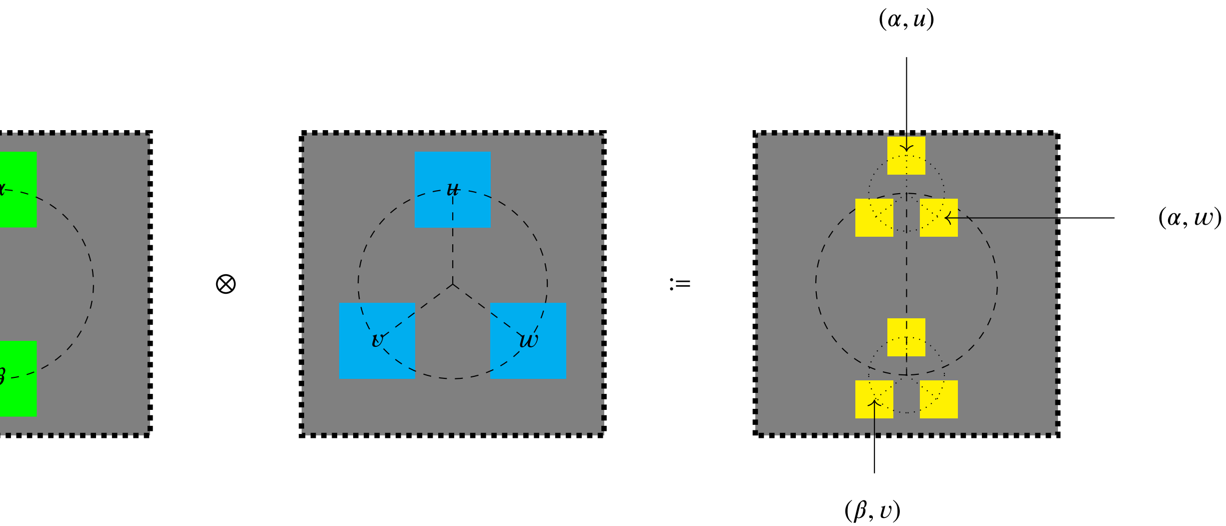




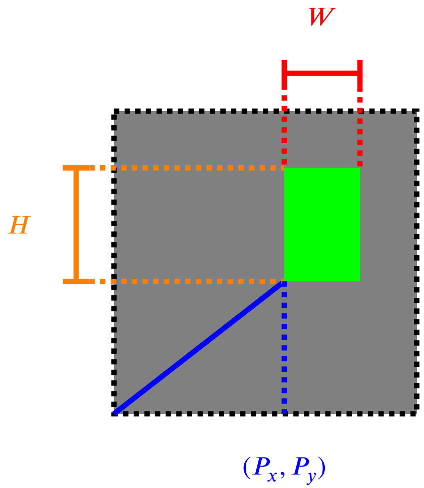
The following equation tells us that we can take any two representations in ▦, put them into a single copy of ▦, and take them out again. Banach and Tarski would



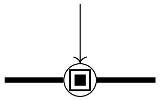
of representations by placing copies of B in each of the open boxes of A.



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.

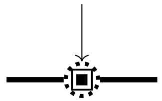


(W, H, P_x, P_y)



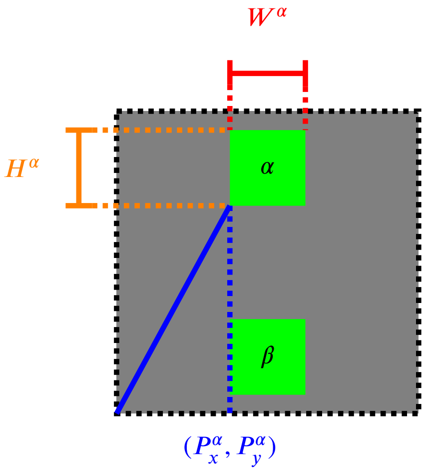
$$(x, y) \mapsto (W x + P_x, H y + P_y)$$

(W, H, P_x, P_y)



$$(x, y)|_{\in(P_x, P_x+W)\times(P_y, P_y+H)} \mapsto \left(\frac{x-P_y}{W}, \frac{y-P_y}{H}\right)$$

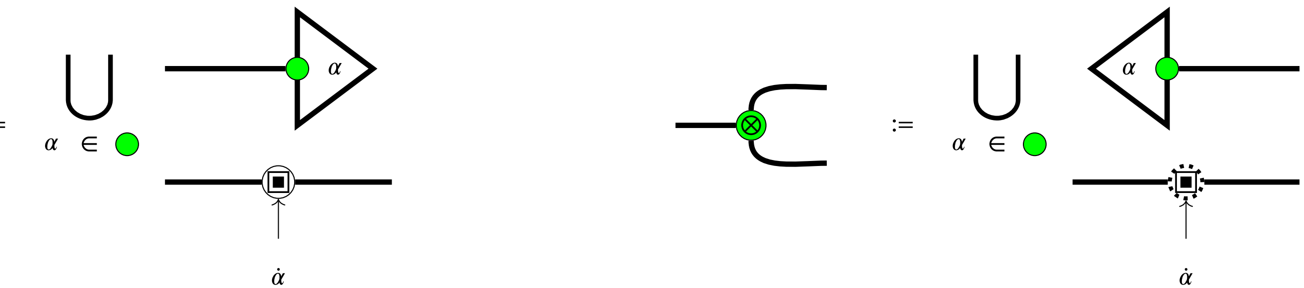
Now, for every representation of a set in ▦ by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a he parameters of the open box, which we notate with a dot above the name of the element.

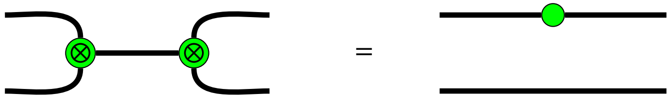


$$\dot{\alpha} = (W^\alpha, H^\alpha, P_x^\alpha, P_y^\alpha)$$

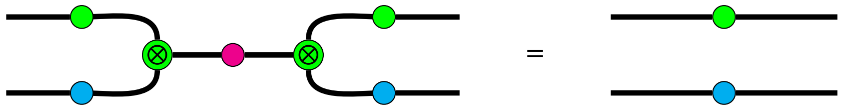
$$\dot{\beta} = (W^\beta, H^\beta, P_x^\beta, P_y^\beta)$$

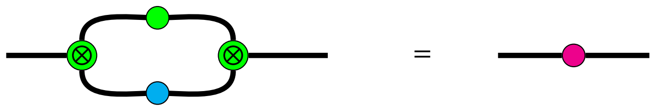
he left" relation $\_ \to X \otimes \_$ and its corresponding cotensor.



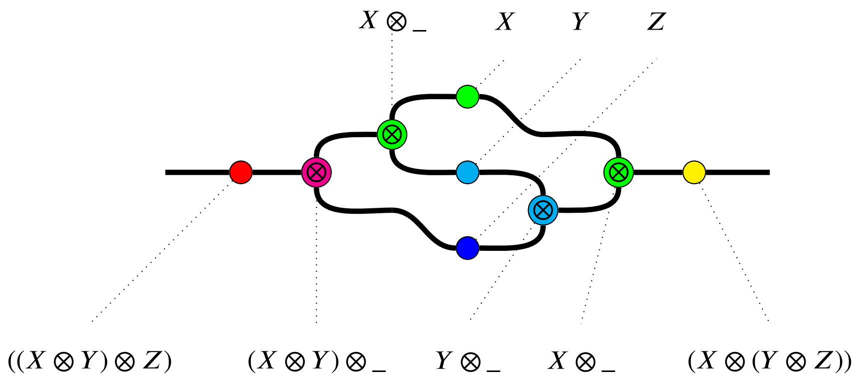


The tensor and cotensor behave as we expect from proof nets for monoidal categories.



