VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

(Acknowledgements will go in a margin note here.)

0

Synopsis

0.1 What this thesis is about

THIS THESIS IS ABOUT DOING LINGUISTICS USING STRING DIAGRAMS.

Text diagrams are compositional blueprints that may be instantiated by classical or quantum computers. Since we know how grammar composes meanings in language, we can quotient out grammar using these formal diagrams. In terms of potential practical value, this process turns big black boxes into composites of small black boxes, which is a happy middle ground for humans and computers: smaller pieces for machines to learn representations for, and easy-to-reckon representations for humans. In terms of theoretical value for formal linguistics, the mathematics of string diagrams – applied category theory – allows us to unify different views of syntax and expand the reach of formal semantics, along the way providing the only meaningful conception of linguistics in a world where large language models exist. Enjoy a sideshow in the margins.

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

Natural language is *the* human superpower, 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. A theory of language is a theory of everything that can be theorised.

POINT OF INFORMATION: WHAT ARE STRING DIAGRAMS?

String diagrams are a heuristically natural yet mathematically formal syntax for representing complex, composite systems. I will define them formally and demonstrate their use in Section **??**. I say *mathematically* formal to emphasise that string diagrams are not merely heuristic tools backed by a handbook of standards decided by committee: they are unambiguous mathematical objects that you can bet your life on [12**?** , 15, 14, 22].

Just as crustaceans independently converge to crab-like shapes by what is called *carcinisation*, formal notation for formal theories of "real world" problem domains undergo "string diagrammatisation". Why should that be so? 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 express ourselves using composition.

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. In time, probably by doodling a helpful line during calculation to match indices, connected indices become wires, and string diagrams are born.

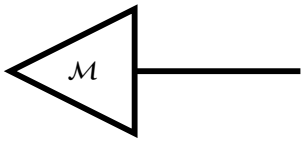


Figure 1: Let's say that the meaning of text is how it updates a model. So we start with some model of the way things are, modelled as data on a wire.

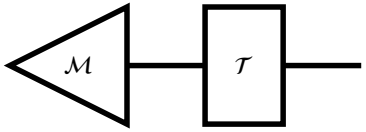


Figure 2: Text updates that model; like a gate updates the data on a wire.

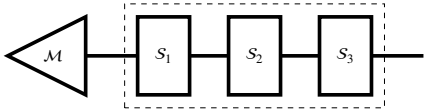


Figure 3: Text is made of sentences; like a circuit is made of gates and wires.

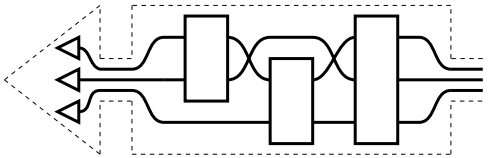


Figure 4: Let's say that ***The meaning of a sentence is how it updates the meanings of its parts.*** As a first approximation, let's say that the *parts* of a sentence are the nouns it contains or refers to. Noun data is carried by wires. Collections of nouns are related by gates, which play the roles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

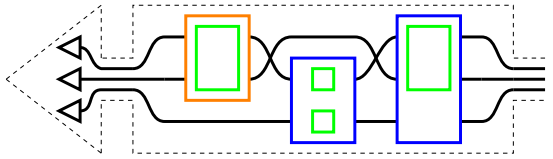


Figure 5: Gates can be related by higher order gates, which play the roles of adverbs, adpositions, and conjunctions; anything that modifies the data of first order gates like verbs.

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

Just as Camus treats suicide as the fundamental question, the first question we must address is why it is worth continuing, why this thesis is worth writing, why this topic is worth attention. Although this thesis is pure theory, I will start with the issue of practical value because I imagine practical people are impatient.

We only need to address the single, devastating, question above. Let me outline the terms and stakes. Large Language Models, at the time of writing, 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 [27]. This sounds unimpressive, but – in tandem with reinforcement learning in the case of chatGPT [19] – it is enough to tell and explain jokes [2], pass the SAT [24] and score within human ranges on IQ tests [25].

As a historical aside, there is an aspect of genuine scientific surprise that text-prediction can do this kind of magic. On the account of [16], 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 [6] 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. While there remain limitations in LLMs, such as tendency to hallucinate facts and (ironically, for a computer) bad arithmetic [10], it is evident to all 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 silicon 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 [9], 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 [23]. The tragedy goes like this: there 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 longer than even the idea of computers. 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 our understanding and theories 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 our current understanding. That is the bitter lesson: dumb methods that use a lot of data and compute outperform clever design and principled understanding.

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 in particular. I am unfortunately placed in that I feel enough affinity to

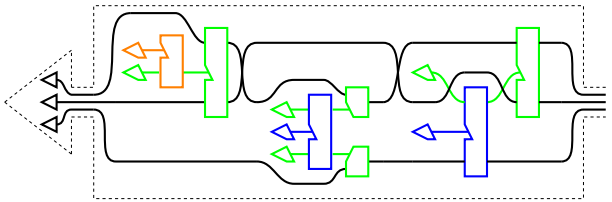


Figure 6: In practice, higher order gates may be implemented as gates that modify parameters of other gates. Parameters are depicted as additional inputs to gates.

placeholder

Figure 7: Grammar, and *function words* – words that operate on meanings – are absorbed by the geometry of the diagram.

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. First a response that answers with practical values in mind, and then a response that asserts and rests upon the distinct values of linguists.

REPLY: EXPRESSING GRAMMAR AS COMPOSITION OF PROCESSES MAY YIELD PRACTICAL BENEFITS. MOREOVER, WE WANT ECONOMY, GENERALITY, AND SAFETY FOR LANGUAGE MODELS, AND WE CAN POTENTIALLY DO THAT WITH NO 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'.

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 that Chomsky posits [4] 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 some 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 [17], 90 billion neurons [11], 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 – 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 [5, 18] – 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 [13] and 5 bits per character. The architecture has 540 billion neurons, and required 3.2 million kilowatt hours of energy for training [26]. 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*. If it is worth hunting a fraction of a percent of improvement on a benchmark, forget your hares, here is a stag.

So how do we hunt the stag? What do 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 fishes in water, and it is hard for fishes to characterise the nature of water. So the happy observations are difficult to produce and easily verified, and that means there just a few that we know of that are are unobjectionably worth taking into account. One, or *the* such observation is *compositionality*. Frege's initial conception of compositionality [] was borne of meditations on language, and states that a whole is the sum of its parts; the more that sounds like a contentless tautology, the cleverer Frege is for spotting it. 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. Another guise of compositionality is *systematicity* – the concepts differ slightly or not at all depending on where you are from academically. Systematicity [] refers to when a system can (generate/process) infinitely many (inputs/outputs/expressions) using finite (means/rules/pieces) in a "consistent" (or "systematic") manner; the more contentless and circular that sounds, the cleverer Fodor is for expressing it. 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. The last example, that of "finitely presented structures" in mathematics, is probably the closest formal incarnation of systematicity. The only way we know how to achieve systematicity in practice is composition. In the practical domain of computers, systematicity is synonymous with programmability and expressibility.

If we accept that compositionality is a necessary part of the solution to the problem of the stimulus, the issue with purely data-driven architectures is either that we know immediately that they cannot be compositional, or their innards are too large and their workings too opaque to tell with confidence. I hope the framework I present in this thesis can be a bridge, a way to split the cake fairly between the two halves of the problem: meanings for the machines, compositionality for the commons. Syntax is still difficult and quite 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.

OBJECTION: THE HUMAN MECHANISM CANNOT BE WORTH IMPLEMENTING IN SILICO BECAUSE THAT REQUIRES SUBJECT-MATTER EXPERTISE, AND THAT GOES AGAINST 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 [][deep, decision, qlearn, GAN, transformers] – 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. These structural changes are motivated by understandings (at varying degrees of formality) of the "geometry of the problem" []. The value proposition here is that with an appropriate mathematical lingua franca for structure, composition, and interaction (Category Theory), we can mindfully design rather than stumble upon the "meta-methods" Sutton calls for, allowing experts to encode *how* they 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.

Objection: GOFAI called and they want their symbolic-compositional approaches back. Can't you see that connectionist methods have already won?

Connectionism is winning, in the sense that the lion's share of AI research today is connectionist []. Moreover, hostility (or at least indifference) to symbolic approaches is a stance espoused by some of the leading lights of modern machine learning []. This stance is worth elaborating and steelmanning for pen-and-paper-people in the context of language. First, many linguistic phenomena are nebulous [] – 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 []. The latter term refers to a special kind of computational difficulty which 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 the lesson is that even for very simple systems, it is possible that no amount of causal-mechanistic understanding will simplify compututational simulation. These two points together weakly characterise the kinds of problem domains where machine learning shines. It is just a fact that LLM outputs today conform to any sensible 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 also a fact that they will only get better from here.

So, as far as practical language applications are concerned, for a theory to be a nonstarter, it needs to bring something to the table. 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 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, like deciding on a medical course of treatment [] or making decisions with financial consequences []. 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.

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[1] 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 [], to say nothing of the potential harm that may result from a poorly safeguarded model []. Add to this the ever-growing dirty laundry lists of AI models failing [] in inhuman ways, and the task of incorporating compositionality – a formal understanding of *how* models

[1] As a contextual aside, I recount the following from [], which argues that symbol-grounding is solvable from data alone, and in the process surveys the front of the symbol-grounding problem in explainability: 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 [21] phenomenon, so no amount of linguistic competence can be evidence that LLMs solve the symbol-grounding problem – e.g. knowing what words refer to and mean. However, the available evidence appears to suggest otherwise – large models converge on word embeddings for geographical place names that are isomorphic to their physical locations []. Since we know that brain activity patterns encode words in a manner that facilitates analogical reasoning in an abstract conceptual space [], extrapolating the ability of LLMs to encode analogical representations would in the limit suggest that LLMs encode meanings in a way isomorphic to how we do – at least for individual tokens.

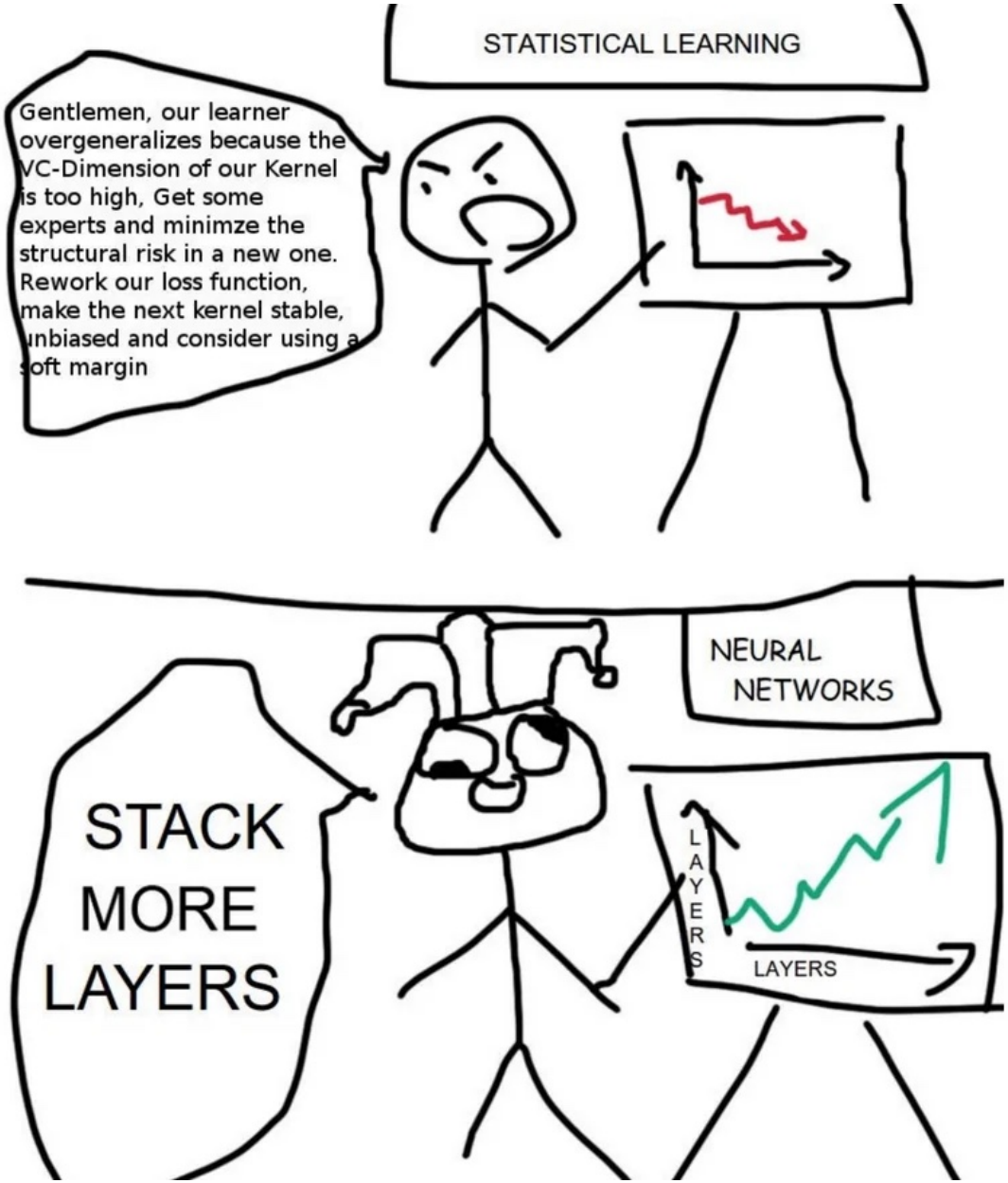


Figure 8: A caricature [7] that summarises the opposing epistemic stances of the symbolic/connectionist divide at a glance. For those unfamiliar, here is a recap. The symbolic side is synonymous with Good-old-fashioned-artificial-intelligence (GOFAI), research programme from the 70s to create general artificial intelligence. However, programming this explicitly turned out to be very hard because it was tantamount to systematising all of reasoning and knowledge []frameproblem, which is why GOFAI is sometimes described as knowledge-based. In the meantime connectionist methods – today synonymous with Deep Learning [] – leapfrogged GOFAI, for reasons explored in more detail in Section **??**. What distinguished connectionism was a reliance on data and compute rather than explicit programming, so it is sometimes described as knowledge-lean. A bullish sentiment arose among connectionists that cranking the handle to increase the size of the computer and the amount of training data would suffice to eventually obtain general artificial intelligence [1]. Debates surrounding this position fall generally under the umbrella of the epistemology of Data Science [20, 8]. In the case of LLMs specifically, modern debates of the bullish sentiment are developing, often rapidly. For example, in a thorough survey of LLM capabilities, [] warned against a fallacious conflation of linguistic and cognitive abilities, while observing several failure modes of GPT3 in cognitive domains. By the time the paper was published, those observations no longer held for GPT3's successor, ChatGPT [], which patched the failures with the introduction of reinforcement learning.

learn and reason – gains urgency.

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. The second kind is the inverse, where connectionist architectures are organised and reasoned with by symbolic-compositional means. Some examples of the first include implementing data structures as operations on high-dimensional vectors, taking advantage of the idiosyncrasies of linear algebra in very high dimension [], or work that explores how the structure of word-embeddings in latent space encode semantic relationships between tokens []. Some examples of the second include reasoning about the capability of graph neural networks by identifying their underlying compositional structure [], or architectures explicitly designed to instantiate symbolic-compositional structures using neural nets as constituent parts, such as GANs [] and gradient boosted decision trees []. The work in this thesis builds upon a research programme – DisCoCat, elaborated in Section **??** – which lies somewhere in the middle of the duality. It is, to the best of my knowledge, the only approach that explicitly incorporates mathematically rigourous compositional structures with data-driven learning methods from the ground up. Fortifying this bridge across the aisle requires a little give from both sides; I ask only that reader entertain some pretty string diagrams.

OBJECTION: AREN'T STRING DIAGRAMS JUST GRAPHS? WE UNDERSTAND GRAPHS WELL ENOUGH THAT IF THAT WERE THE SOLUTION WE WOULD HAVE HAD IT BY NOW.

Yes and no![2] 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 a drink 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 cards 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 a mistake to dismiss graphs as "just sets of vertices and edges". String diagrams are indeed "just" a special family of graphs, just as much as prime numbers are special integers and analytic functions are special functions.

In a broader context, representations matter for the sake of improved human-machine relations. These two representations are the same as far as a computer or a formal symbol-pusher is concerned, but they make world of difference to a human native of meatspace. We ought to swing the pendulum towards incorporating human-friendly representations in language models, so that we may audit those representations for explainability concerns. As it stands, there is something fundamentally inhu-

[2] 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, and the oral tradition and living history of mathematics in institutions of learning were decimated by the deaths of mathematicians at war. In a broader historical context [], Bourbaki was a reactionary response to the crisis in the foundations of mathematics at the beginning of the century, elicited by Russell's paradox. Accordingly, their aims were rationalist, totalitarian, and high-modernist [], 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 an evolved form of Euclid's that eschews intuition and example, a format pretending at timelessness that requires years of initiation to effectively read and write, and remains *de rigeur* for rigour today in dry mathematics textbooks. The deeper objection arises from the supposition that serious mathematics ought to look 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.

man and behavioural about treating the production of language as a string of words drawn from a probability distribution. In practice, this is what LLMs do. When *you* use language, do you feel like a diceroll? 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: we can guarantee 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. We sketch how to approach this in Section **??**.

0.3 ***Question:*** *How do string diagrams help us understand language better?*

Another way to deal with the devastating question of LLMs is to reject it, on the basis that 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 [], 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. The answer in short is that linguists have failed to provide an adequate understanding of language at the most basic level, and that string diagrams make it simpler to express an adequate account. Let's give the practical reader one more objection before we elaborate the reply.

OBJECTION: IF THE BETTER THEORY IS ONE THAT GIVES BETTER PREDICTIONS, AREN'T MANY THEORIES OF LANGUAGE KNOCKED OUT OF THE GAME BY AN LLM BEFORE THEY CAN EVEN GET STARTED?

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 [], 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.

While empirical capture is necessary for explanatory theories, it is insufficient. We may belay the order to vacate linguistics

departments when we consider the case study of models of 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" []. This was because Ptolemaic epicycles can overfit to approximate any observed trajectory of planets. It took until Einstein's relativity to explain the 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.