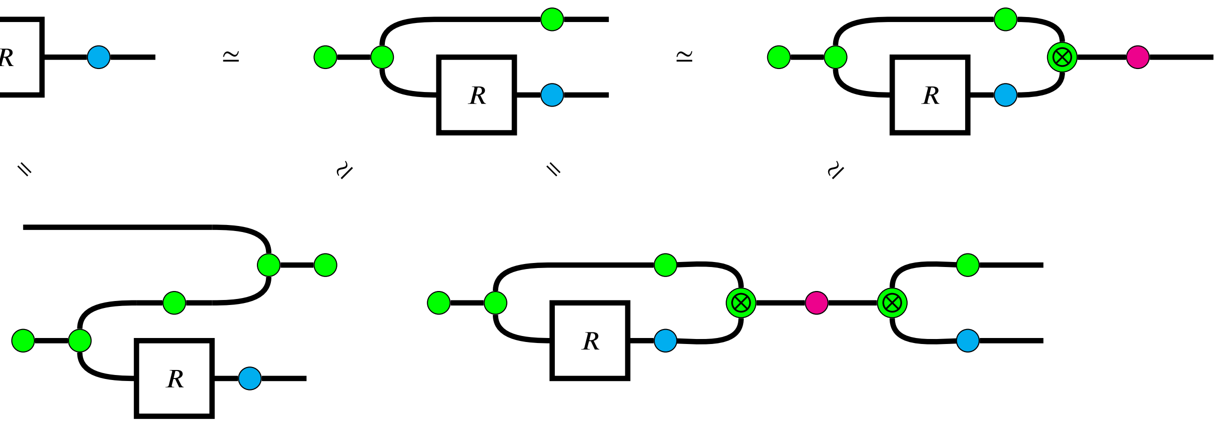


And by construction, the (co)tensors and (co)pluses interact as we expect, and they come with all the natural isomorphisms between representations we expect. For example, below we exhibit an explicit associator natural isomorphism between representations.

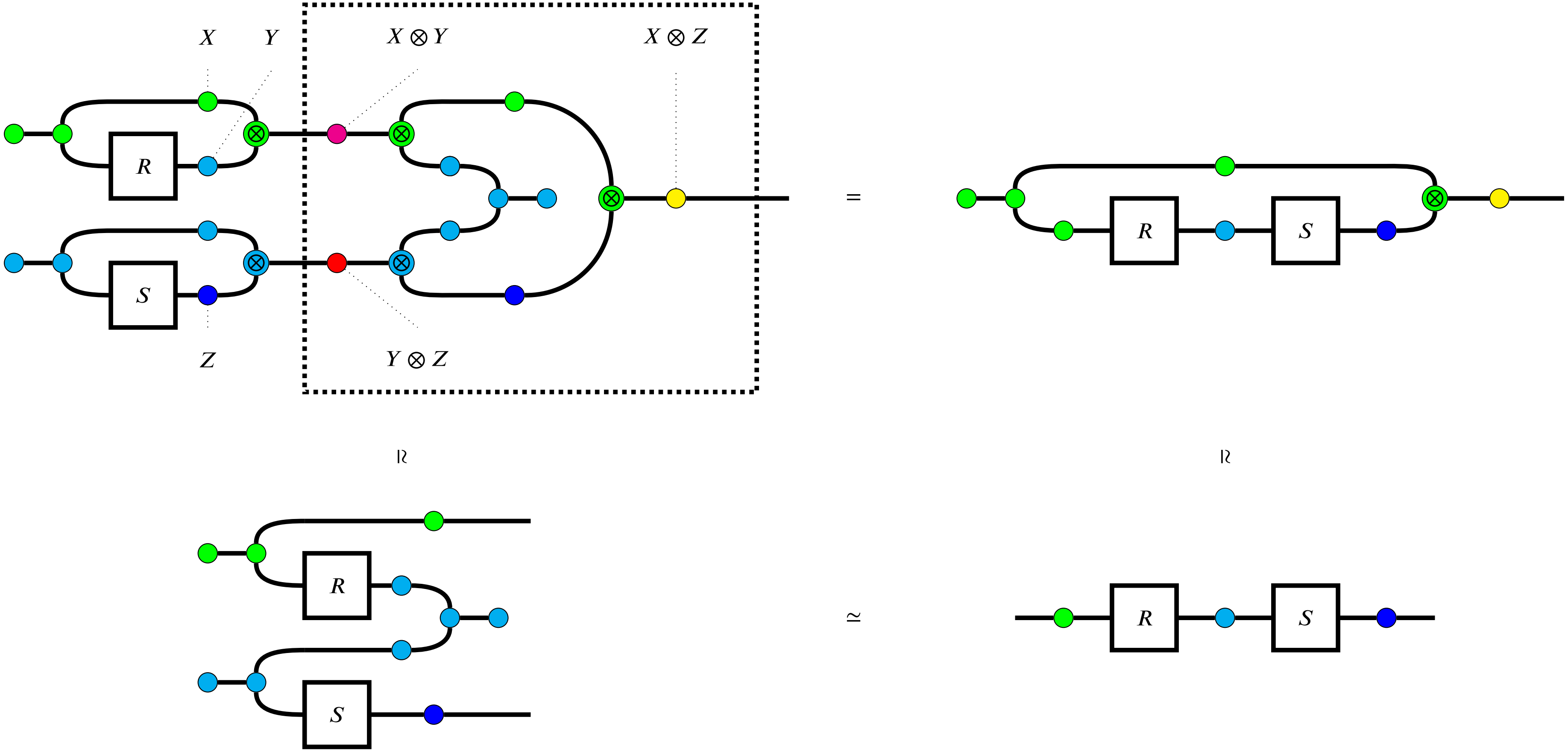


Construction 4.1.7 (Representing relations between sets and their composition within ▦)**.** With all the above, we can establish a special kind of process-state duality;

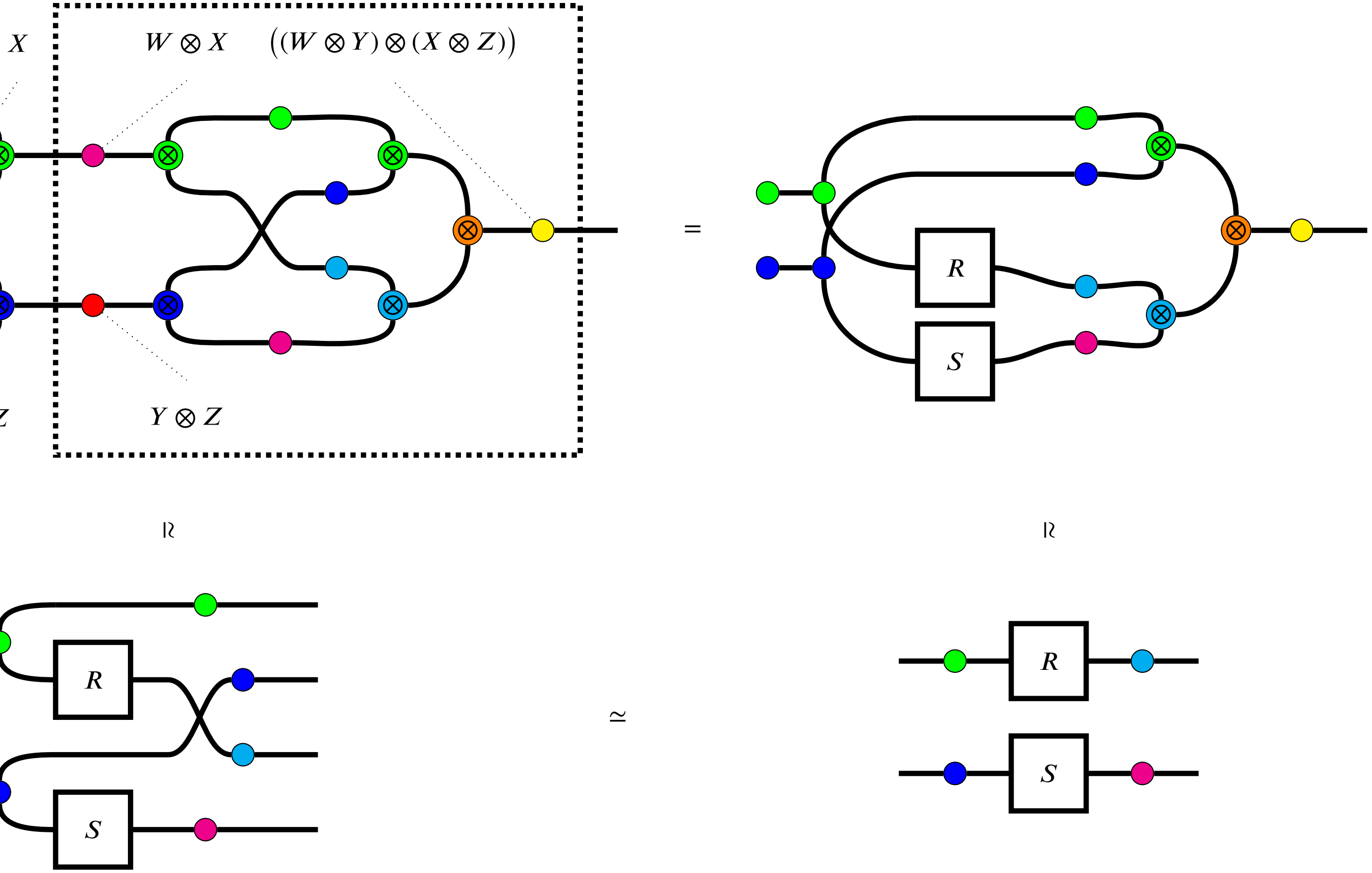
to states of ▦, up to the representation scheme we have chosen.



Moreover, we have continuous relations that perform sequental composition of relations.



rallel composition of relations by tensors.



4.2 *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 level of analysis 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.

4.2.1 *Temperature and colour: the Planckian Locus*

Example 4.2.1 (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 4.2.1 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 4.2.2 (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 4.2.3 (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.

4.2.2 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`.

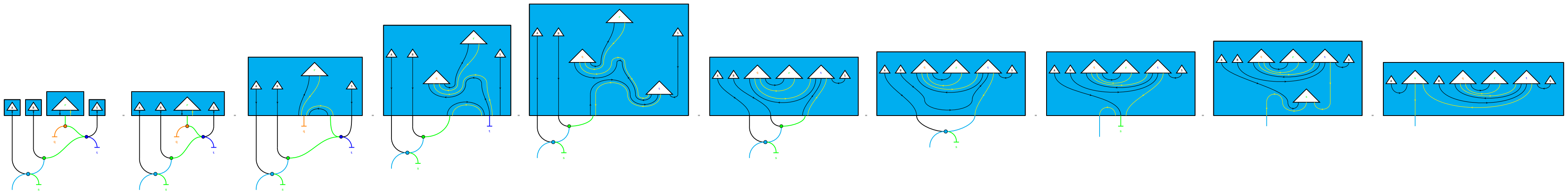
Now I'll try to demonstrate by example that cofunctors between text circuits do all of the things we have asked for. The components of text circuits serve as an algebraic basis for dynamic conceptual complexes, while the cofunctor handles partial structuring of one conceptual domain in terms of another.

Example 4.2.4 (`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 **??**.

4.3 *How do we communicate using language?*

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



Here are some naïve observations on the nature of speaking and listening. Let's suppose that a speaker, Preube, wants to communicate a thought to Fondo. Preube and Fondo cooperate to achieve the miracle; Preube encodes his thoughts – a structure that isn't a one-dimensional string of symbols – into a one-dimensional string of symbols. And then Fondo does the reverse, turning a one-dimensional string of symbols into a thought-structure like that of Preube's. It may still be that Preube and Fondo have radically different internal conceptions of what `FLOWERS` or `GIVING` or `BEETLES IN BOXES` are, but that is alright: we only care that the *interacting structure* of the thought-relations 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 Preube must supply decisions about phrasing a thought in the process of speaking it. At some point at the beginning of an utterance, Preube 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 Preube 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 Fondo 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 Fondo 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 Preube and Fondo 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 4.3.1 (`Alice likes flowers that Bob gives Claire.`)**.** Let's say Preube is using a context-free grammar to produce sentences, and Fondo a pregroup grammar.

Example 4.3.2 (`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.

4.3.1 Discrete Monoidal Fibrations

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[5] : 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*.

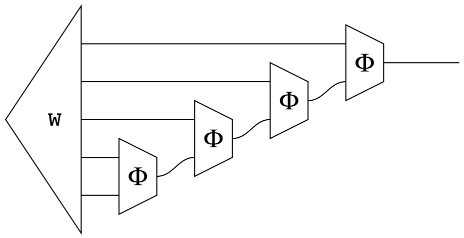
4.3.2 Strictified diagrams for monoidal categories

The crux of the issue sketched in Figure **??** 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.

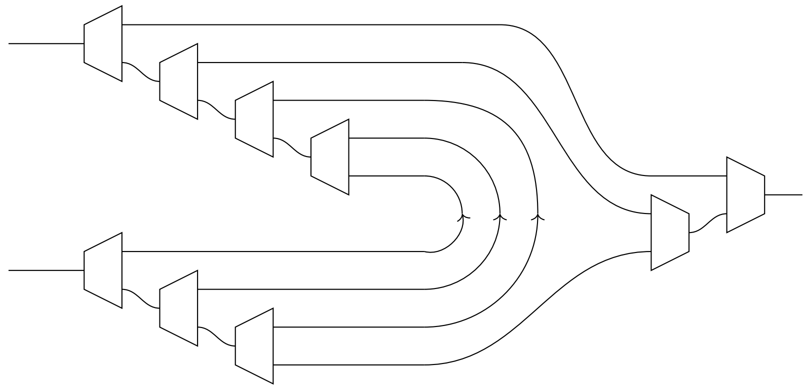
We are seeking some way to algebraically group or bracket together pregroup types that arise from a single word, in a distinguished way from concatenation-as-tensor. In this way we can preserve the structure of pregroup-sequent proofs: grouping indicates a node in the proof-tree, while tensor indicates 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 4.3.7 (Pregroups with bracketing)**.** Where $\mathcal{P}$ is a monoidal category generated by pregroup states and (directed) cups, we define pregroups-with-bracketing as subcategory of the *free* strictification $\overline{\mathcal{P}}$, which consists of all the generators of the strictification $\overline{\mathcal{P}}$ as given by Definition **??**, but none of the additional equations. The subcategory is constructed from the following generators:

- For each pregroup state $\texttt{w} : I \to \bigotimes_i X_i$, a strictified state generator $\texttt{w} : I \to (\cdots((X_1 \otimes X_2) \otimes \cdots X_i) \cdots)$ with left-nested syntactic tensors. To illustrate, a state with 5 wires would correspond to a generator as follows:



- 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 directed 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 fully detensors, applies the directed cup, and then retensors. Diagrammatically, this amounts to asking for generators that look like the following, that mimick a single proof step.



Construction 4.3.7 solves the (co)domain assignment problem by organising the data as a monoidal cofunctor.

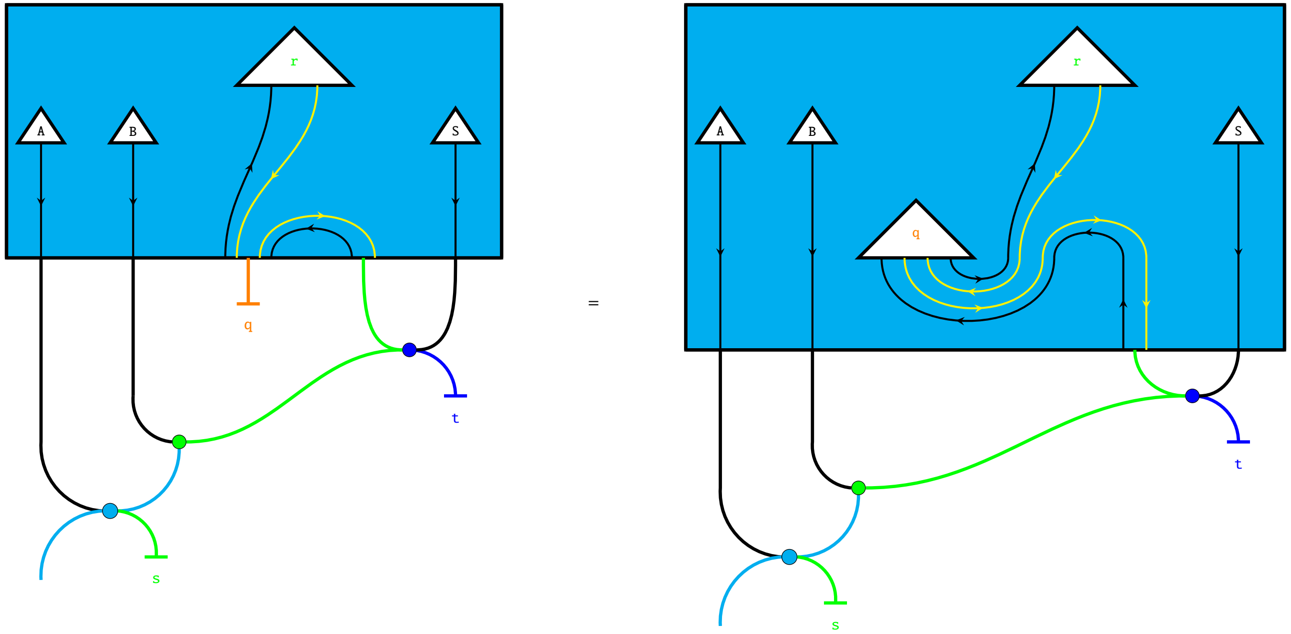
4.3.3 *Monoidal Cofunctors*

Pregroups by themselves are thin rigid autonomous categories, meaning there is at most one morphism between any pair of objects and that there are directed cups and caps in addition to plain monoidal structure. Thinness raises big issues for compositional semantics. Firstly, contra the aims of DisCoCat, all grammatically correct sentences correspond to states on the sentence type, which must all be equal by thinness, hence *no functor* from the pregroup grammar to any other category can distinguish different sentences. One workaround to save claims of functoriality is to construct a free rigid autonomous category from a collection of word-states as generators, i.e. to start from *pregroup diagrams for a pregroup grammar* rather than just pregroups themselves, and these pregroup diagrams are the usual domain of a semantic interpretation functor into $\mathbf{FinVect}^{\otimes}$ or other strongly compact closed categories like variants of **Rel**. All we're observing here is that for a given pregroup grammar, there is a cofunctor from pregroup diagrams of that grammar to the pregroup grammar as a thin rigid autonomous category, presentable as a span of monoidal functors with the free strictification as the apex.