There are several criteria where the adequacy of LLMs is unclear or debatable. Fruitfulness is a sociological criterion for goodness of explanatory theories, in that they should generate new predictions and lead to further discoveries and research []. While they are certainly a potent catalyst for research in machine learning, it is unclear how they relate to the subject matter of linguistics. Whether they satisfy Popper's criterion of falsifiability is as of yet not determined, because it is not settled how to go about falsifying the linguistic predictions of LLMs, or even express what the content of a theory embodied by an LLM is. The closest examples to falsifiability that come to mind are tests of LLM fallibility for reasoning and compositional phenomena [], or their weakness to adversarial prompt-injections [], 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 [], and they do not unify or subsume our prior understandings []. The first two points are unobjectionable, so I will briefly elaborate on unification and subsumption of prior understandings, borrowing a framework from cognitive neuroscience. A common methodology for investigating cognitive systems is Marr's 3 levels [] (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 [] and of token-embeddings in latent space []. We can explain how transformer models work in terms of attention mechanisms and lookback [], 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 []. 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.

FIRST REPLY: STRING DIAGRAMS ARE A GOOD METALANGUAGE FOR FORMAL LINGUISTICS

Set-theoretical foundations of mathematics are not well suited for complex and interacting moving parts. The chief drawback is that if you want to specify a function, you have 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[3]. As you may already know, this representation-dependency is a nightmare when dealing with a complex system: you have to specify all the implementation details from start to finish, bottom-up. 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; 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 interacting 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 does not a theory make; we will inevitably wish to bring it all together. I show 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.

SECOND REPLY: APPLIED CATEGORY THEORY IS THE ONLY SENSIBLE MATHEMATICS FOR THE ONLY SENSIBLE CONCEPTIONS OF THEORIES OF LANGUAGE. I will proceed to argue for a new standard of what could possibly count as

[3] This is a foundational, innate problem 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 that are equivalent in spirit but not in syntax; a sign that the syntax is a hindrance. 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:

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

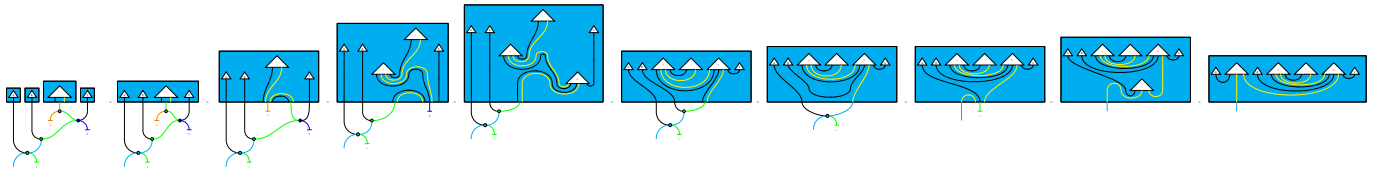
And here is Wiener's definition:

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

a theory of language in the age of LLMs, and then I will adhere to the standard I made up, and claim my work is important because it is the only thing that adheres to the standard I made up. I know it's unsportspersonlike, but it feels like the right thing to do in a doctoral thesis. Now it is my turn to ask a question:

0.4 ***Question:*** *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. Please write to me if you disagree because I haven't found anyone yet that does. 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 from a thought and a listener parses it to recover the thought. Here is a sequence of diagram equations – which we will revisit in Section **??** – 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.



I know that CFGs and pregroups are syntactic backwoods, which is why I will also show how the story works with generalisations of TAGs (circuit-adjoining grammars) as productive grammars and more powerful pregroup *diagrams* as parsing grammars in Sections **??**. Now I will briefly go on the offense.

Start(Offense): In my view, that picture is so far the only honest standard of what counts as a combined theory of syntax and semantics, if one accepts 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.

If that is unobjectionable, then to the best of my knowledge, that picture is *only* account of both syntax and semantics worth anything that you have ever seen. Mathematical elaboration and coverage of objections will happen in Section **??**. Now here is why that picture, for now, is the only game in town for an account of syntax and semantics in natural language in a post-LLM world.

THEORIES OF GRAMMAR BY THEMSELVES ARE INSUFFICIENT TO ACCOUNT FOR COMMUNICATION. 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.

THE INVESTIGATION OF COHERENCE OF THEORIES OF GRAMMAR MUST OBEY CERTAIN CONSTRAINTS. 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, so the exercise is pointless. Secondly, "strong equivalence" that seeks equivalence at a structural level between theories of syntax is sufficient but unnecessary. 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 IS NECESSARY TO SEEK A TARGETED FORM OF STRONG EQUIVALENCE BETWEEN A SPECIFIC PRODUCTIVE AND A SPECIFIC PARSING GRAMMAR. Specific grammars – and not formats of grammar, such as "all CFGs" – that model natural languages, even poorly, are the only linguistically relevant objects of study. If you purport to have a specific grammar that produces sentences in natural language, then to explain communication, you must supply a specific partnered parsing grammar such that the two are strongly equivalent. Only once that structural correspondence has been obtained, *that* underlying structure is an appropriate domain to begin a Montagovian account of semantics, broadly construed as a homomorphism between syntax and semantics. It is not enough to just talk of homomorphisms between a single theory of syntax and some semantics, unless you believe that language is adequately characterised as a solipsistic exercise where meaningful utterances are either drawn from the void or produced for nobody. The Montagovian programme may mathematically interesting and well-studied, but unless one additionally has agreement between grammars of producing and parsing speech, none of it can account for the obvious fact that communication happens. Obviously Montagovian semantics by itself is also insufficient to explain communication, but it plays a central role in the mathematics for an adequate account.

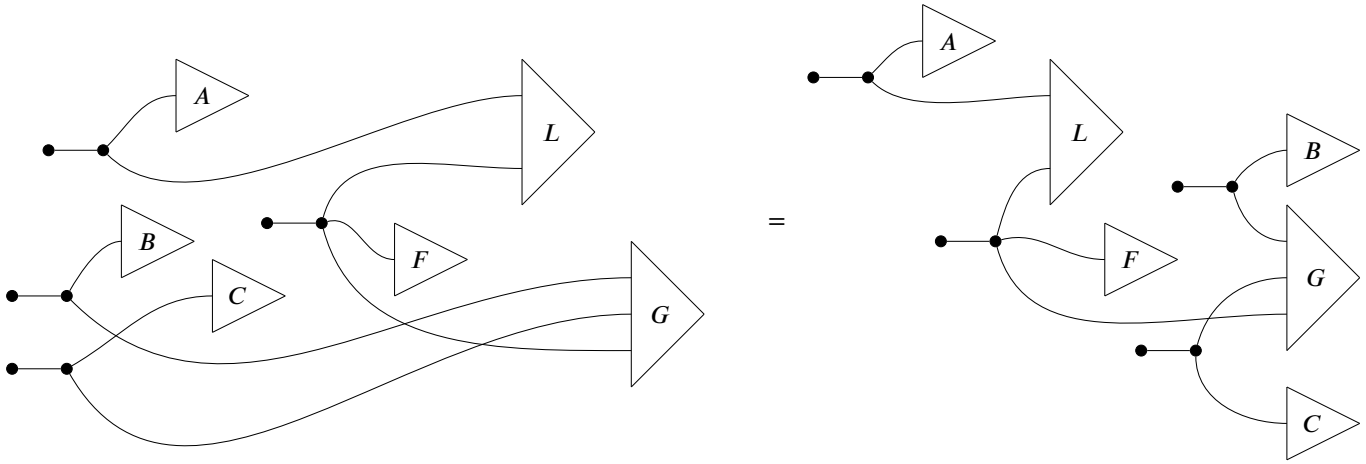
A POST-LLM THEORY OF LANGUAGE REQUIRES COHERENCE BETWEEN THEORIES OF SYNTAX – ONE FOR THE SPEAKER AND ONE FOR THE LISTENER – SUCH THAT THEIR UNDERLYING STRUCTURES ARE STRONGLY EQUIVALENT AND ARE AMENABLE TO DISTRIBUTIONAL SEMANTICS. Anything short of this fails to provide any understanding of language beyond what can be gleaned from an LLM. If you don't have two coherent theories of syntax for speaker and listener, you cannot account for communication. If you don't have a matching theory of semantics that can be learned in principle from data and represented as vectors-and-operations-upon-them, your theory will never be of practical relevance; **N.B.** go collect an award and dance on Minsky's grave if you figure out how to learn insert-FOL-variant-here representations of dictionary words from data. The stakes are, at worst, the entirety of linguistics as an enterprise of inquiry. At best, it's all made up and nothing really matters so I will subscribe to this standard anyway just because. To the best of my knowledge there isn't any, even partial account of language that satisfies the bare minumum. There are popular accounts of how grammar composes (albeit practically worthless) truth-theoretic semantics [Heim and Kratzer], and there are too many equivalence proofs between different grammatical formalisms, but I can't find anything that does the minimum all at once, and if it does exist, it's probably unreadably expressed in outdated mathematics not fit for purpose. So in this thesis I will supply a post-LLM theory of language, and I will endeavour to do it as diagrammatically as possible. I only care to get the absolute basics right by my own standard, and I will accordingly treat everything else about language like pragmatics and social-whatever as an afterthought, *as it should be*: consider air resistance only after you know how motion works in a vacuum. If you're a linguist and you don't like what I'm saying and you're not already tenured, good luck. **End(Offense)**

0.4.1 Grammars of speakers and listeners

There is a distinction between *grammars of the speaker* – which produce sentences – and *grammars of the listener* – which deduce sentences. Viewed as mathematical processes, the two kinds of grammars go in opposite directions; speaking grammars (e.g. string-rewrite systems) start with some grammatical structure, and require informational input (e.g. which rule comes next) to produce lists of words – sentences. Conversely, listening grammars (e.g. typelogical grammars) start with a sentence, and require informational input (e.g. grammatical typing and

which proof rule to try next) to deduce or parse a grammatical structure. Since we can understand each other, these two types of grammar must enjoy a systematic correspondence, and if one believes that semantics is compositional according to syntax, then the correspondence must further explain how both speaking and listening grammars manipulate the same underlying semantic expression, whatever that may be.

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. Just as a running example that does not affect the point, let's say we can gloss the thought in first order logic as $\exists a \exists b \exists c \exists f \ : \ A(a) \wedge B(b) \wedge C(c) \wedge F(f) \wedge L(a, f) \wedge G(b, c, f)$. In diagrammatic first order logic [], this is equivalently presented as the following diagrams (and any other diagram that agrees up to connectivity)

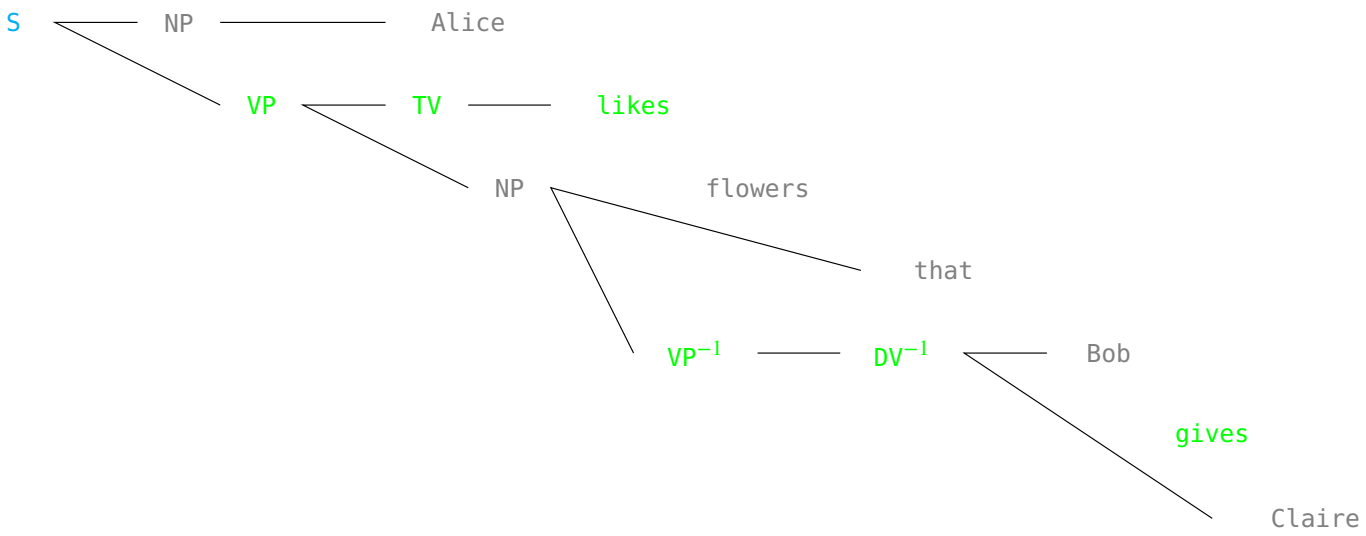


Preube and Fondo cooperate to achieve the miracle; Preube encodes his thoughts – a structure that isn't a one-dimensional string of symbols – into a one-dimensional string of symbols. And then Fondo does the reverse, turning a one-dimensional string of symbols into a thought-structure like that of Preube's. What I mean by "like that of Preube's" is that both Preube and Fondo agree on the structure of entities and relations up to the words for those entities and relations. For example, Preube could ask Fondo comprehension questions such as `WHO GAVE WHAT? TO WHO?`, and if Fondo can always correctly answer – e.g. `BOB GAVE FLOWERS. TO CLAIRE.` – then both Preube and Fondo agree on the relational structure of the communicated thought to the extent permitted by language. 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.

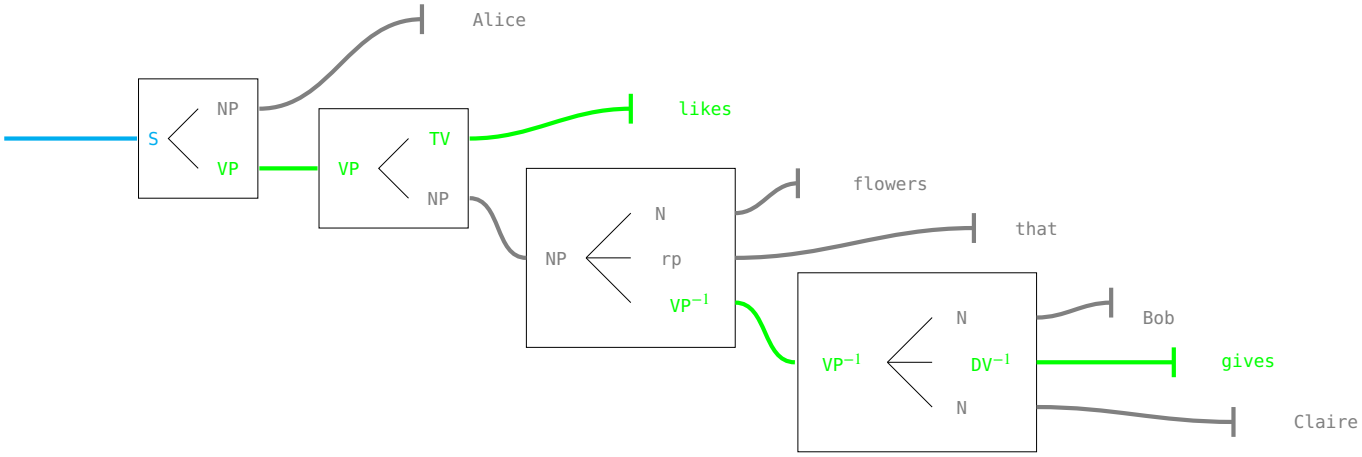
We assume Preube and Fondo speak the same language, so both know how words in their language correspond to putative building blocks of thoughts, and how the order of words in sentences and special grammatical words affect the (de-/re-)construction procedures. Now we have to explain how it is that the 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, and these topological manipulations make it easy to convert between different shapes of language.

Example 0.4.1 (`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. The rule of the game is that Preube and Fondo can agree on a string-diagrammatic encoding strategy before having to communicate with each other. Here is one such strategy: Preube might generate the

example sentence like so:

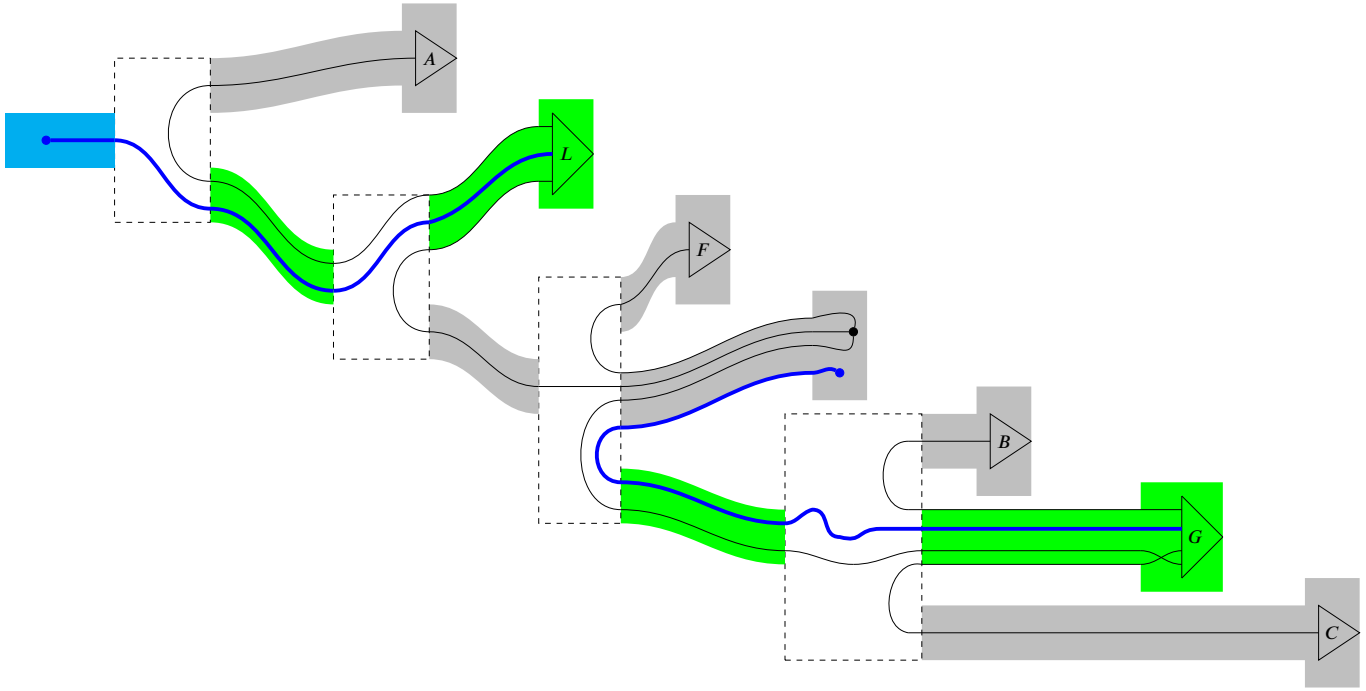


Mathematically, it makes no difference if we take the Poincaré dual of the tree, so that zero-dimensional nodes become one-dimensional wires, and branchings become zero-dimensional points linking wires – but we can just as well depict those points as boxes to label them more clearly.