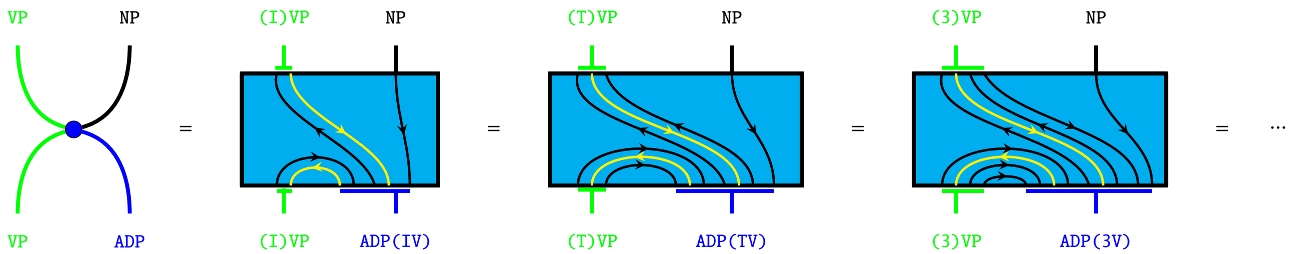
Proposition 4.3.12 (Construction 4.3.7 yields a discrete monoidal fibration)**.**

4.3.4 Communicative constraints as a cofunctor from productive to parsing grammar

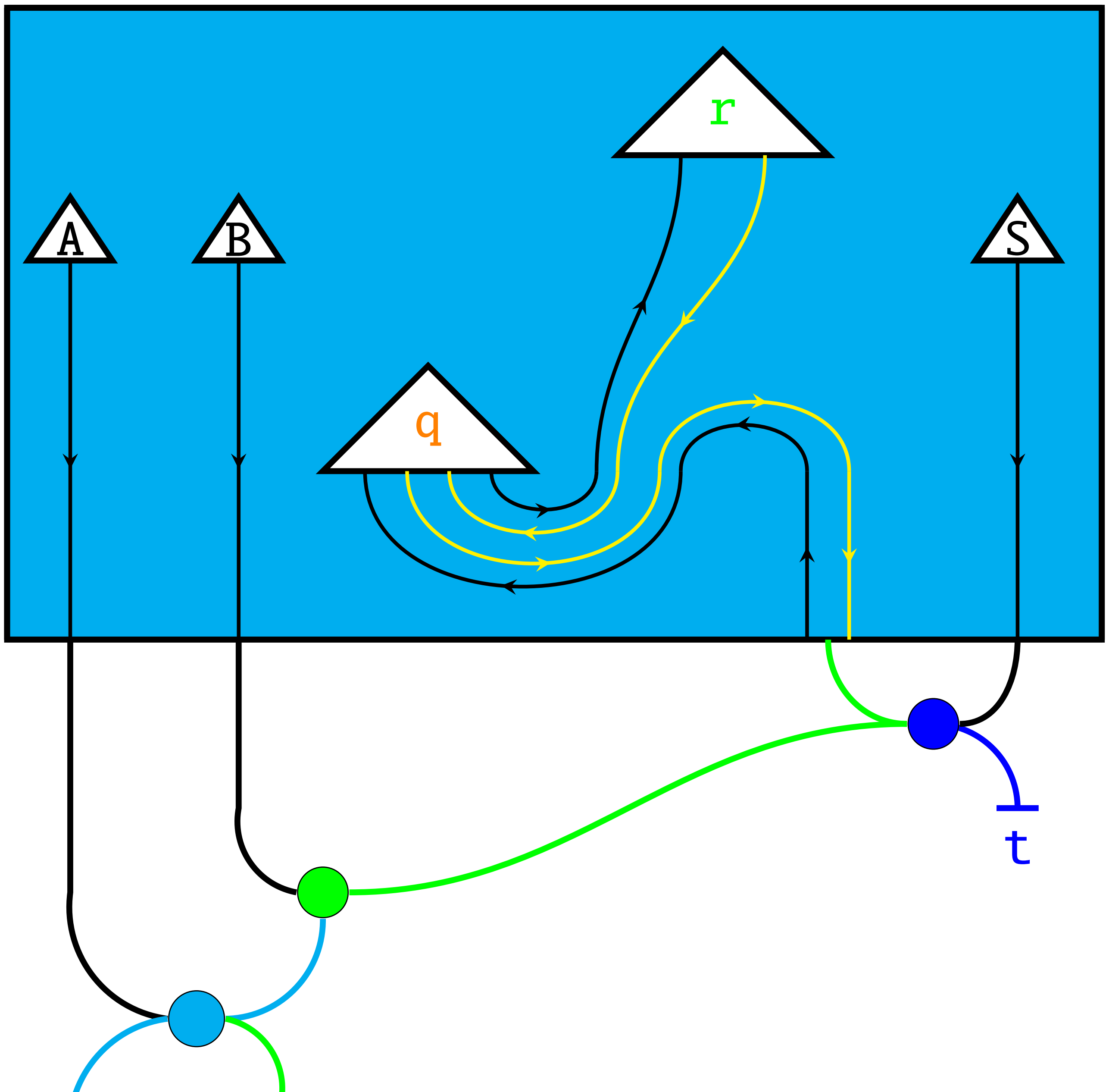
Since the functor is a monoidal discrete fibration, it introduces the appropriate choice of `quickly` when we pull the functor-box down, while leaving everything else in parallel alone.

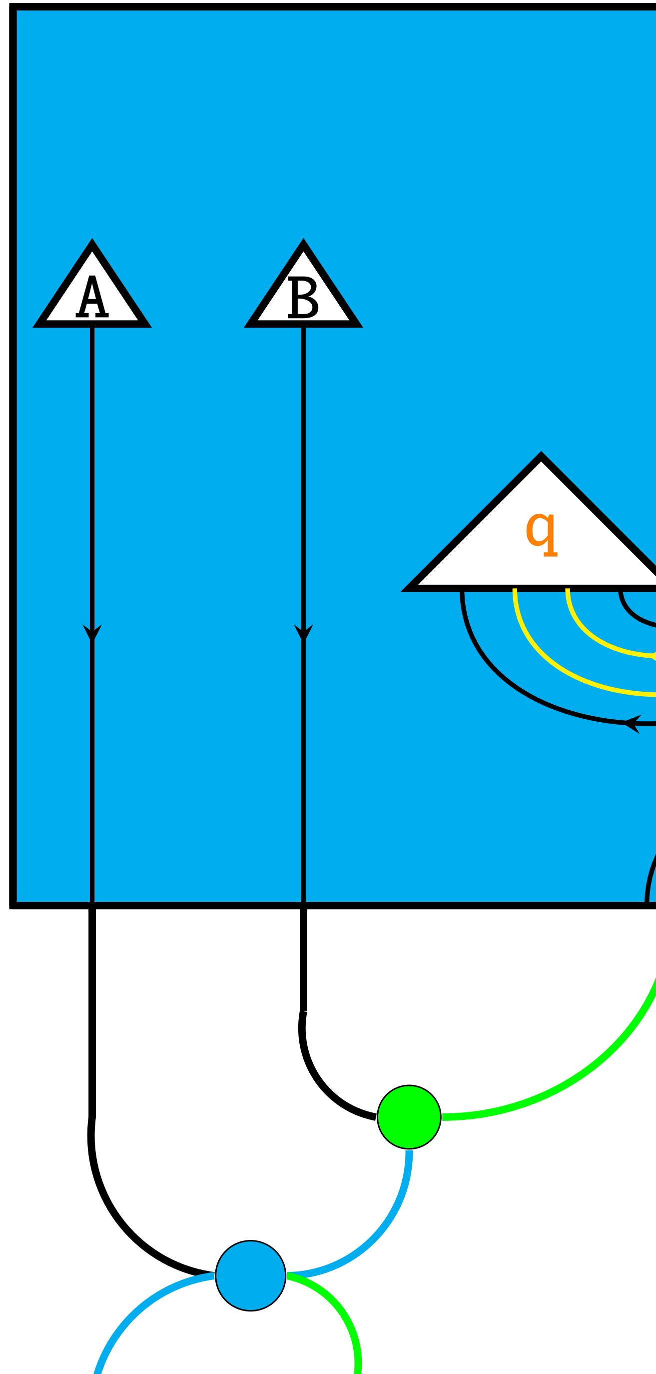


Adpositions can apply for verbs of any noun-arity. We again use the fiber of the functor for bookkeeping by asking it to send all of the following partial pregroup diagrams to the adposition generator. We consider the pregroup typing of a verb of noun-arity $k \geq 1$ to be ${}^{-1}n \cdot s \cdot \underbrace{n^{-1} \cdots n^{-1}}_{(k-1)}$.