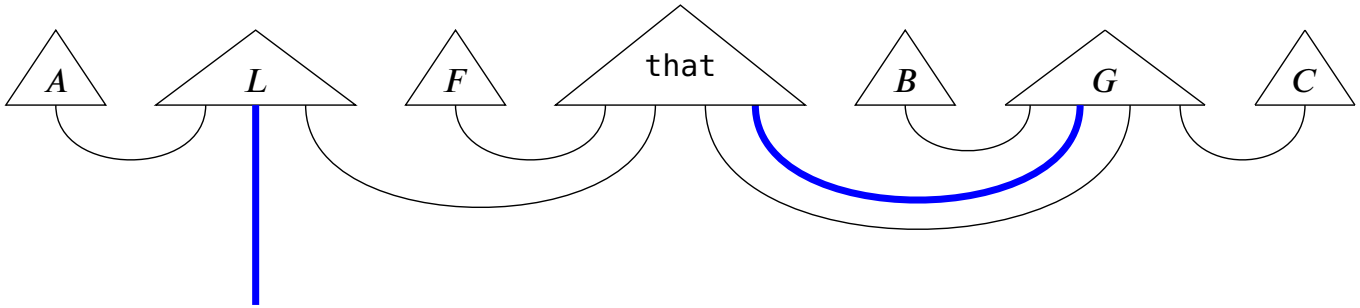


Now that Preube can express their grammatical structure string-diagrammatically, they can try to deform their first-order-logic diagram – representing what they mean to communicate – subject to the constraint that every one of their branchings (the structure of the CFG) is something recoverable by Fondo using just pregroup reductions. To do so, Preube introduces a formal blue wire to mimic Fondo's sentence-type, and stuffs some complexity inside the labels in the form of internal wirings: a multiwire configuration for `that`, and a twist for `gives`. Those

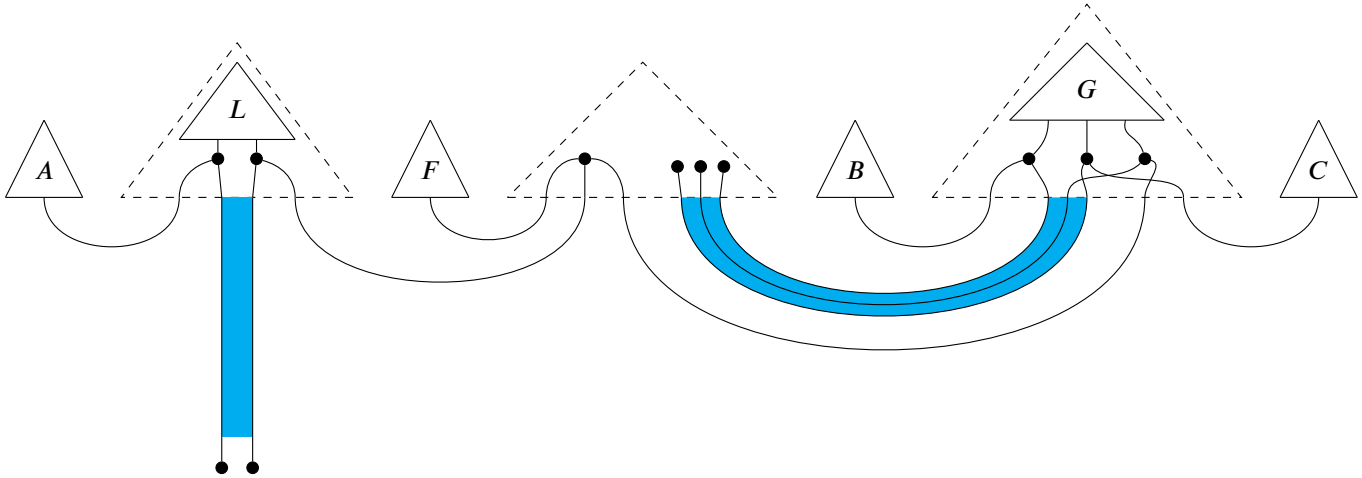
internal wirings are the content of Preube and Fondo's shared strategy.



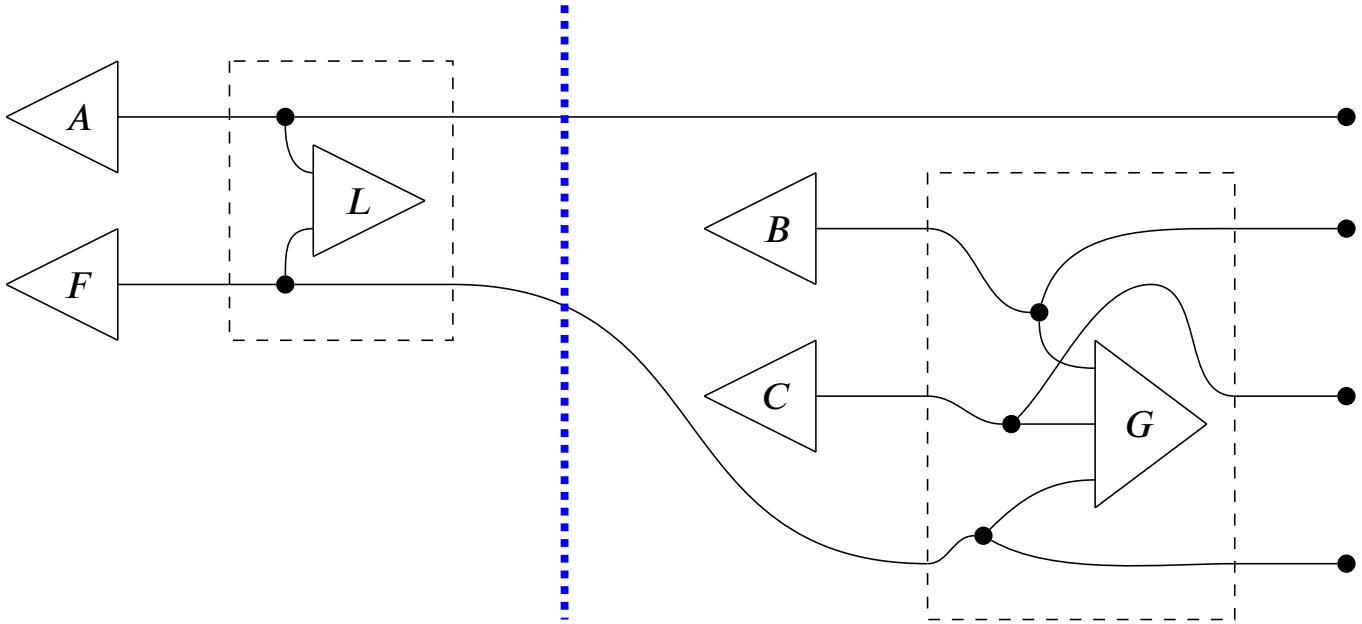
So, when Fondo receives the sentence, Fondo's pregroup derivation yields a pregroup diagram that is connectively equivalent to what Preube stuffed inside the context-free grammar structure. So now the two have strong equivalence between their grammars in the sense that every one of Preube's branches is resolved by one of Fondo's reductions.

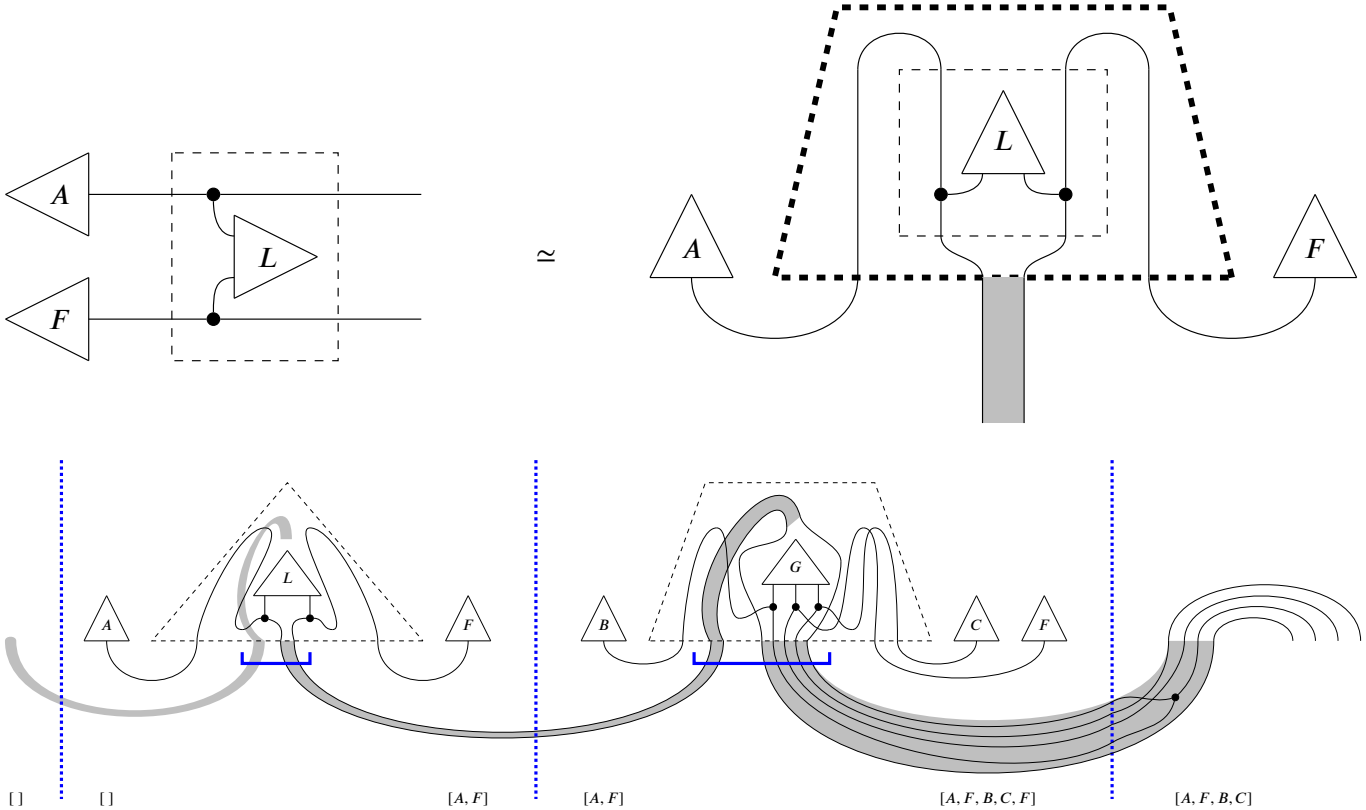


Now to fully recover Preube's intended FOL-diagram, Fondo refers to the internal wirings that form their shared strategy, and fills those in.



Example 0.4.2 (`Bob gives Claire flowers.  Alice likes flowers.`)**.**





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. I will now outline the constraints imposed by this interaction, illustrating them simply using string-rewrite grammars for the speaker and pregroup grammars for the listener.

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, those decisions represent information that must be supplied to Preube in the process of speaking a thought. For this reason, we consider context-free-grammars (and more generally, other string-rewrite systems) to be *grammars of the speaker*. 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 grammars of the speaker 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.

Example 0.4.3 (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:

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

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

$$\dfrac{\dfrac{\dfrac{\texttt{the} : n \cdot n^{-1} \qquad \texttt{old} : n}{\texttt{the\_old} : n} \qquad \texttt{man} :^{-1} n \cdot s \cdot n^{-1}}{\texttt{the\_old\_man} : s \cdot n^{-1}} \qquad \dfrac{\texttt{the} : n \cdot n^{-1} \qquad \texttt{boat} : n}{\texttt{the\_old} : n}}{\texttt{the\_old\_man\_the\_boat\_} : s}$$

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

Example 0.4.4 (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.

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

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

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

$$[\![\texttt{everyone}]\!] = \lambda(\lambda x.V(x)).\forall x : V(x) \tag{1}$$

$$[\![\texttt{loves}]\!] = \lambda x \lambda y.\texttt{loves}(x, y) \tag{2}$$

$$[\![\texttt{someone}]\!] = \lambda(\lambda y.V(y)).\exists y : V(y) \tag{3}$$

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

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

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

As is convention for parsing, let's grant that there's a daemon in Fondo's head that makes all these lexical disambiguation choices for them, automatically settling on which sense of `the old man` or `somebody` is appropriate. As mathematicians looking for a toy model to get started, we are looking for the simplest kind of choice that Fondo can be trusted to make with only grammatical information available to them.

Example 0.4.5. (grammar pieces – as simple as it gets) `Bob runs. Bob quickly runs. Bob drinks beer. Bob quickly drinks beer.`

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 [] and n-categories as formalised by [], 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 [] 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 [] 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 about formal semantics beyond Montague and logic that text circuits in **ContRel** allow us to explore. I model 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. 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. I hope you enjoy the read, or if nothing else, I hope you like my diagrams!

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 [], 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 [] 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 **??**.

1
Bibliography

[1] Chris Anderson. The End of Theory: The Data Deluge Makes the Scientific Method Obsolete. *Wired*, 2008. Section: tags.

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

[5] 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].

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

[7] (deleted user). StAcK MoRe LaYeRs, April 2018.

[8] Jules Desai, David Watson, Vincent Wang, Mariarosaria Taddeo, and Luciano Floridi. The epistemological foundations of data science: a critical analysis. *SSRN Electronic Journal*, 2022.

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

[10] 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].

[11] 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.

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

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

[20] Wolfgang Pietsch. *On the Epistemology of Data Science: Conceptual Tools for a New Inductivism*, volume 148 of *Philosophical Studies Series*. Springer International Publishing, Cham, 2022.

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

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

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

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

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

[26] 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.

[27] 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].

2
Background

2.1 A Partial History of String Diagrams

OBSERVATION 1: WE HAVE A PRIMITIVE URGE TO DEPICT.

cavedrawingstolanguage

OBSERVATION 2: NATURAL LANGUAGE PRESENTS ITSELF AS A UNIVERSAL INTERFACE FOR THOUGHT.

placeholder

THE PICTORIAL CAPTURE OF LANGUAGE AND THOUGHT IS A CURSED DREAM. We suffer a seductive alchemical urge to capture the essence, laws and structure of things in words and pictures – containment in symbol and form. On occasion, this urge is focused on itself as an ouroboros; a doomed endeavour to use the faculties of abstraction, representation, and structure to consume itself.

peirce

magnusson

maybejung

IT IS OUR TURN TO SUFFER THIS SICKNESS. The difference this time around is in approach: we use *string diagrams*. If depiction is the domain of geometry – the mathematics of form in space, and if language is the domain of algebra – the mathematics of symbolic manipulation, then string diagrams are our bridge. We recall a quote by the late Michael Atiyah:

Algebra is the offer made by the devil to the mathematician. The devil says: "I will give you this powerful machine, it will answer any question you like. All you need to do is give me your soul: give up geometry and you will have this marvellous machine."

String diagrams are a loophole in the devil's contract. Descartes' led an incursion into geometry with algebra, and string diagrams are the belated counteroffensive, or peaceful synthesis.

STRING DIAGRAMS ARE FORMAL REPRESENTATION AND REASONING SYSTEMS FOR MONOIDAL CATEGORIES. "Formal" distributes over "representation" and "reasoning", which arose chronologically in that order.

2.1.1 *Formal visual representation*

Pre-formal diagrams are about visual expression. By the restrictions of writing technologies, the kind of visual representations we are interested in are two-dimensional. So visual means geometric in the plane. The ancient ancestors of modern string diagrams are systems for visual representation of formal symbolic constructions. For these ancestral diagrams, reasoning, if there is any, takes place elsewhere in the symbolic realm, not within and between diagrams.

$$Euclid : Geometry$$

$$Venn\&Carroll : Syllogisms$$

$$Petri : Chemistry$$

2.1.2 *Convergent Evolution*

String diagrams arise naturally when keeping track of many pairwise connections between entities becomes unwieldy for one-dimensional syntax. The usual evolutionary path is to first adopt Einstein notation with indexed superscripts and subscripts for bookkeeping, followed by the depiction of connected indices as wires in the plane.

In Physics

In Logic

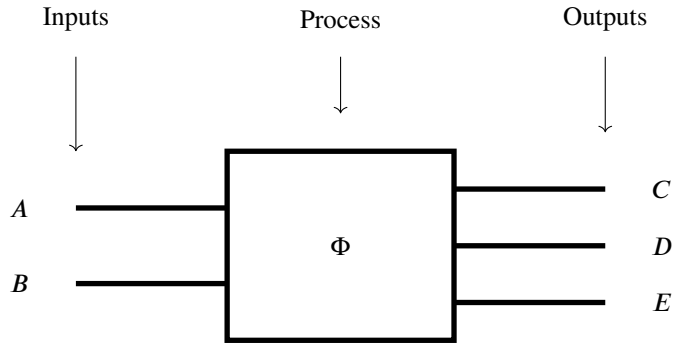
In Computer Science

2.1.3 *Formal visual reasoning*

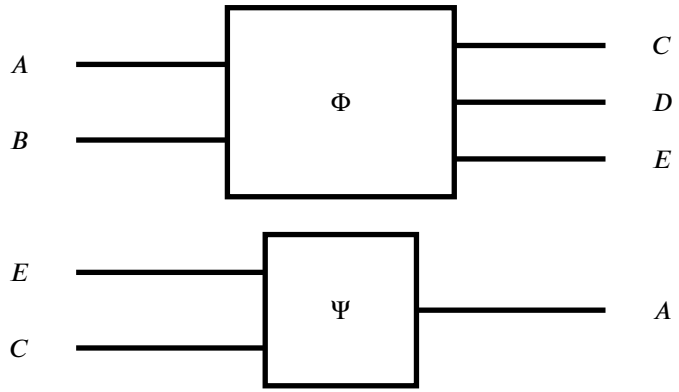
2.2 *Process Theories*

This section seeks to give an introduction to process theories in an intuitively grounded manner. We aim here to build the process-theoretic mathematics towards 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 in two dimensional Euclidean space equipped with notions of metric and distance. This section provides 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. Moreover, this section 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 aside on how process theories relate to mathematical foundations and computer science.