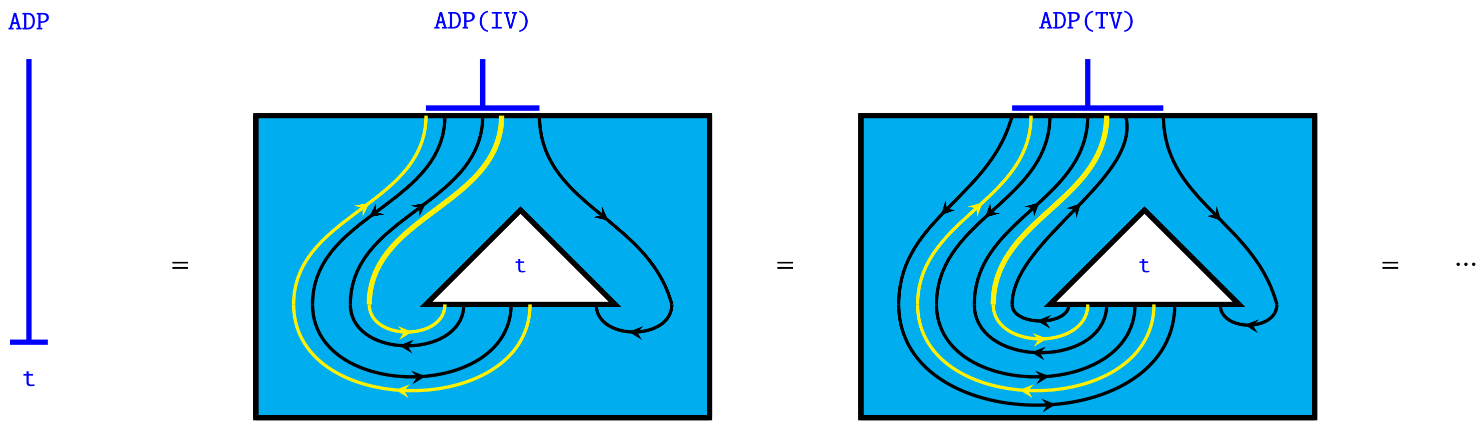


When we pull down the functor-box, the discrete fibration introduces the appropriate choice of diagram from above, corresponding to the intransitive verb case.

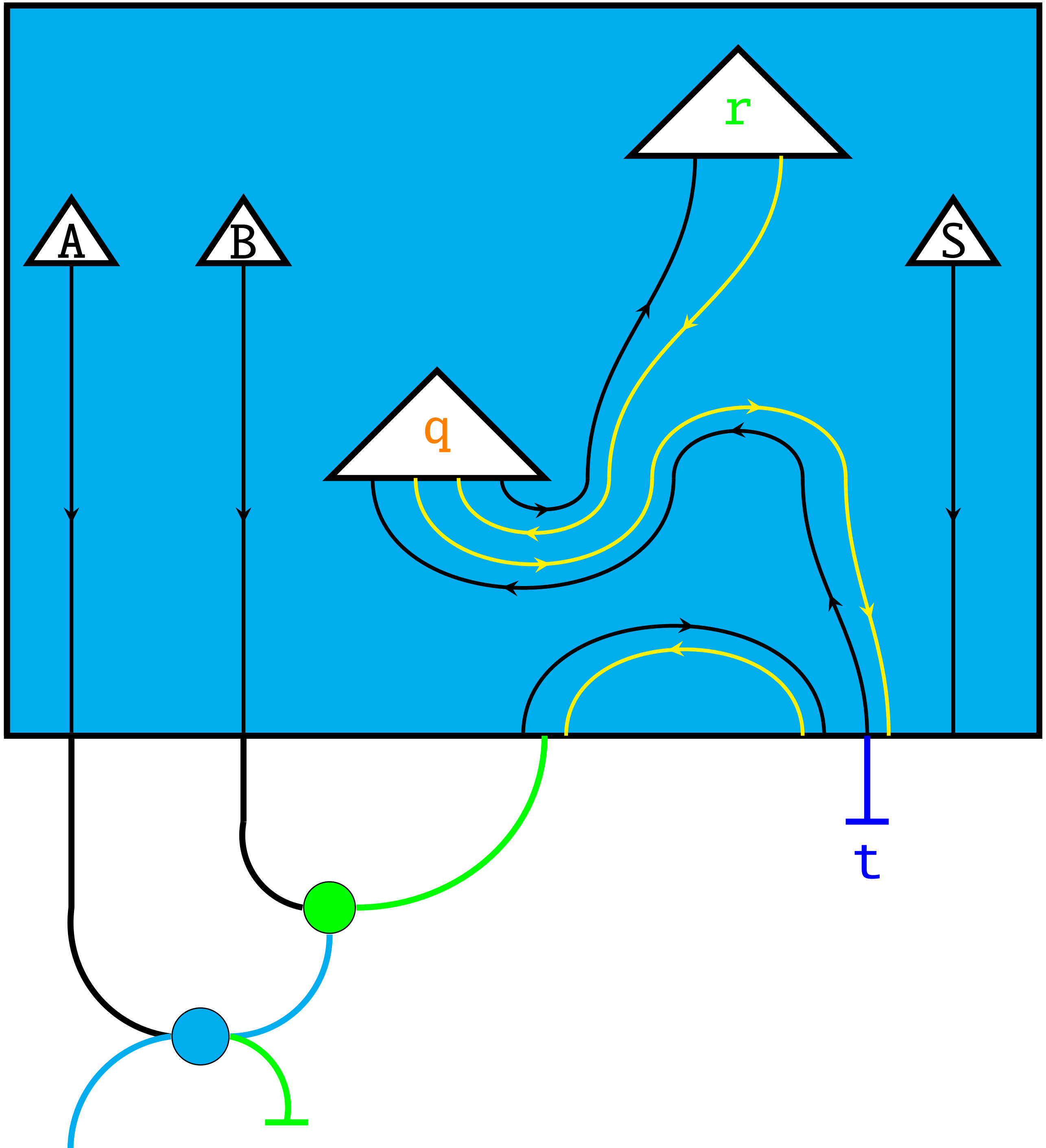


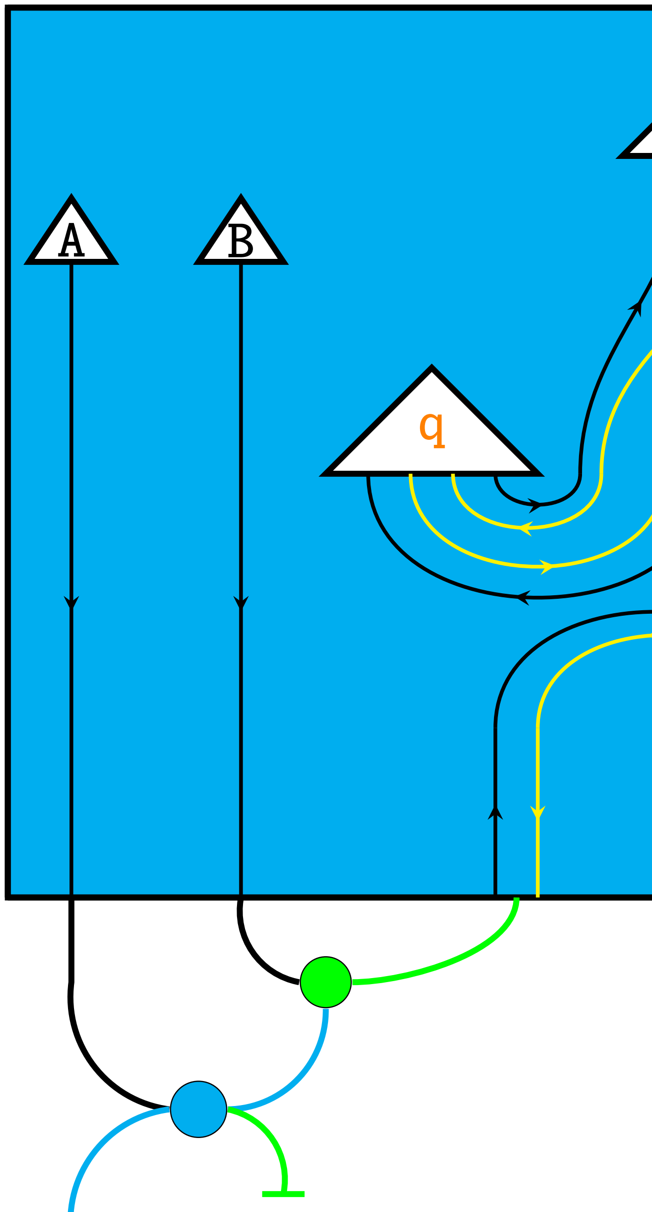


Similarly to `quickly`, we suppose we have a family of processes for the word `to`, one for each noun-arity of verb.