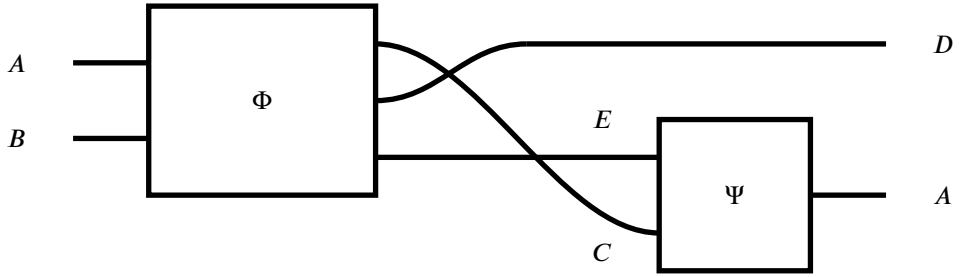
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. We read processes from left to right.



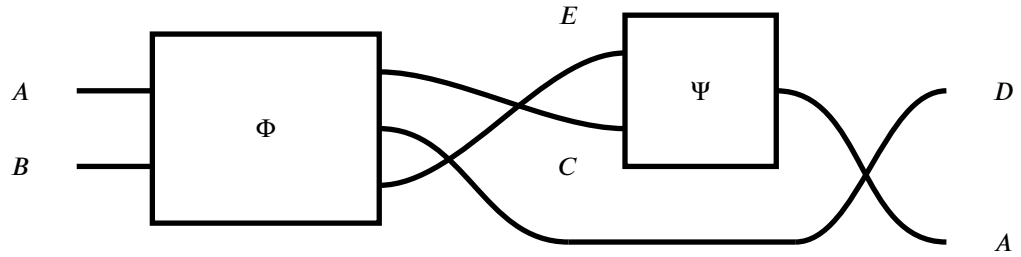
Processes may compose in parallel, which we depict as vertically stacking boxes.



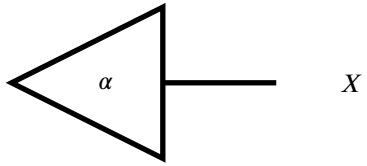
Processes may compose sequentially, which we depict as connecting wires of the same type from left to right.



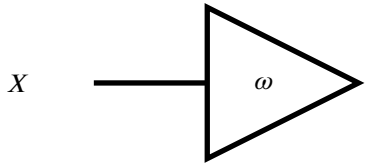
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



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

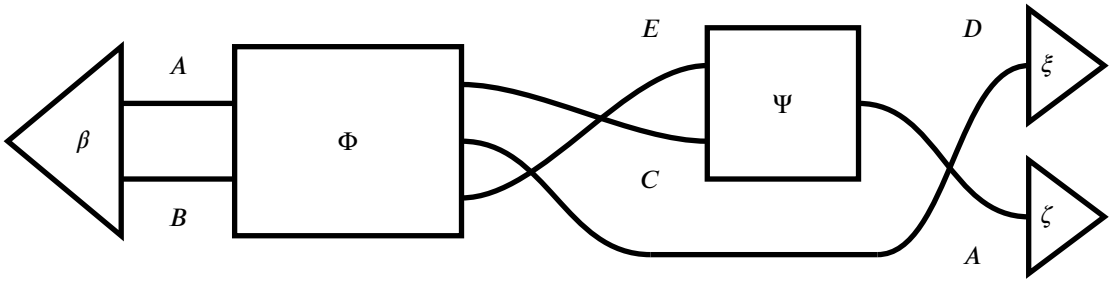


Some processes have no outputs; we call these *tests*.



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[1]:

- 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 2.2.1 (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 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 $\mathbb{R}$.[2]

Example 2.2.2 (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[3] singleton set $\{\star\}$. States of a set A correspond to elements $a \in A$[4]. Every system A has only one test $a \mapsto \star$[5]. There is only one number.

[1] Formally, process theories are symmetric monoidal categories []; see section **??**.

[2] Usually the monoidal product is written with the symbol $\otimes$, which denotes hadamard product for linear maps. The process theory we have just described takes the direct sum $\oplus$ to be the monoidal product. To avoid confusion we will use the linear algebraic notation when linear algebra is concerned.

[3] There are many singletons, but this presents no problem for the later formal definition because they are all equivalent up to unique isomorphism.

[4] We forgo the usual categorical definition of points from the terminal object in favour of generalised points from the monoidal perspective.

[5] This is since the singleton is terminal in **Set**.

Example 2.2.3 (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 $\Phi_* : B \nrightarrow A$.[6] 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.

2.2.1 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 2.2.4 (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 2.2.5 (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 funtion is defined:

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

Example 2.2.6 (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:

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

[6] 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.

We may verify that, no matter the concrete interpretation of the diagram in terms of linear maps, functions or relations copy and delete satisfy the equations in the margin:

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 in the margin, when translated into prose, provide an answer.

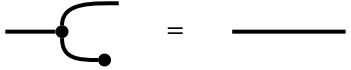
***Coassociativity**:* says there is no difference between copying copies.

***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.

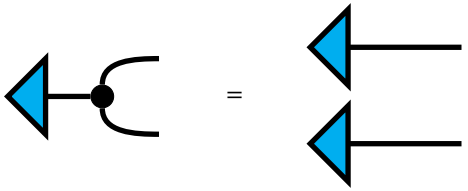
Insofar as we think this is an acceptable characterisation of copying, rather than specify concretely what a copy and delete does for each system X we encounter, we can instead posit that so long as we have processes $\Delta_X : X \otimes X \to X$ and $\epsilon_X : X \to I$ that obey all the equational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

Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 2.2.7 and Remark 2.2.8, 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 2.1, 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 2.1.[7]



counitality (2.1)

Example 2.2.7 (Not all states are copyable)**.** Call a state *copyable* when it satisfies the following diagrammatic equation:

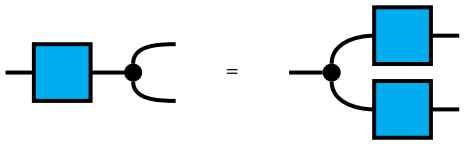


In the process theory of sets and functions, all states are copyable. Not all states are copyable in the process theories of sets and relations and of linear maps. For example, consider the two element set $\mathbb{B} := \{0, 1\}$, and let $\top : \{\star\} \nrightarrow \mathbb{B} := \{(\star, 0), (\star,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

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

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

Remark 2.2.8. 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*.

2.2.2 What is an update?

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

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

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

There are all kinds of reasons one might wish to update the value of a field: Jono might legally change their name, a year might pass and Jono's age must be incremented, Jono might be promoted or demoted or get a raise and so on. It was the concern of database theorists to formalise and axiomatise the notion of updating the value of a field – *independently of the specific programming language implementation of a database* – so that they had reasoning tools to ensure program correctness []. The problem is reducible to axiomatising a *rewrite*: we can think of updating a value as first calculating the new value, then *putting*

[7] Quantum physicists *do* do this; see Dodo: []

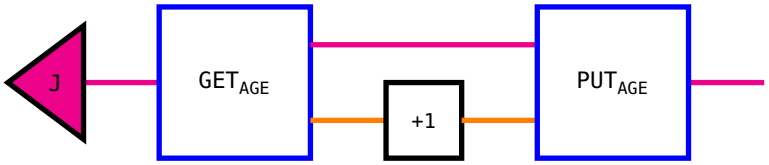
the new value in place of the old. Since often the new value depends in some way on the old value, we also need a procedure to *get* the current value.

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.

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.



2.2.3 *Spatial predicates*

The following simple inference is what we will try to capture process-theoretically:

- `Oxford is north of London`
- `Vincent is in Oxford`
- `Rocco is in London`

How might it follow that `Rocco is south of Vincent`?

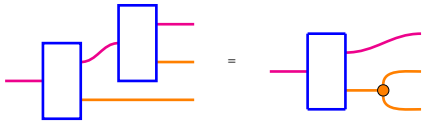
One way we might approach such a problem computationally is to assign a global coordinate system, for instance interpreting 'north' and 'south' and 'is in' using longitude and latitude. Another coordinate system we might use is a locally flat map of England. The fact that either coordinate system would work is a sign that there is a degree of implementation-independence.

This coordinate/implementation-independence is true of most spatial language: we specify locations only by relative positions to other landmarks or things in space, rather than by means of a coordinate system. This is necessarily so for the communication of spatial information between agents who may have very different reference frames.

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.



So the process-theoretic modelling aim is to specify how relations between entities can be *updated* and *classified* without requiring individual spatial entities to intrinsically possess meaningful or determinate spatial location.

So far we have established how to update properties of individual entities. We can build on what we have so far by observing that a relation between two entities can be considered a property of the pair.

placeholder

Spatial relations obey certain compositional constraints, such as transitivity in the case of 'north of':

placeholder

Or the equivalence between 'north of' and 'south of' up to swapping the order of arguments:

There are other general constraints on spatial relations, such as order-independence: the order in which spatial relations are updated does not (at least for perfect reasoners) affect the resultant presentation. This is depicted diagrammatically as commuting gates:

placeholder

2.2.4 Processes, Sets, Computers

`Objection:` BUT WHAT ARE THE *things* THAT THE PROCESSES OPERATE ON?

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

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

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

pointer to that element $\{\star\} \to X$. So the states we have been drawing, when interpreted in the category of sets and function, are precisely elements of the sets that label their output wires.

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

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

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

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

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

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

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

2.3 Defining String Diagrams

2.3.1 Symmetric Monoidal Categories

Definition 2.3.1 (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 $f : a \to b$, $g : b \to c$, $h : c \to d$, $(f; g); h = f; (g; h)$

Definition 2.3.2 (Categorical Product)**.**

Example 2.3.3 (Cartesian product of sets)**.**

Definition 2.3.4 (Functor)**.**

Definition 2.3.5 (Natural Transformation)**.**

Definition 2.3.6 ((1−)**Cat**)**.**

Example 2.3.7 (Monoids in **Set**)**.**

Definition 2.3.8 (Monoidal Category)**.**

Definition 2.3.9 (Symmetric Monoidal Category)**.**

Theorem 2.3.10 (Strictification)**.**

Theorem 2.3.11 (Graphical?)**.**

2.3.2 PROPs

PROP stands for "**PRO**duct and **P**ermutations category", but it may as well be an acryonym for "**P**icture **R**ules **Of P**rocesses".

2.3.3 1-object 4-categories

2.4 A brief diagrammatic introduction to Neural Nets

For those unfamiliar, it is worth understanding the fundamentals of how neural nets work, which illuminates how they are able to transmute large amounts of input data into strong and "theory-free" [] performance at tasks. The ability to convert data (in which we are drowning), along with the universal approximation theorem, explain why data-driven learning methods have recently outperformed handcrafted rules-based approaches to artificial intelligence. I will focus only on the basic deep neural net – so the reader is warned that the state of the art in terms of learning architectures is far more sophisticated, such as transformers [] used in large language models. There are only two important takeaways from this section. First is that the reader must understand why it is that data-driven learning methods outperform human axiomatising when dealing with complex problem domains, and are therefore (with the usual caveats of explainability) preferable. Second is an understanding of the statement of the universal approximation theorem, which will reappear in Section 5, where it will be of service in an attempt to introduce compositionality via interacting ensembles of neural nets.

NEURAL NETS ARISE FROM A TOY MODEL OF BIOLOGICAL NEURONS. At a glance, biological neurons have many receptors and one output, and the neuron fires a signal out if its combined inputs exceed an activation threshold. As a simplification, McCulloch-Pitts neurons are a sum of n inputs passed through an activation function $\sigma : \mathbf{R}^n \to \mathbf{R}$ that is permitted to be nonlinear, but traditionally monotone increasing and sigmoidal, which bounds the range of the function $\exists a_{\mathbb{R}} b_{\mathbb{R}} \forall x_{\mathbb{R}} : a \leq \sigma(x) \leq b$, and asks that σ approaches the lower and upper bounds in the limit as x goes to $-\infty$ and ∞ respectively. Using the diagrammatic calculus for linear algebra [] equipped with a nonlinear activation function – all of which is interpretable in **TopRel**, we can immediately grasp a visual resemblance between the designs of nature and man:

placeholder

THE FIRST USE OF NEURAL NETS WAS IN APPLICATION TO THE PROBLEM OF MACHINE VISION. These first, single-layer neural nets were called *perceptrons*. Mimicking the neuronal organisation of the visual cortex, it was a sensible idea to stack these layers on top of one another [] – these layers are the original reason for the word "deep" in "deep learning", but words change in meaning over time.

placeholder

THE MODERN UBIQUITY OF NEURAL NETS IS DUE TO SEVERAL FACTORS. First is Hinton's backpropagation algorithm [] (which may be obsolete when you are reading this by Hinton's forward-forward second salvo [].) Observe that even after one has decided on the shape of the neural net in terms of neuronal connectivity, there are still many degrees of freedom in the parameters of the activation functions, in particular their horizontal shift (bias) and vertical stretching (weights). Borrowing diagrammatic notation for parameters as orthogonal wires from [], we can depict the degrees of freedom for a single neuron like this:

placeholder

THERE IS A MASSIVE SPACE OF PARAMETERS TO SET FOR EVEN A MODERATELY SIZED NEURAL NET. So how do we set the parameters in such a way that the neural net computes something useful? Backpropagation solves this problem by leveraging the shape of a neural net. There are many easily searchable resources that cover backpropagation for the interested reader, including category-theoretic ones []. The simple explanation goes like this. Let's just focus on the weight parameter of each neuron. By analogy each neuron is a shitty person, and their weight is how strongly they hold a binary opinion. A neural net by analogy is a shitty rigid hierachical society with voters in the back and decision makers in the front. As a simple example, Alice and Bob each make a recommendation to Claire based on what they receive as input.

placeholder

Suppose that Claire's decision is wrong. She revises her own opinion then meets with her confidantes. Alice's recommendation was faulty, so Claire blames her; as a narcissistic defense, the viciousness of the blame is proportional to how wrong Claire was. Alice revises her own opinion proportional how mean Claire is being, and then Alice goes to seek out her confidantes to perpetuate a vicious cycle of psychological violence. Bob on the other hand was right, Claire tells him this with sheepishness proportional to her error, and he starts gloating "I told you so!" with glee proportional to how much cleverer he feels than Claire. So Bob becomes slightly more entrenched in his opinion, and then he goes to seek out his confidantes to either congratulate or belittle them, again proportional to how right he was. When all of the blame and backpatting has backpropagated throughout society, all the shitty people have adjusted their opinions, and their shitty society will be less prone to making the same mistake again. In human terms, this process is repeated for all of recorded history or longer, and then you have a neural net that can recognise handwritten digits.

placeholder

All this process needs to get started is a lot of labelled pairs of data, input along with the desired output for that input. The formal terminology for the scenario above that converts data into performance is "training", which is a computationally intensive process when lots of data is involved for big neural nets. So the second factor of the ubiquity of neural nets is Moore's law and analogues, which have overseen exponential growth in computational power and digital data storage capacity. Neural nets convert data and compute power as fuel into practical applications, and we live in an era of increasingly plentiful data and compute. Hence, the bitter lesson []; clever theories are no match for stupid methods with lots of data and a big computer. But why the hell should any of this work in the first place? Surely there are limits to what neural nets can do. Now the third factor; Moore's law and the bitter lesson might be cheated, but the third factor is a law backed by mathematics.

Theorem 2.4.1 (Universal Approximation Theorem)**.**

ANY PROBLEM THAT CAN BE ENCODED AS A CONTINUOUS TRANSFORMATION OF LISTS OF REAL NUMBERS INTO

OTHER LISTS OF REAL NUMBERS IS POTENTIAL PREY FOR A BIG ENOUGH NEURAL NET. A little creative thought will show that many practical problems can be treated in this way. The litigious can easily spot problems in neural nets outside of this law. For example, to the best of our knowledge there is no known lower bound for how much data is required – as a function of desired accuracy within a desired confidence – for a neural net to learn its target, so for all we know, any big neural net could suddenly fail on an easy input instance for no good reason. The universal approximation theorem is a double-edged sword, and the side that cuts the holder is that for complex problems, the input data cannot span the whole problem domain, so there will be many neural nets that agree perfectly on the training data but will perform differently out-of-distribution – this common phenomenon is called "overfitting". Moreover, a major component of explainability from the human perspective is having mental representations and rules for their manipulation, whereas it is in general very difficult to determine how or even whether the neural net has internal representations and rules for its inputs. This introduces an ethical criterion that most data-driven algorithms fail: it is not a big deal to be recommended a show we do not like or have a machine play a game more innovatively than a human could, but if the stakes are higher, such as operating a vehicle or on a patient, we would prefer safety guarantees predicated upon internal representations and rules that we can understand and negotiate in human terms. In Section 5 we will try to blunt the painful edges by using the universal approximation theorem and overfitting to our advantage in search of compositional, explainable internal representations.

(Typed) Lambda-Calculus	Intuitionistic Logic	Cartesian Closed Categories
Types	Propositions	Categories
Curry	Howard	Lambek

Computation	Syntax	Semantics
(Typed) Lambda-calculus	Combinatory Categorial Grammar	Cartesian Closed Categories
Montague	Ajdukiewicz	Lambek

2.5 A brief history of formal linguistics from the categorial perspective

SUMMARY: Logical Type Theory in mathematics had a twin sister Categorial Grammar in linguistics, born in the 30s to Ajdukiewicz, raised into the 50s by Bar-Hillel and Lambek. Montague brought formal semantics to the picture via the Lambda-Calculus in the 70s, around the time Category Theory was getting started. The Curry-Howard correspondence between types and logic became Curry-Howard-Lambek to include categories, thus relating the typed lambda-calculus, intuitionistic logic, and cartesian closed categories. Lambek and Moortgat evolved categorial grammar into typelogical grammar; the use of different proof systems as models of grammar, which in turn suggested semantic categories beyond the cartesian closed setting. Alternative semantics in the form of quantum computers were realised by Coecke and Lambek, who together added string diagrams to the correspondence via a *literally* observed correspondence between quantum bell states and reductions in Pregroup Grammar. Sazradeh and Clark enter the collaboration, bringing in Firth's distributional semantics – which had by the time become computationally practical as the basis of neural methods for word encoding – to create DisCoCat; a Distributional, Compositional and Categorial framework for language. Mirroring the developmental circumstances of Discourse Representation Theory (itself independently conceived by Kamp and Heim), Coecke suggested promoting DisCoCat as framework for sentences towards a circuit-shaped framework for text – DisCoCirc. But there remained unanswered questions and ugly spots, and some poor sap had to work out the formal details and clean things up. That poor sap is me.

2.5.1 Curry-Howard-Lambek

This correspondence means that you can use the lambda-calculus on any family of data organised as a cartesian closed category; this could be strings, or sets and functions, topological shapes with holes, neural nets and finite vectors. So one small benefit of the category-theoretic viewpoint is a formal underpinning for these mild extensions.

Describing the bigger benefit requires a bigger picture.

and Lambek and Coecke fully integrated the picture with syntax and categories. So the linguist's trinity may look something like this:

Or this:

So here is the story today as far as a linguist may be concerned. We know that Curry-Howard-Lambek- correspondence generalises: if you poke Howard for a grammar that is expressively distinct from a CCG, the type-theory changes, so Lambek gives a different family of semantic categories with different internal logics, and Curry gives you a syntactic composition gadget that

Computation	Syntax	Semantics
(Typed) Lambda-calculus	Pregroup Grammar	Rigid Autonomous Monoidal Categories
Montague	Ajdukiewicz	Lambek

differs from the lambda-calculus.

2.5.2 What did Montague consider grammar to be?

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

MONTAGUE SEMANTICS/GRAMMAR AS MONTAGUE ENVISIONED IT IS LARGELY CONTAINED IN TWO PAPERS – *Universal Grammar* [8], and *The Proper Treatment of Quantifiers in English* [9] – both written shortly before his murder in 1971. The methods employed were not *mathematically* novel – the lambda calculus had been around since [DATE], and Tarski and Carnap had been developing intensional higher-order logics since [] – but for linguists who, by-and-large, only knew first order predicate logic, these methods were a tour-de-force that solved longstanding problems in formal semantics. Thus, Montague semantics has largely been in the care of linguists rather than mathematicians. This meant sparse opportunity for the ideas to 'update' according to mainstream developments in mathematics.

[8]
[9]

THERE IS A NATURAL DIVISION OF MONTAGUE'S APPROACH INTO TWO STRUCTURAL COMPONENTS. First, the notion of a structure-preserving map from syntax to semantics. Second, the use of a powerful and expressive logic for semantics. We acknowledge the importance of the latter for formal semantic engineering, but here we will focus on just the former. According to Partee [], a formal semanticist, advocate, and torch-bearer for Montague, the chief interest of Montague's approach (as far as his contemporary *linguists* were concerned) lay in the following ideas:

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

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

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

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

In Section 1 of *Universal Grammar*, Montague's first paragraph establishes common notions of relation and function – the latter he calls *operation*, to distinguish the n-ary case from the unary case which he calls *function*. This is all done with ordinals indexing lists of elements of an arbitrary but fixed set A, which leads later on to nested indices and redundancy by repeated

mention of A. We will try to avoid these issues going forward by eliding some data where there is no confusion, following common modern practice.

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

Definition 2.5.2 is equivalent to asking for all projections. Definitions 2.5.2 and 2.5.4 together characterise Montagovian algebras as (concrete) clones [].

These are (concrete) clones [] and their homomorphisms. In modern terms, (abstract) clones known to be in bijection with Lawvere theories [] ——- .

2.5.3 On Syntax

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

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

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

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

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

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

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

Definition 2.5.1 (Generating data of an Algebra)**.** Let A be the carrier set, and F_γ be a set of functions $A^k \to A$ for some $k \in \mathbb{N}$, indexed by $\gamma \in \Gamma$. Denoted $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$

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

Definition 2.5.2 (Identities)**.** A family of operations populated, for all $n, m \in \mathbb{N}$, $n \leq m$, by an m-ary operation $I_{n,m}$, defined on all m-tuples as

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

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

Definition 2.5.3 (Constants)**.** For all elements of the carrier $x \in A$, and all $m \in \mathbb{N}$, a constant operation $C_{x,m}$ defined on all m-tuples a as:

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

Definition 2.5.4 (Composition)**.** Given an n-ary operation G, and n instances of m-ary operations $H_{1 \leq i \leq n}$, define the composite $G(H_i)_{1 \leq i \leq n}$ to act on m-tuples a by:

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

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

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

Definition 2.5.5 (Polynomial Operations)**.** The polynomial operations over an algebra $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ are defined to be smallest class K containing all $F_{\gamma \in \Gamma}$, identities, constants, closed under composition.

Definition 2.5.6 (Homomorphism of Algebras)**.** h is a homomorphism from $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ *into* $\langle B, G_\gamma \rangle_{\gamma \in \Delta}$ iff

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

2.

be typed in a way that permits operations that arbitrarily change types, which appears to defeat the purpose. We doubt these artefacts are intentional, so we will interpret Montague assuming his intent was a type-system as we would recognise one today.

By an evident extension of [Prop 3.51] to the typed case, a *disambiguated language* is a multi-sorted Lawvere theory without relations, where the sorts are generated from products of a pointed set $(\Delta, \delta_0 : 1 \to \Delta)$.

3

String Diagrams for Text

3.1 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.

placeholder

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.

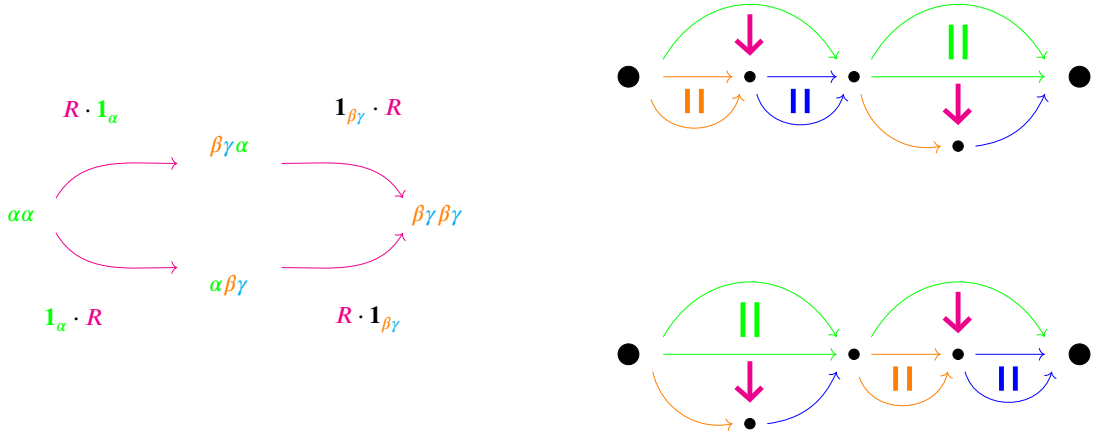
Mathematicians, computer scientists, and physicists may have good reasons to work with weak n-categories [], but what is the value proposition for formal linguists? A philosophical draw for the formal semanticist is that insofar as semantics is synonymy – the study of when expressions are equivalent – 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.

3.1.1 String-rewrite systems as 1-object-2-categories

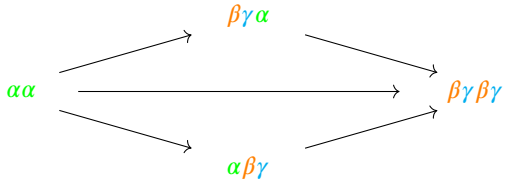
Say we have an alphabet $\Sigma := \{\alpha, \beta, \gamma\}$. Then the Kleene-star Σ^* consists of all strings (including the empty string ϵ) made up of Σ, and we consider formal languages on Σ to be subsets of Σ^*. Another way of viewing Σ^* is as the free monoid generated by Σ under the binary concatenation 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 3.1.1.

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



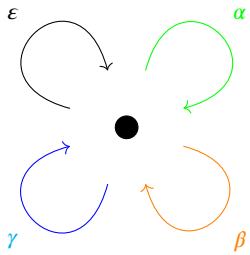


Figure 3.1: The category in question can be visualised as a commutative diagram.

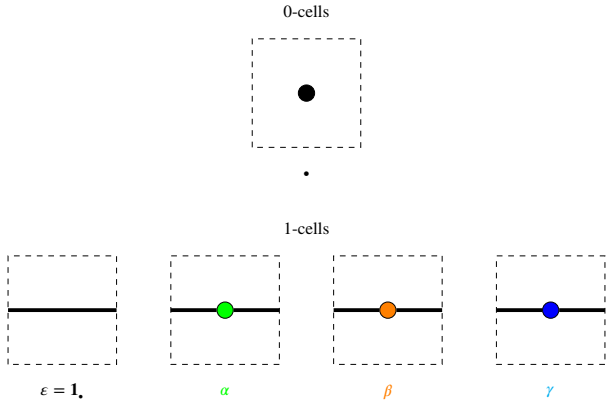
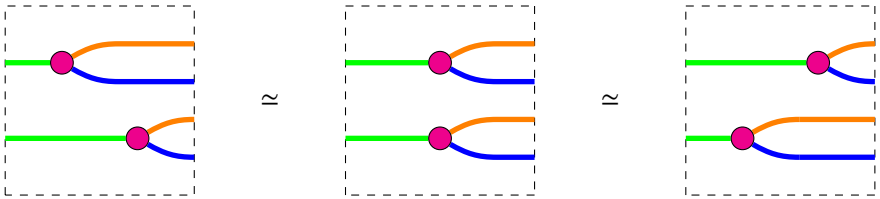


Figure 3.2: When there are too many generating morphisms, we can instead present the same data as a table of n-cells; there is a single 0-cell $\star$, and three non-identity 1-cells corresponding to α, β, γ, each with source and target 0-cells $\star$. Typically identity morphisms can be omitted from tables as they come for free. Observe that composition of identities enforces the behaviour of the empty string, so that for any string x, we have $\epsilon \cdot x = x = \epsilon \cdot x$.

Figure 3.3: For a concrete example, we can depict the string $\alpha \cdot \gamma \cdot \gamma \cdot \beta$ as a morphism in a commuting diagram.

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



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

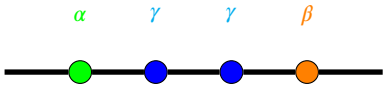


Figure 3.4: The string-diagrammatic view, where ⋆ is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

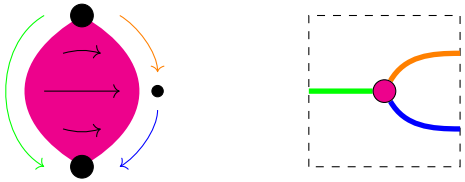
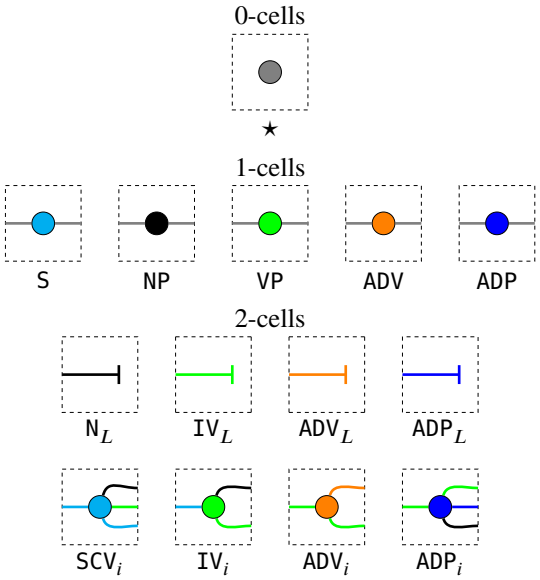


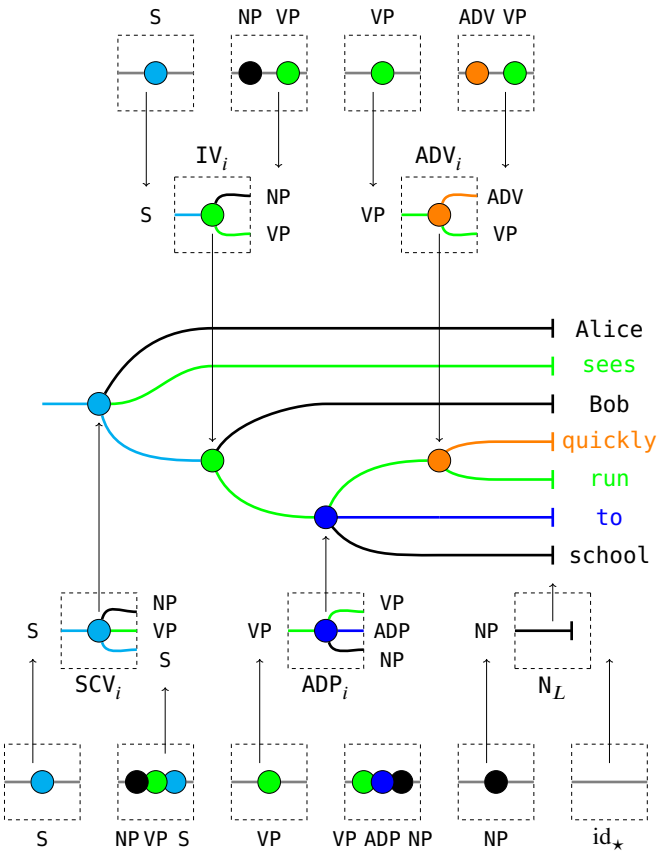
Figure 3.5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual of this view gives us a string diagram for the 2-cell R.

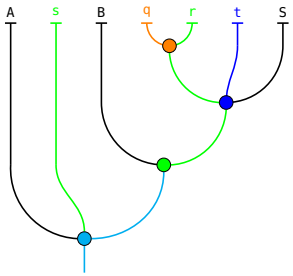
3.1.2 A context free grammar to generate `Alice sees Bob quickly run to school`

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. 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 ⋆, 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.



Consider the sentence `Alice sees Bob quickly run to school`, which we take to be generated by the following context-free grammar derivation, read from left-to-right. We additionally depict the breakdown of the derivation in terms of rewrites of lower dimension from our signature.


S
NP VP
VP
ADV VP
IV$_i$
ADV$_i$
S
NP
VP
VP
ADV
VP
Alice
sees
Bob
quickly
run
to
school
S
NP
VP
S
SCV$_i$
VP
VP
ADP
NP
ADP$_i$
NP
N$_L$
S
NP VP S
VP
VP ADP NP
NP
id$_\star$


A
s
B
q
r
t
S

3.1.3 Tree Adjoining Grammars

Here is a formal but unenlightening definition of *elementary* tree adjoining grammars which we will convert to diagrams. We will deal with the extensions of links and local constraints to adjoining shortly.

Definition 3.1.1 (*Elementary* Tree Adjoining Grammar: Classic Computer Science style)**.** *A **TAG** is a tuple $(\mathcal{N}, \mathcal{N}^{\downarrow}, \mathcal{N}^{*}, \Sigma, \mathcal{I}, \mathcal{A})$. These denote, respectively:

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

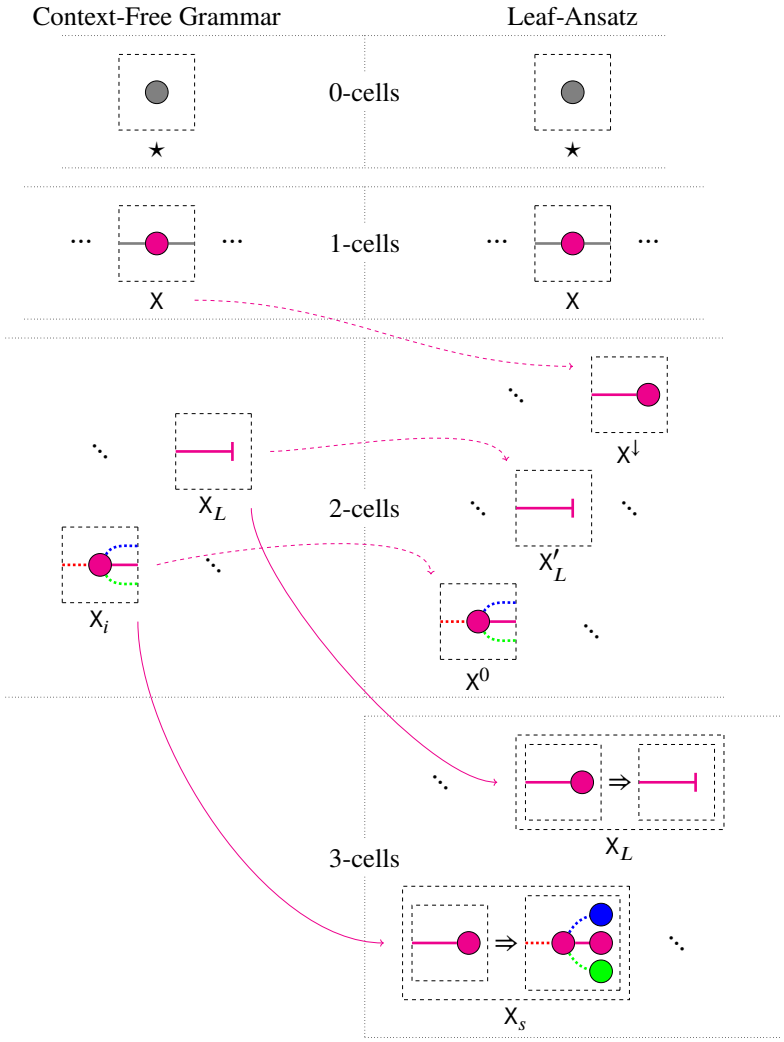
There are two operations to build what are called *derived trees* from elementary and derived trees. These operations are called *substitution* and *adjoining*.

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

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

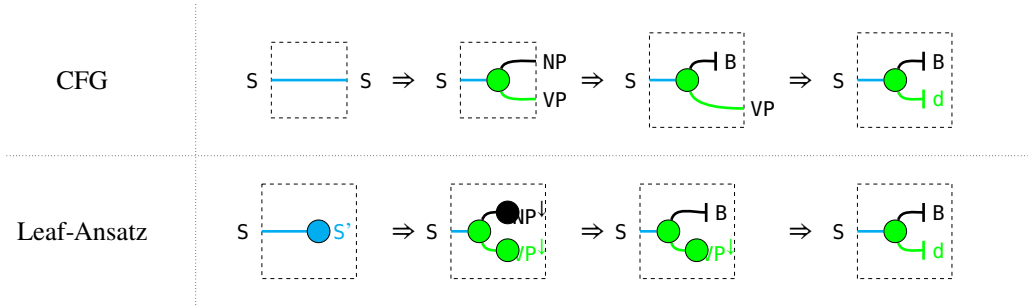
Construction 3.1.2 (Leaf-Ansatz of a CFG)**.** Given a signature $\mathfrak{G}$ for a CFG, we construct a new signature $\mathfrak{G}'$ which has the same 0- and 1-cells as $\mathfrak{G}$. See the dashed magenta arrows in the schematic below. For each 1-cell wire type `X` of $\mathfrak{G}$, we introduce 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 see the solid magenta

arrows in the schematic below. 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 $\mathfrak{G}'$.