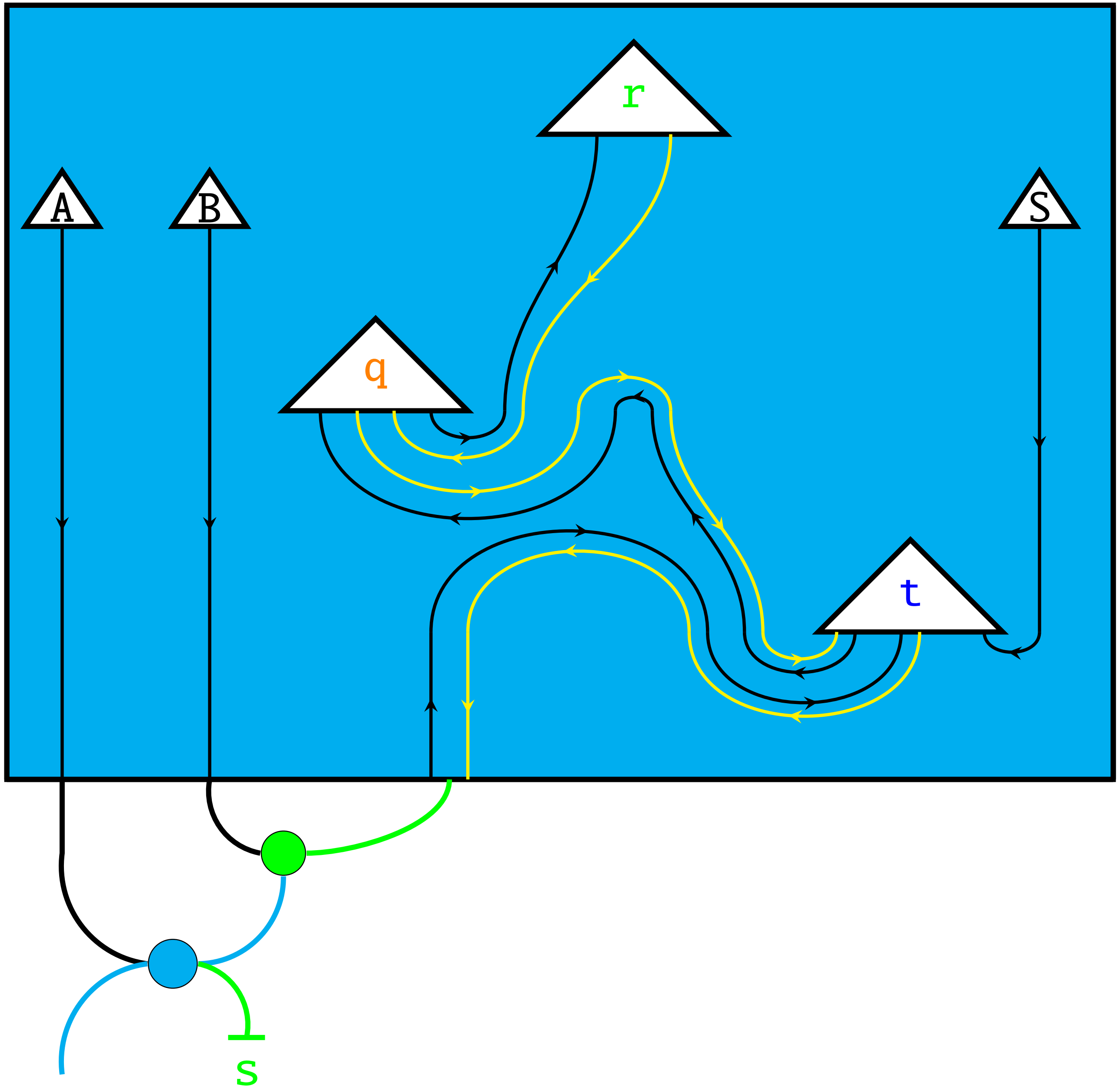


Again the discrete fibration introduces the appropriate choice of `to` when we pull the functor box down.

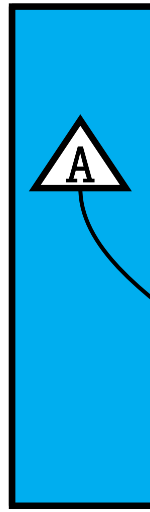




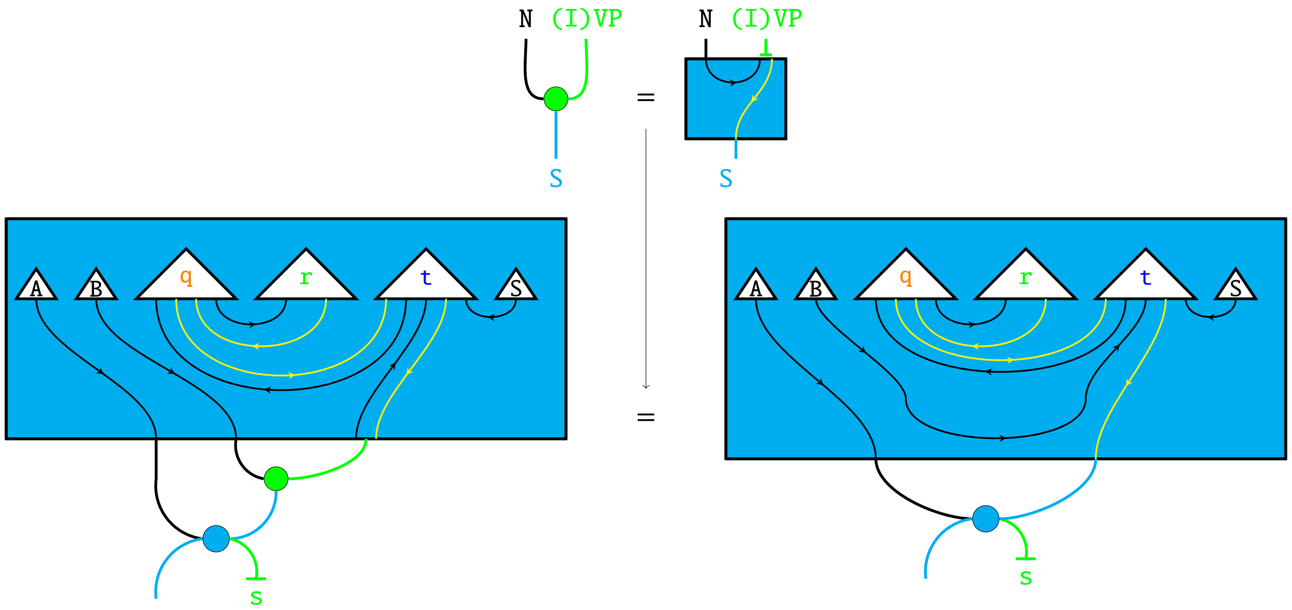
Now we visually simplify the inside of the functor-box by applying yanking equations.