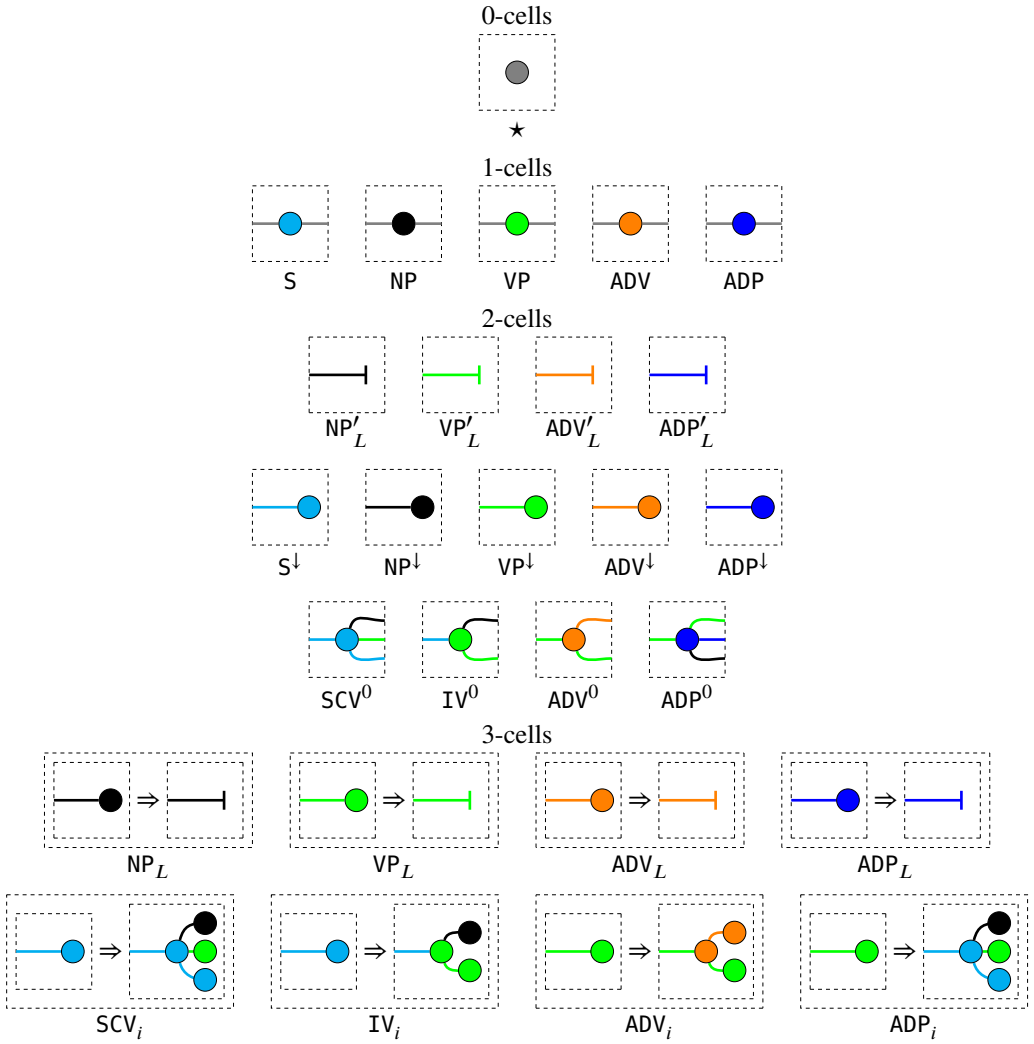
Example 3.1.3. The leaf-ansatz construction just makes formal the following observation: there are multiple equivalent ways of modelling terminal symbols in a rewrite system considered string-diagrammatically. One way (which we have already done) is to treat non-terminals as wires and terminals as effects, so that the presence of an open wire available for composition visually indicates non-terminality. Another (which is the leaf-ansatz construction) treats all symbols in a rewrite system as leaves, where bookkeeping the distinction between (non-)terminals

occurs in the signature. So for a sentence like `Bob drinks`, we have the following derivations that match step for step in the two ways we have considered.



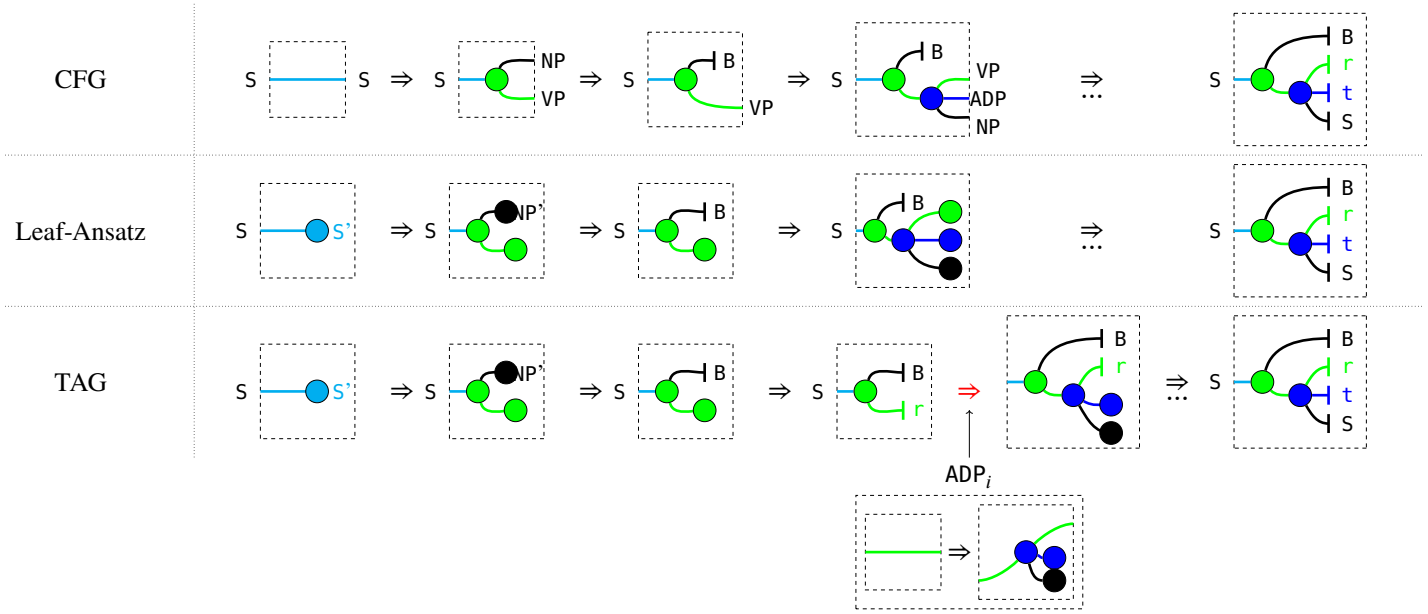
Proposition 3.1.4 (Leaf-ansatzes of CFGs are precisely TAGs with only initial trees and substitution)**.** *Proof.* By construction. Consider a CFG given by 2-categorical signature $\mathfrak{G}$, with leaf-ansatz signature $\mathfrak{G}'$. The types $\mathtt{X}$ of $\mathfrak{G}$ become substitution 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 $\mathfrak{G}$. □

Example 3.1.5 (Leaf-ansatz signature of `Alice sees Bob quickly run to school` CFG)**.**



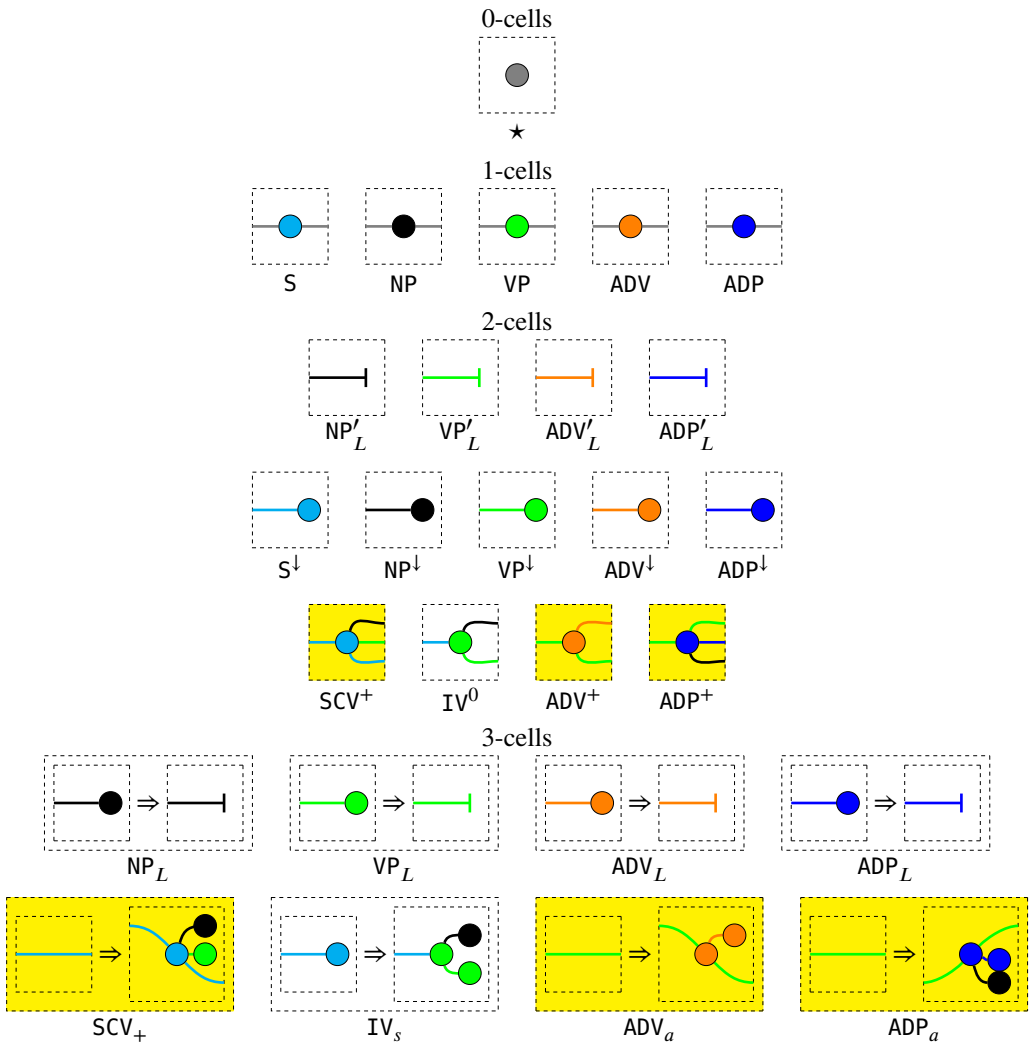
Example 3.1.6 (Adjoining is sprouting subtrees in the middle of branches)**.** One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is 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, as follows:



Example 3.1.7 (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 3.1.8. 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 3.1.4 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. See the example above; the highlighted 3-cells of $\mathfrak{G}'$ are obtained systematically from

the auxiliary 2-cells as follows: the root and foot nodes $\mathsf{X}, \mathsf{X}^\star$ 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-X open wires Y with their leaf-ansatzes $\mathsf{Y}^\downarrow$. This establishes a correspondence between any 2-cells of $\mathfrak{G}$ considered as auxiliary trees in $\mathfrak{G}'$. □

3.1.4 Tree adjoining grammars with local constraints

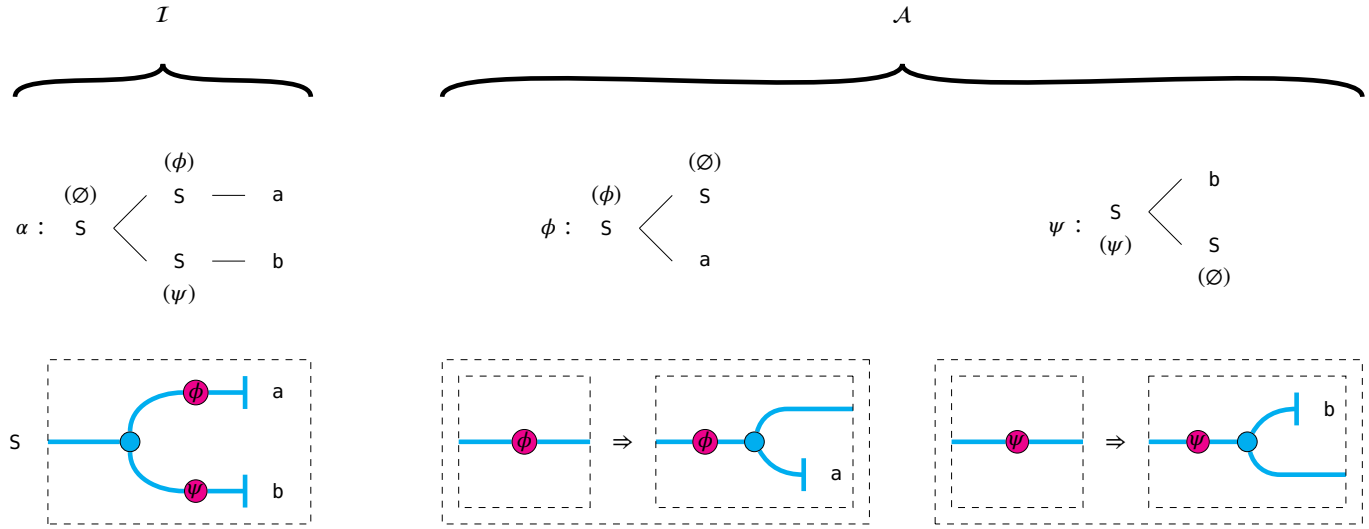
The usual conception of TAGs includes two extensions to the basic definition presented above. First, there may be *local constraints* on adjoining, which only allows certain trees to be adjoined at certain nodes. Second, TAGs may have links, which are extra edges between nodes obeying a c-command condition.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

Definition 3.1.9 (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.

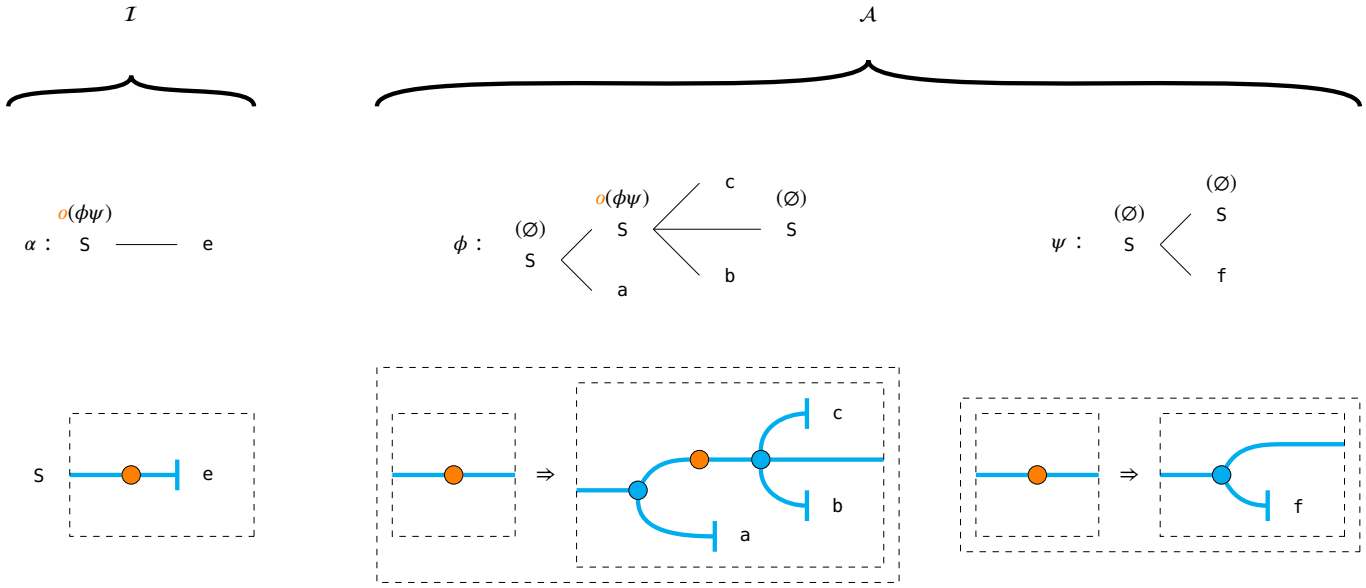
The n-categorical approach easily accommodates local constraints. For (SA), whereas before we take the source of adjoining rewrites to be identities, we can instead introduce ansatz endomorphisms that are rewritable to the desired subsets. For (NA), we assert that identities have no rewrites (NA). For (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.

Example 3.1.10 (Selective and null adjoining diagrammatically)**.** The below is 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.



Example 3.1.11 (Obligatory adjoining diagrammatically)**.** The below is 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.



3.1.5 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.

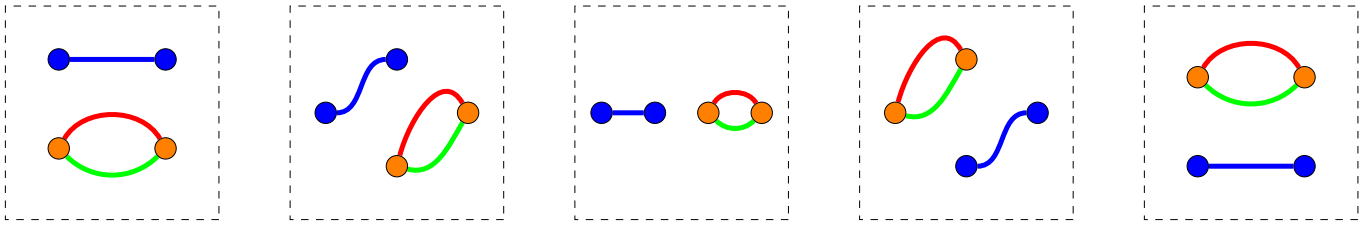
First we observe that in a 1-object-2-categorical setting, a morphism from the identity ε on the base object $\star$ to itself would, in our analogy with string-rewrite systems, be a rewrite from the empty string to itself. For example, a rewrite R may introduce a symbol from the empty string and then delete it. A rewrite S may create a pair of symbols from nothing and then annihilate them.