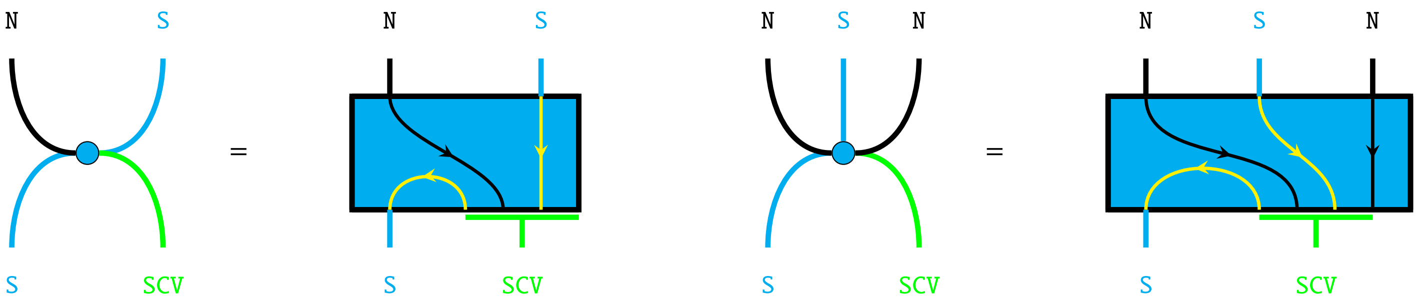
=



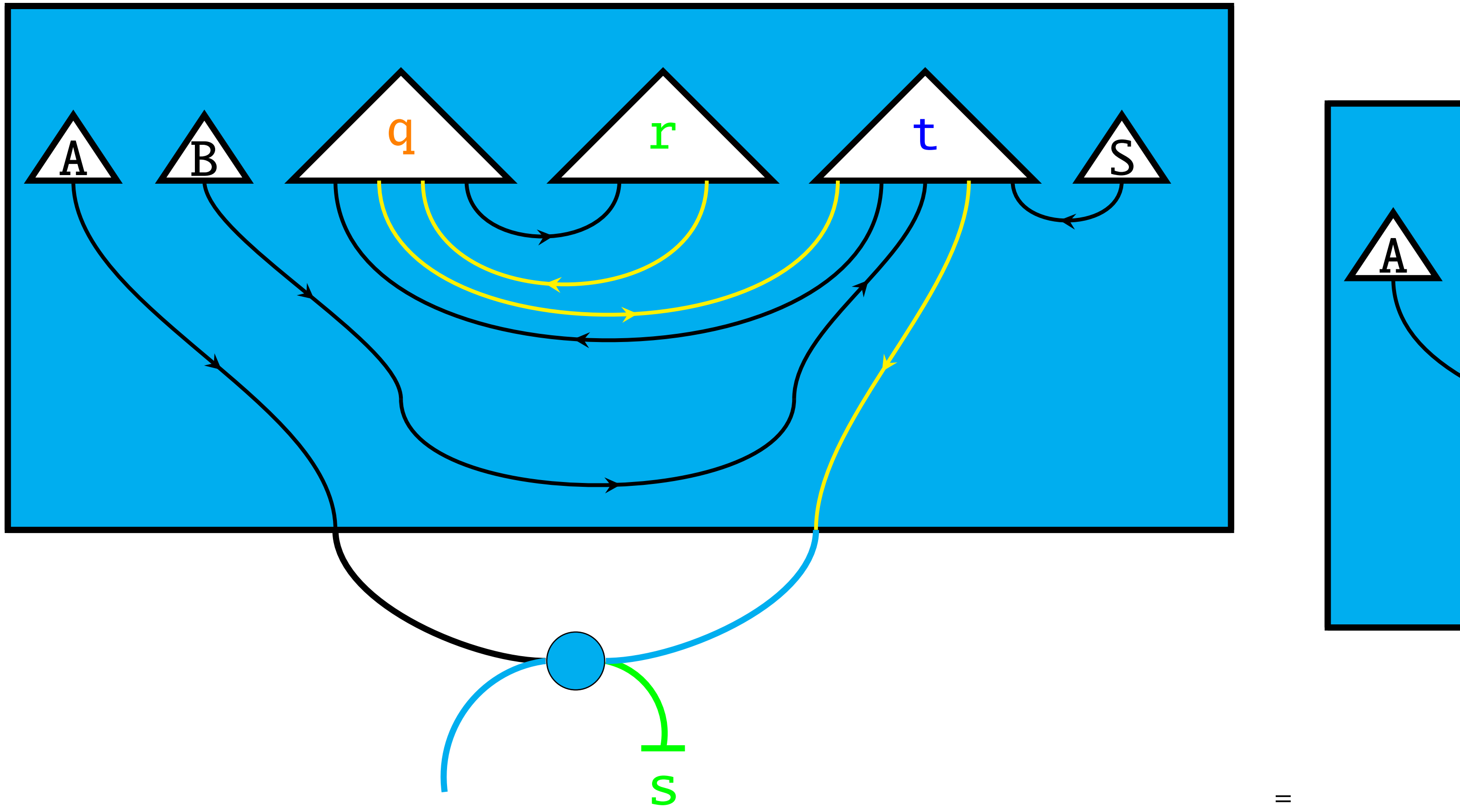
Similarly as before, we can pull the functor-box past the intransitive verb node. There is only one pregroup type ${}^{-1}n \cdot s$ that corresponds to the grammatical category `(I)VP`.



Proceeding similarly, we can pull the functor-box past the sentential-complement-verb node. There are multiple possible pregroup types for `SCV`, depending on how many noun-phrases are taken as arguments in addition to the sentence. For example, in `Alice sees [sentence]`, `sees` returns a sentence after taking a noun to the left and a sentence to the right, so it has pregroup typing ${}^{-1}n \cdot s \cdot s^{-1}$. On the other hand, for something like `Alice tells Bob [sentence]`, `tells` returns a sentence after taking a noun (the teller) to the left, a noun (the tellee) to the left, and a sentence (the story) to the left, so it has a pregroup typing ${}^{-1}n \cdot s \cdot n^{-1} \cdot s^{-1}$. These two instances of sentential-complement-verbs are introduced by different nodes. We can record both of these pregroup typings in the functor by asking for the following:

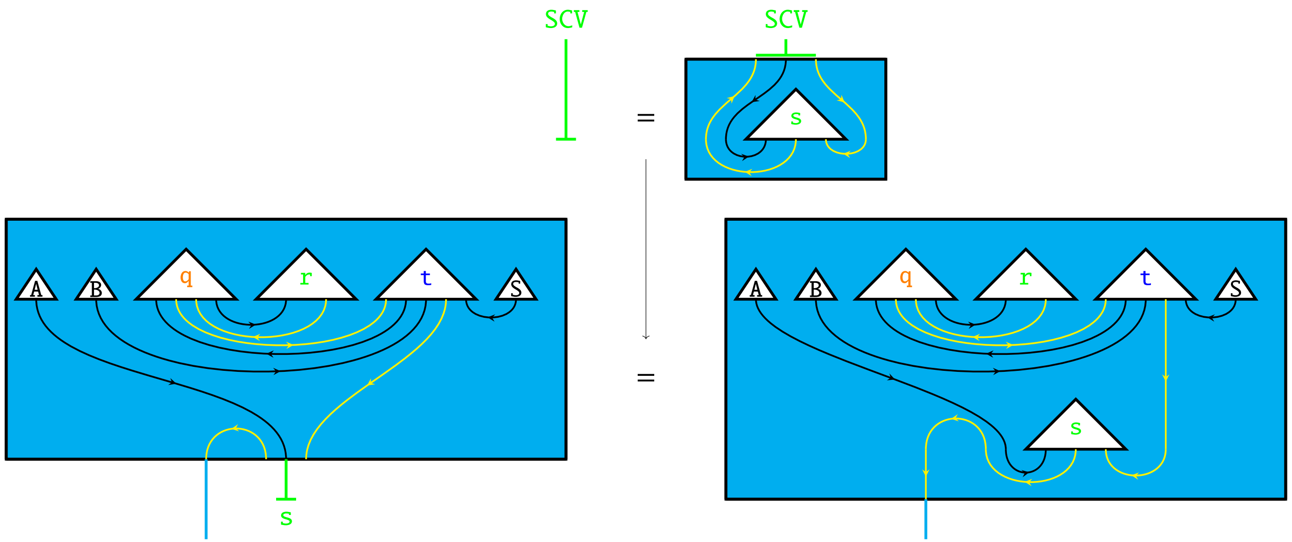


Pulling down the functor box:

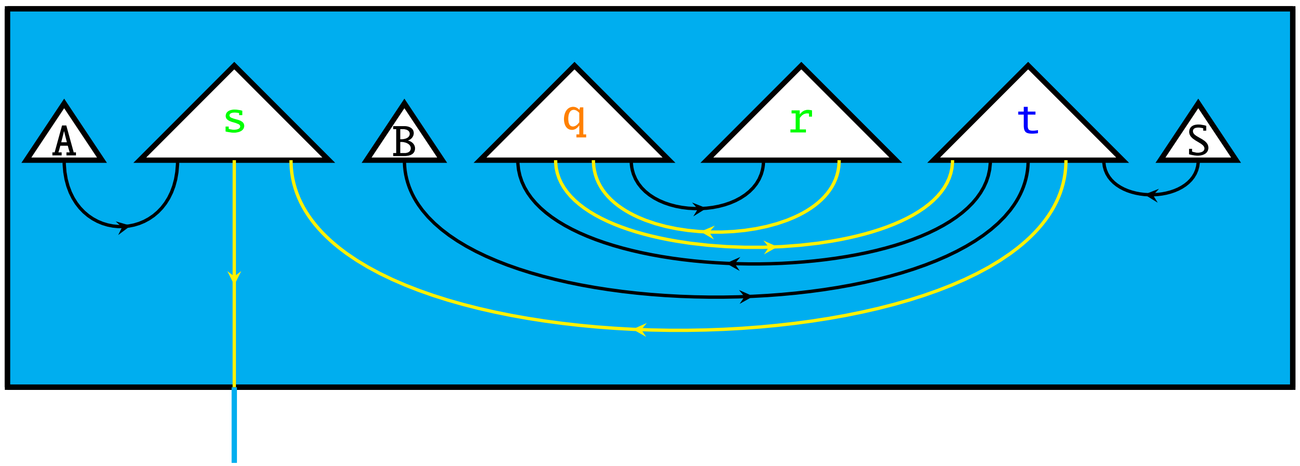


=

As before, we can ask the functor to send an appropriate partial pregroup diagram to the dependent label `sēe`.



Now again we can visually simplify using the yanking equation and isotopies, which obtains a pregroup diagram.



The pregroup diagram corresponds to a particular pregroup proof of the syntactic correctness of the sentence `Alice sees Bob run quickly to school`.

$$\dfrac{\dfrac{\texttt{A} : n \qquad \texttt{s} : {}^{-1}n \cdot s \cdot s^{-1}}{\texttt{A␣s} : s \cdot s^{-1}} \qquad \dfrac{\texttt{B} : n \qquad \dfrac{\dfrac{\dfrac{\texttt{q} : ({}^{-1}n \cdot s) \cdot ({}^{-1}n \cdot s)^{-1} \qquad \texttt{r} : {}^{-1}n \cdot s}{\texttt{q␣r} : {}^{-1}n \cdot s} \qquad \texttt{t} : {}^{-1}({}^{-1}n \cdot s) \cdot ({}^{-1}n \cdot s) \cdot n^{-1}}{\texttt{q␣r␣t} : ({}^{-1}n \cdot s) \cdot n^{-1}} \qquad \texttt{S} : n}{\texttt{q␣r␣t␣S} : {}^{-1}n \cdot s}}{\texttt{B␣q␣r␣t␣S} : s}}{\texttt{A␣s␣B␣q␣r␣t␣S} : s}$$

4.3.5 Relating circuits and CFGs fibrationally

4.3.6 *Where internal wirings come from*

4.3.7 *Discrete monoidal fibrations for grammatical functions*

4.3.8 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 Fondo 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 4.3.13 (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 4.3.14 (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:

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

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:

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

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 4.3.15 (Ambiguous scoping)**.** Consider the following sentence:

`Everyone loves someone`

The sentence is secretly (at least) two, corresponding to two possible parses. The usual reading is (glossed) $\forall x \exists y : \texttt{loves}(x, y)$. 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.

$$\dfrac{\dfrac{\texttt{everyone} : (n \multimap s) \multimap s \qquad \texttt{loves} : n \multimap s \multimapinv n}{\texttt{everyone\textvisiblespace loves} : s \multimapinv n} \qquad \texttt{someone} : (s \multimapinv n) \multimap s}{\texttt{everyone\textvisiblespace loves\textvisiblespace someone} : s}$$

$$\dfrac{\texttt{everyone} : (n \multimap s) \multimap s \qquad \dfrac{\texttt{loves} : n \multimap s \multimapinv n \qquad \texttt{someone} : (s \multimapinv n) \multimap s}{\texttt{loves\textvisiblespace someone} : n \multimap s}}{\texttt{everyone\textvisiblespace loves\textvisiblespace someone} : s}$$

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:

$$[\![\texttt{everyone}]\!] = \lambda(\lambda x.V(x)).\forall x : V(x) \tag{4.3}$$

$$[\![\texttt{loves}]\!] = \lambda x \lambda y.\texttt{loves}(x, y) \tag{4.4}$$

$$[\![\texttt{someone}]\!] = \lambda(\lambda y.V(y)).\exists y : V(y) \tag{4.5}$$

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.

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

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

Example 4.3.16. (grammar pieces – as simple as it gets) `Bob runs.` `Bob quickly runs.` `Bob drinks beer.` `Bob quickly drinks beer.`

5
Bibliography

[1] 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].

[2] Matthias Bastian. Google PaLM: Giant language AI can explain jokes, April 2022.

[3] 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.

[4] 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].

[5] 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.

[6] 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].

[7] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. A Categorical Semantics of Signal Flow Graphs. volume CONCUR 2014 - Concurrency Theory - 25th International Conference, September 2014.

[8] 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].

[9] Chapman, David. Nebulosity | Meaningness, December 2010.

[10] 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].

[11] Noam Chomsky. *New Horizons in the Study of Language and Mind*. Cambridge University Press, Cambridge, 2000.

[12] 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].

[13] Kenneth Church. A Pendulum Swung Too Far. *Linguistic Issues in Language Technology*, 6, October 2011.

[14] Bob Coecke. Compositionality as we see it, everywhere around us, October 2021. arXiv:2110.05327 [quant-ph].

[15] 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].

[16] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017.

[17] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning, March 2010. arXiv:1003.4394 [cs, math].

[18] 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.

[19] Luciano Floridi. *The Fourth Revolution: How the Infosphere is Reshaping Human Reality*. OUP Oxford, New York ; Oxford, June 2014.

[20] 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.

[21] 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].

[22] 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].

[23] Peter Gärdenfors. *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. January 2014.

[24] 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.

[25] Jules Hedges. String diagrams for game theory, March 2015. arXiv:1503.06072 [cs, math].

[26] 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].

[27] 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.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[29] Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal Inference by String Diagram Surgery, July 2019. arXiv:1811.08338 [cs, math].

[30] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991.

[31] Pentti Kanerva. Computing with High-Dimensional Vectors. *IEEE Design & Test*, 36(3):7–14, June 2019.

[32] Tabarak Khan. What are tokens and how to count them?, 2023.

[33] 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].

[34] Nikolaus Kriegeskorte and Katherine R. Storrs. Grid Cells for Conceptual Spaces? *Neuron*, 92(2):280–284, October 2016.

[35] 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.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[37] 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].

[38] 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.

[39] Robin Lorenz and Sean Tull. Causal models in string diagrams, April 2023. arXiv:2304.07638 [cs, math].

[40] 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.

[41] D. Marr. Artificial intelligence—A personal view. *Artificial Intelligence*, 9(1):37–48, August 1977.

[42] Marjorie McShane and Sergei Nirenburg. *Linguistics for the Age of AI*. March 2021.

[43] 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.

[44] Sharan Narang and Aakanksha Chowdhery. Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance, 2022.

[45] OpenAI. ChatGPT: Optimizing Language Models for Dialogue, November 2022.

[46] 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].

[47] 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.

[48] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–424, September 1980. Publisher: Cambridge University Press.

[49] Peter Selinger. A survey of graphical languages for monoidal categories. volume 813, pages 289–355. 2010. arXiv:0908.3347 [math].

[50] Paweł Sobociński. Graphical Linear Algebra, June 2015.

[51] Richard Sutton. The Bitter Lesson, 2019.

[52] Anders Søgaard. Grounding the Vector Space of an Octopus: Word Meaning from Raw Text. *Minds and Machines*, 33(1):33–54, March 2023.

[53] Taori, Rohan, Gulrajani, Ishaan, Zhang, Tianyi, Dubois, Yann, Li, Xuechen, Guestrin, Carlos, Liang, Percy, and Hashimoto, Tatsunori B. Stanford CRFM, 2023.

[54] teddy [@teddynpc]. I made ChatGPT take a full SAT test. Here's how it did: https://t.co/734sPFU3HY, December 2022.

[55] Alan D. Thompson. GPT-3.5 IQ testing using Raven's Progressive Matrices, 2022.

[56] 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.

[57] 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].

[58] 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].

[59] Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc, Champaign, IL, illustrated edition edition, August 2002.