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

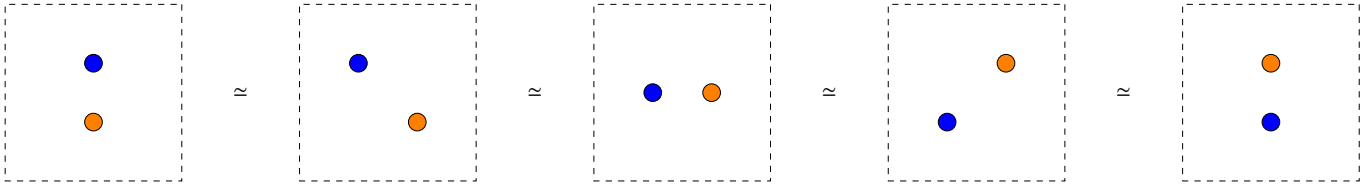
In our analogy with string rewrite systems, we might like that the following rewrites are equivalent, while respecting that they are not equal.

1. Start and finish R and S concurrently, R on the left and S on the right: $\varepsilon = \varepsilon \cdot \varepsilon \mapsto x \cdot a \cdot b \mapsto \varepsilon \cdot \varepsilon = \varepsilon$
2. Start R first. Start S on the right concurrently as R finishes: $\varepsilon \mapsto x = x \cdot \epsilon \mapsto \epsilon \cdot a \cdot b = a \cdot b \mapsto \epsilon$
3. Start and finish R first, and then start and finish S: $\varepsilon \mapsto x \mapsto \varepsilon \mapsto a \cdot b \mapsto \varepsilon$
4. Start S first. Start R on the right concurrently as S finishes: $\varepsilon \mapsto a \cdot b = a \cdot b \cdot \varepsilon \mapsto \varepsilon \cdot x = x \mapsto \varepsilon$
5. Do S and R concurrently, S on the left and R on the right: $\varepsilon = \varepsilon \cdot \varepsilon \mapsto a \cdot b \cdot x \mapsto \varepsilon \cdot \varepsilon = \varepsilon$

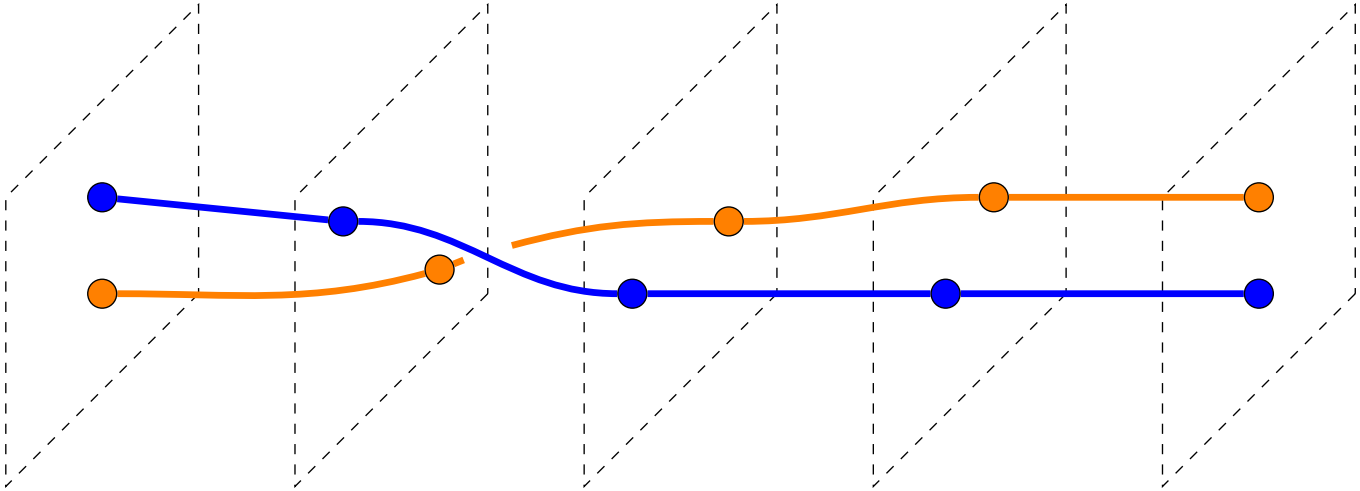
Diagrammatically, we may represent these rewrites from left to right as follows, representing x, a, b as blue, red, and green wires respectively:



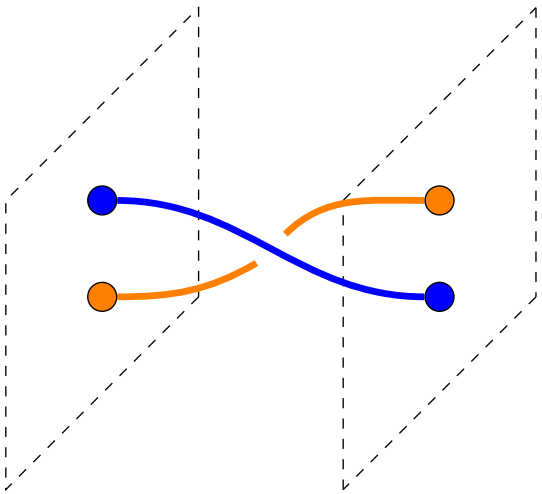
Such rewrites from the empty string to itself are more generally called *scalars* in the monoidal setting, viewed 2-categorically. We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument [] is that dots may circle around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

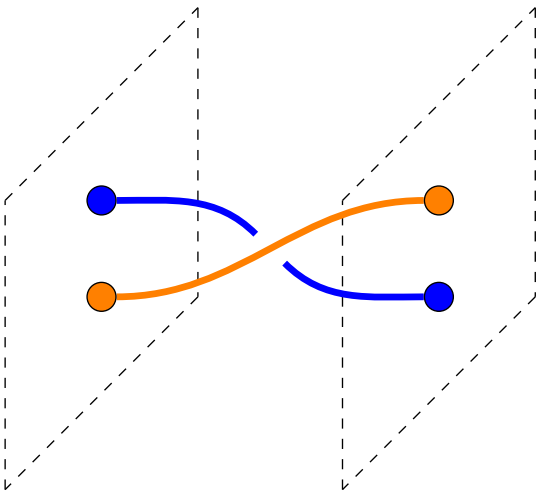


We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette.

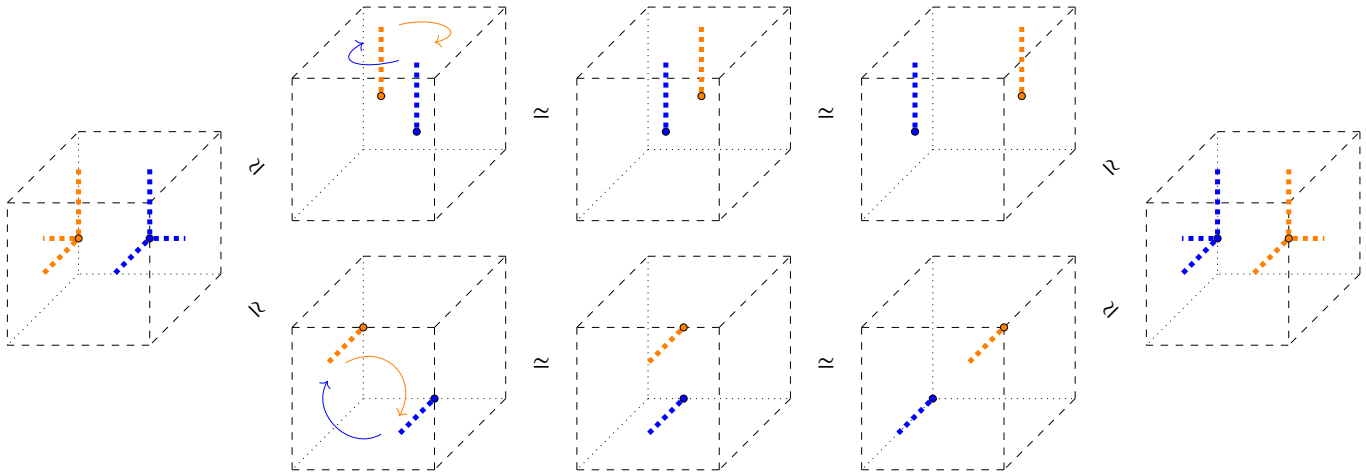


Up to processive isotopies [], which are continuous bijective transformations that don't let wires double back on themselves, we can identify two different braidings that are not continuously deformable to one another in the 3-dimensional space of the vignette. We may depict the two braidings as follows, where wires either go over or under one another.



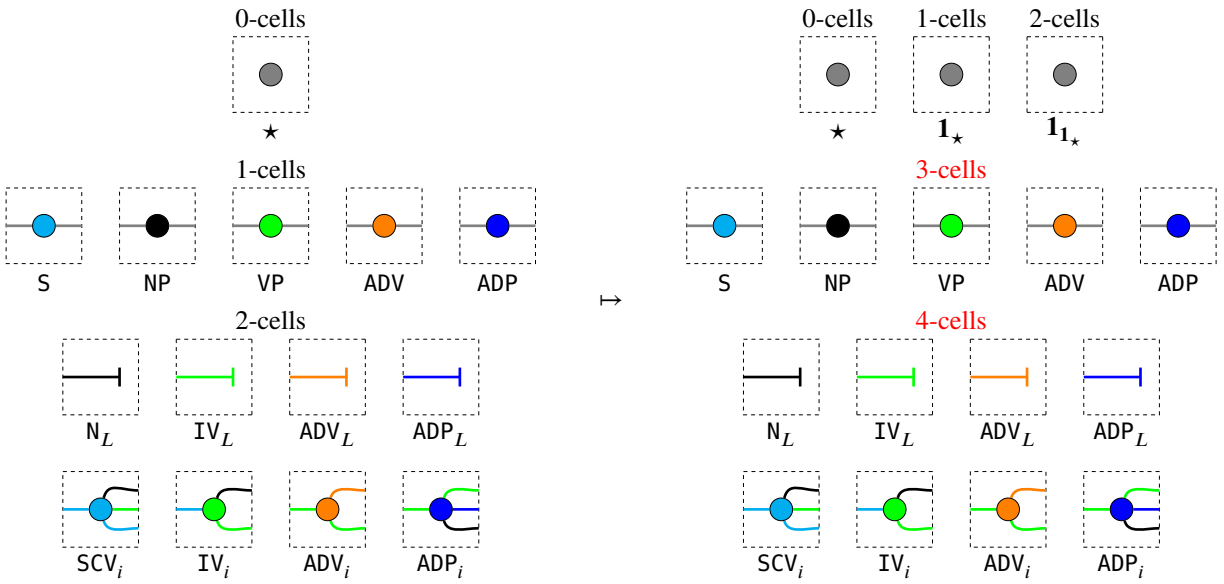


Now we have a setting in which we can consider wires swapping places. However, having two different ways to swap wires presents a complication. While useful for knot-theory, in symmetric monoidal categories we don't want this distinction. Perhaps surprisingly, this distinction is eliminated if we consider 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). 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 $\mathbf{1}_{1_\star}$, in essence turning dots on a line into dots in a volume; this procedure is called *suspension* []. 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, and the first non-identity cell has degree 4. Symmetric monoidal categories are equivalently seen as 1-object-4-categories []. To summarise, by appropriately suspending the signature, we obtain a formal way to power up our diagrams to permit twisting wires, as in symmetric monoidal categories.

Remark 3.1.12 (**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.

3.1.6 TAGs with links

Definition 3.1.13 (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.

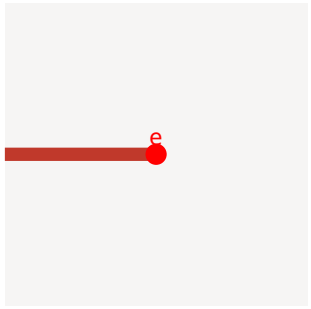
Example 3.1.14. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n-categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out. The trees given in Examples 2.4 are:

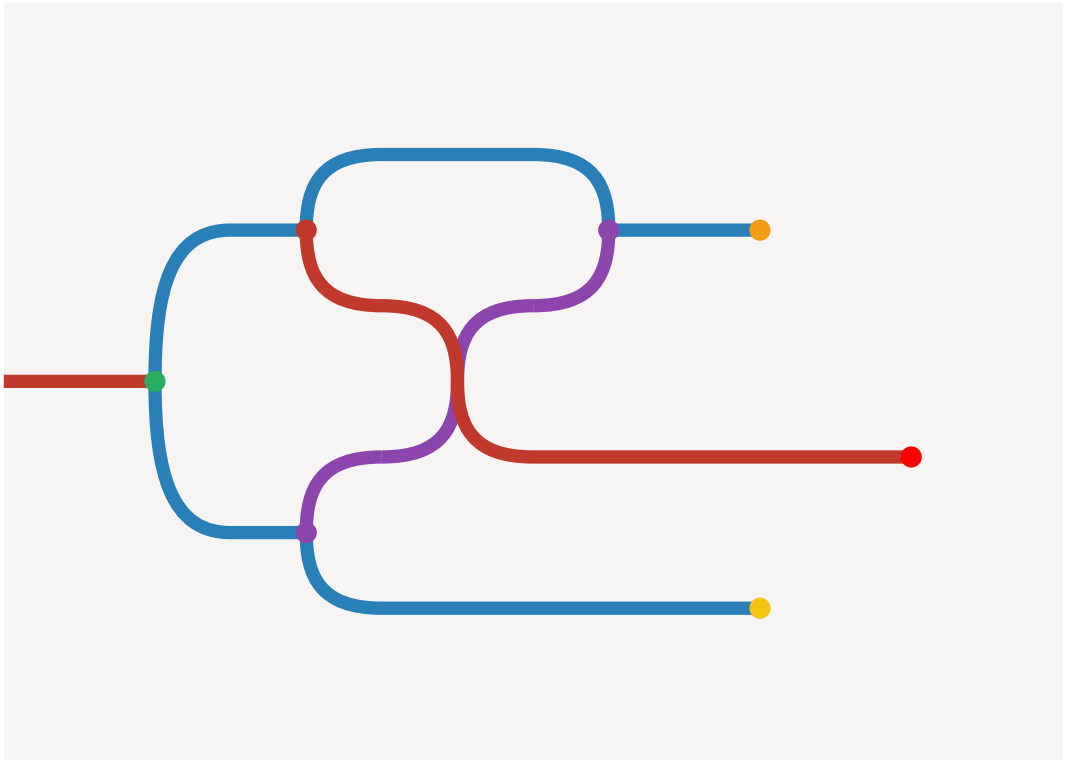
placeholder

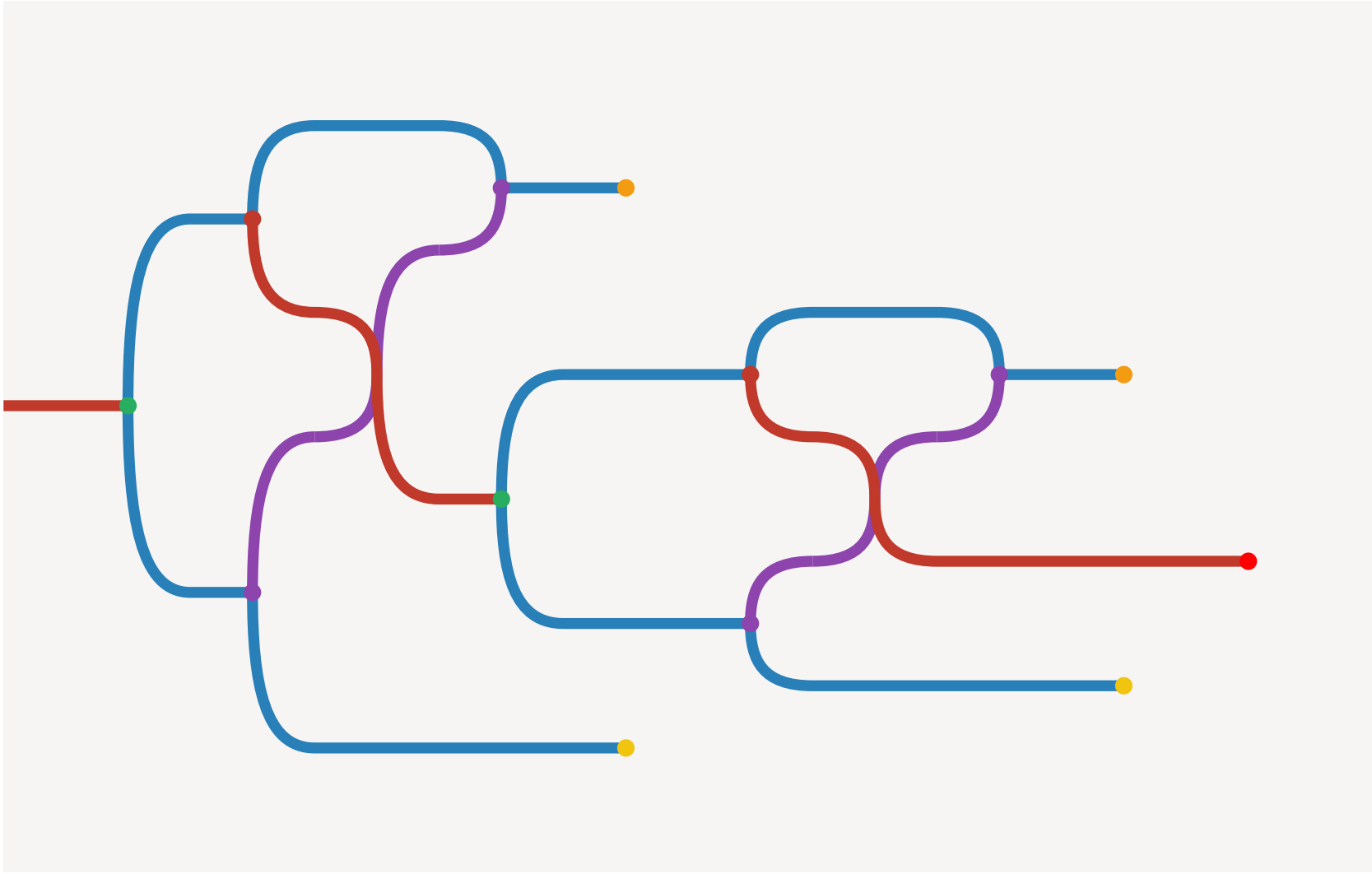
Which we interpret as expressions comprised of the following signature:

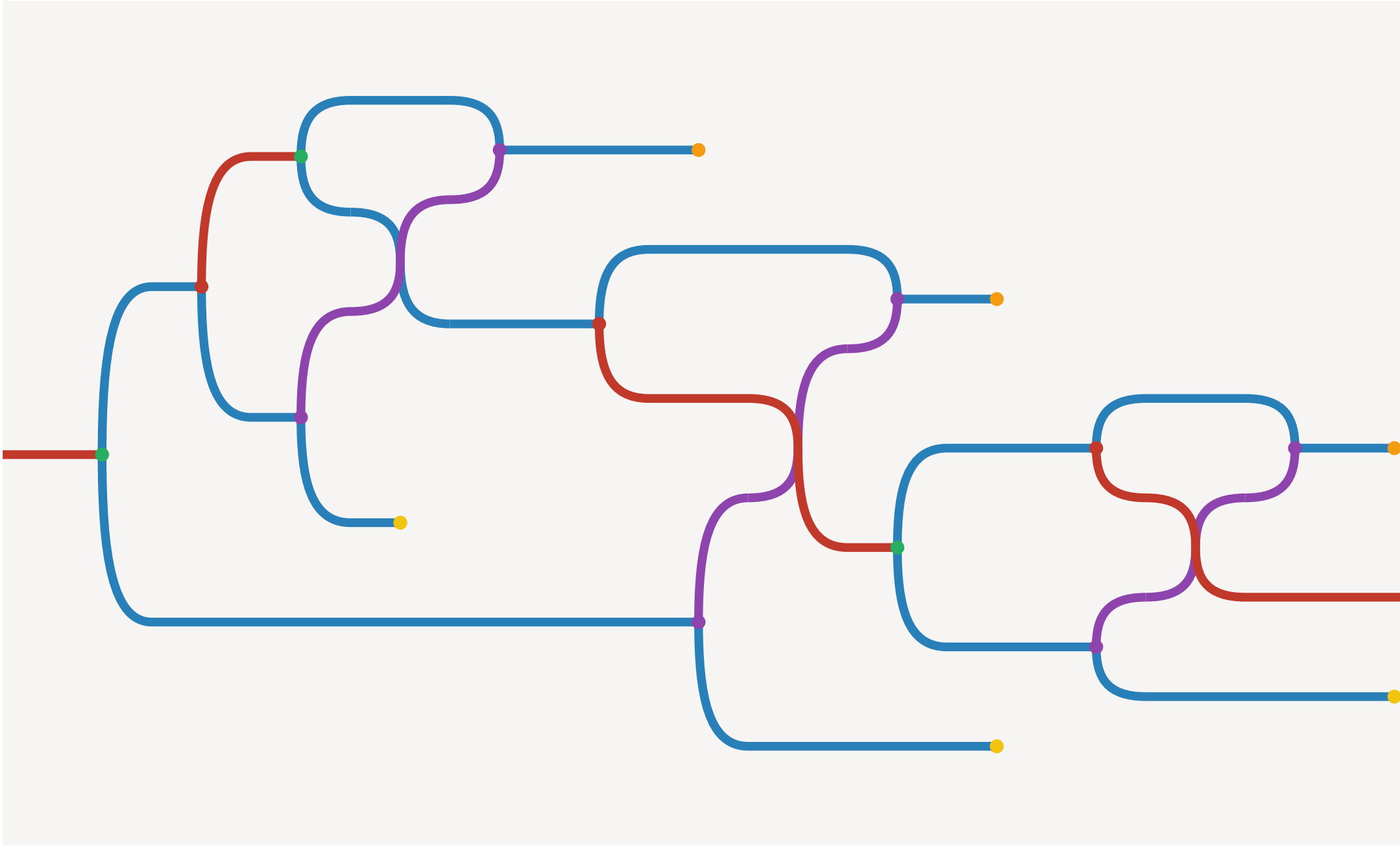
$$placeholder$$

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.









N.B. In practice when using `homotopy.io` for the symmetric monoidal setting, it is simpler to suspend symmetric monoidal signatures to begin at 4-cells rather than 3-cells. The reason

for this is that under- and over-braids still exist in the symmetric monoidal setting, and while sequentially composed braids are homotopically equivalent to the pair of identities, they are not uniquely so, thus these homotopies must be input manually. By beginning at 4-cells (or higher, due to the stabilisation hypothesis []), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

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

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

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

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

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

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

We spell out how this data becomes an n-categorical signature by enumerating cell dimensions:

0. A single object $\star$
1. None.
2. None.
3. - For each $\mathbf{T} \in \mathcal{N}$, a cell $\mathbf{T} : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$.
   - $\top : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$ A wire for terminal symbols.

- For each $\mathbf{L} \in \mathfrak{L}$, a cell $\mathbf{L} : \mathbf{1}_{\mathbf{1}_\star} \to \mathbf{1}_{\mathbf{1}_\star}$.

4. We distinguish between *atomic* and *composite* generators at this level. The trees themselves are composites of *atomic* generators. The *atomic* generators are:
   - For each node n that occurs in either $\mathcal{I}$ or $\mathcal{A}$, we populate cells by a case analysis:
     - If n is a terminal $\sigma \in \Sigma$, we create a cell $\sigma : \top \to \mathbf{1}_{\mathbf{1}_\star}$.
     - If $n = (\mathbf{T}, \mathbf{S}, \dagger, \bar{*})$, we create $\mathbf{S}^{\square}_{\mathbf{T}} : \mathbf{T} \to \mathbf{T}$ if the node is marked obligatory ($\dagger = \square$), and a cell $\mathbf{S}^{\Diamond}_{\mathbf{T}} : \mathbf{T} \to \mathbf{T}$ otherwise.
     - If $n = (\mathbf{T}, \mathbf{S}, \dagger, *)$, it is a foot note, for which we create a cell $\mathbf{S}^{\square}_{\mathbf{T}} : \mathbf{T} \to \mathbf{1}_{\mathbf{1}_\star}$ if the node is marked obligatory ($\dagger = \square$), and a cell $\mathbf{S}^{\Diamond}_{\mathbf{T}} : \mathbf{T} \to \mathbf{1}_{\mathbf{1}_\star}$ otherwise.
   - For each $\mathbf{L} \in \mathfrak{L}$ (which is also a type $\mathbf{T} \in \mathcal{N}$) a pair of cells $\mathbf{T}^{\mathbf{L}} := \mathbf{T} \to \mathbf{T} \otimes \mathbf{L}$ and $\mathbf{T}_{\mathbf{L}} := \mathbf{L} \otimes \mathbf{T} \to \mathbf{T}$.
   - For each node p of type $\mathbf{T}_p$ in either $\mathcal{I}$ or $\mathcal{A}$ with a nonempty left-to-right list of children $C_p :=< c_1, c_2, \cdots c_i, c \ldots c_n >$ with types $\mathbf{T}_i$, a branch cell $C_p : \mathbf{T}_p \to \bigotimes_{i=1}^{n} \mathbf{T}_i$.

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

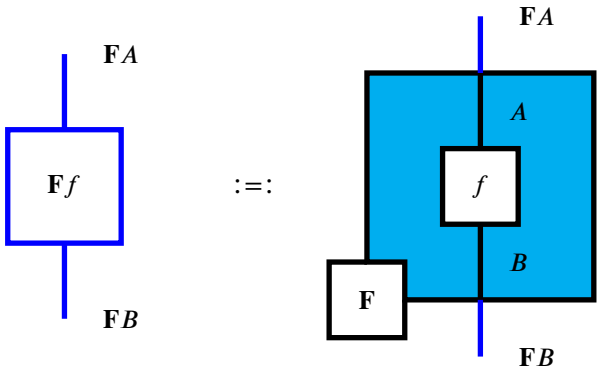
5.

3.1.7 Discrete Monoidal Fibrations

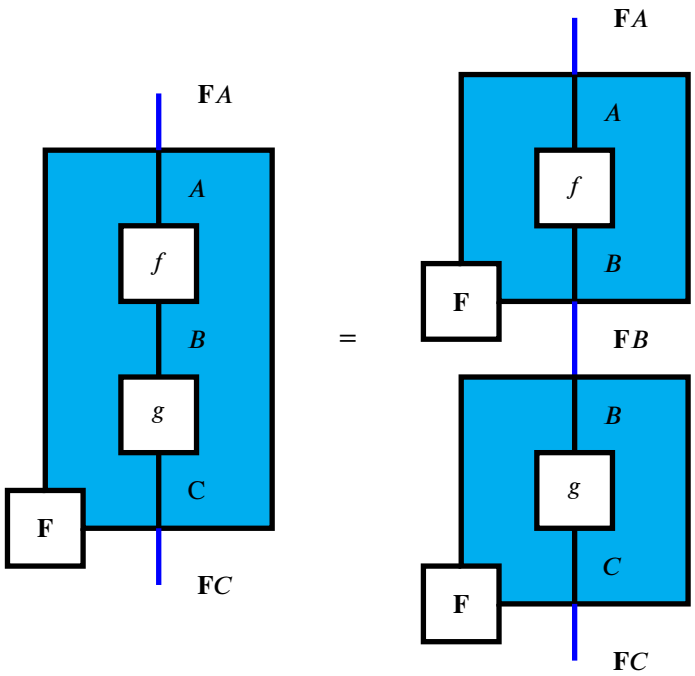
We introduce the concept of a discrete monoidal fibration: a mathematical bookkeeping tool that relates kinds of choices speakers and listeners make when generating and parsing text respectively. We proceed diagrammatically. The first concept is that of a *monoidal functor box*.

Scholium 3.1.16. Functor boxes are from [meillies]. Expressing the coherence conditions of monoidal functors using equations involving functor boxes as below is not new [meillies]. The idea of a functor being simultaneously monoidal and a fibration is not new [monoidalfibration]. What is new is minor: the express requirement that the lifts of the fibration satisfy interchange, which is in general not guaranteed by just having a functor be monoidal and a (even discrete) fibration [fosco].

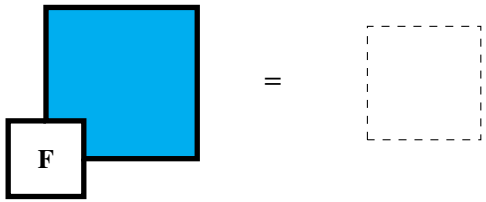
Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \to \mathcal{D}$. Then we have the following diagrammatic representation of a morphism $\mathbf{F}A \xrightarrow{\mathbf{F}f} \mathbf{F}B$ in $\mathcal{D}$:

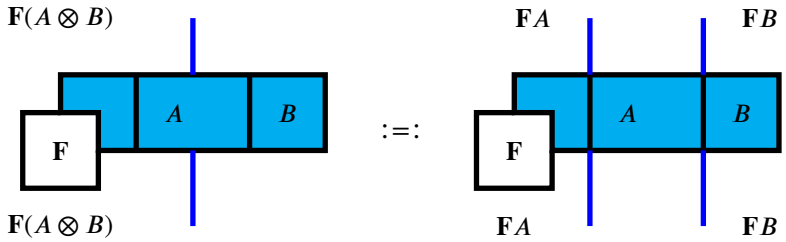


The use of a functor box is like a window from the target category $\mathcal{D}$ into the source category $\mathcal{C}$; when we know that a morphism in $\mathcal{D}$ is the image under $\mathbf{F}$ of some morphism in $\mathcal{C}$, the functor box notation is just a way of presenting all of that data at once. Since $\mathbf{F}$ is a functor, we must have that $\mathbf{F}f ; \mathbf{F}g = \mathbf{F}(f ; g)$. Diagrammatically this equation is represented by freely splitting and merging functor boxes vertically.



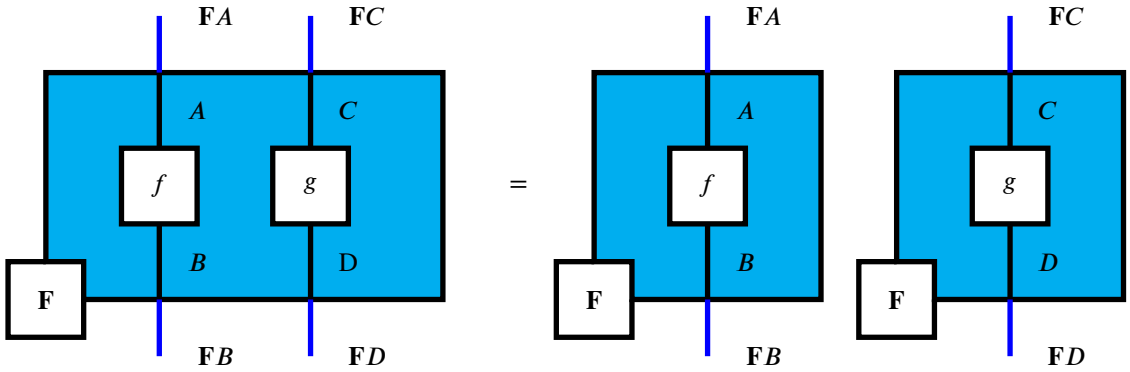
Assume that **F** is strict monoidal; without loss of generality by the strictification theorem [], this lets us gloss over the associators and unitors. For **F** to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A \otimes_D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



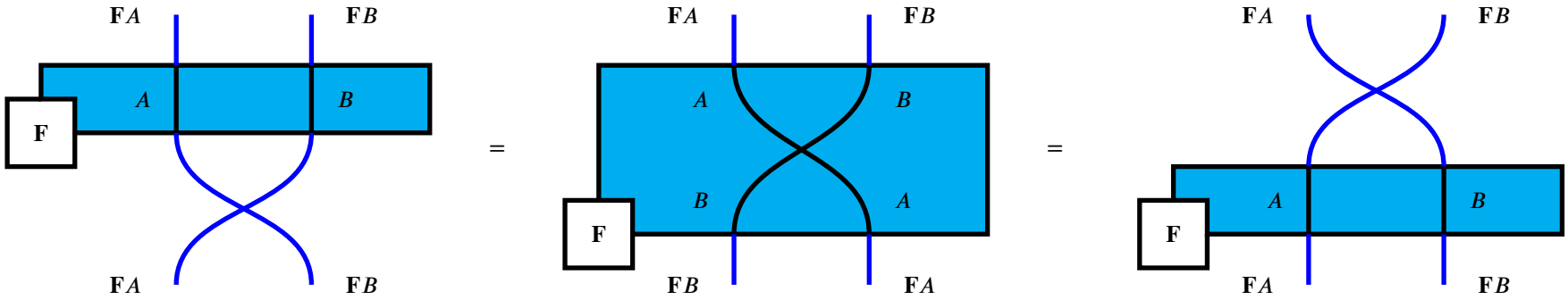


What remains is the monoidality of **F**, which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes

horizontally; analogously to how splitting vertically is the functor-boxes' way of respecting sequential composition, splitting horizontally is how they respect parallel composition.

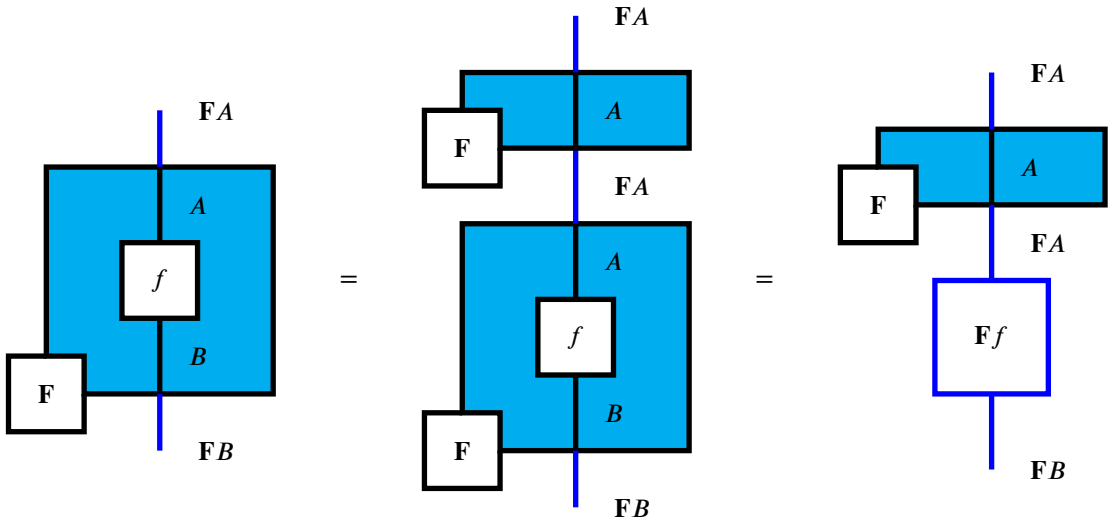


And for when we want **F** to be a (strict) symmetric monoidal functor, we are just asking that boxes and twists do not get stuck on one another.



Remark 3.1.17. To motivate fibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a

functor box out of the bottom:



When can we do the reverse? That is, take a morphism in $\mathcal{D}$ and *slide it into* a functor box? We know that in general this is not possible, because not all morphisms in $\mathcal{D}$ may be in the image of $\mathbf{F}$. So instead we ask "under what circumstances" can we do this for a functor $\mathbf{F}$? The answer is when $\mathbf{F}$ is a discrete fibration.

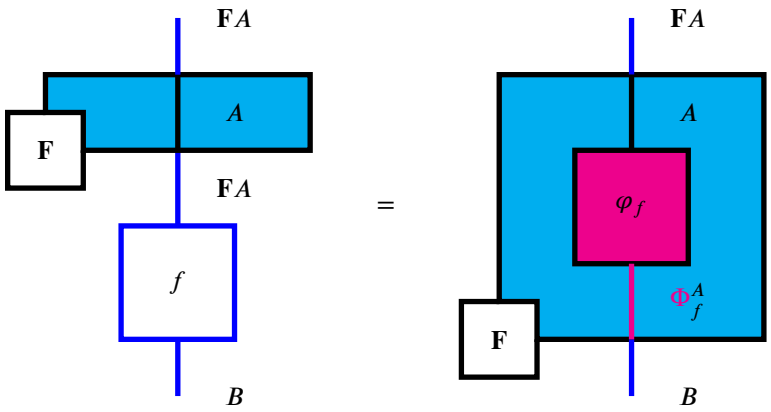
Definition 3.1.18 (Discrete opfibration)**.** $\mathbf{F} : \mathcal{C} \to \mathcal{D}$ is a *discrete fibration* when:

for all morphisms $f : \mathbf{F}A \to B$ in $\mathcal{D}$ with domain in the image of $\mathbf{F}$...
there exists a unique object Φ_f^A and a unique morphism $\phi_f : A \to \Phi_f^A$ in $\mathcal{C}$...
such that $f = \mathbf{F}\phi_f$.

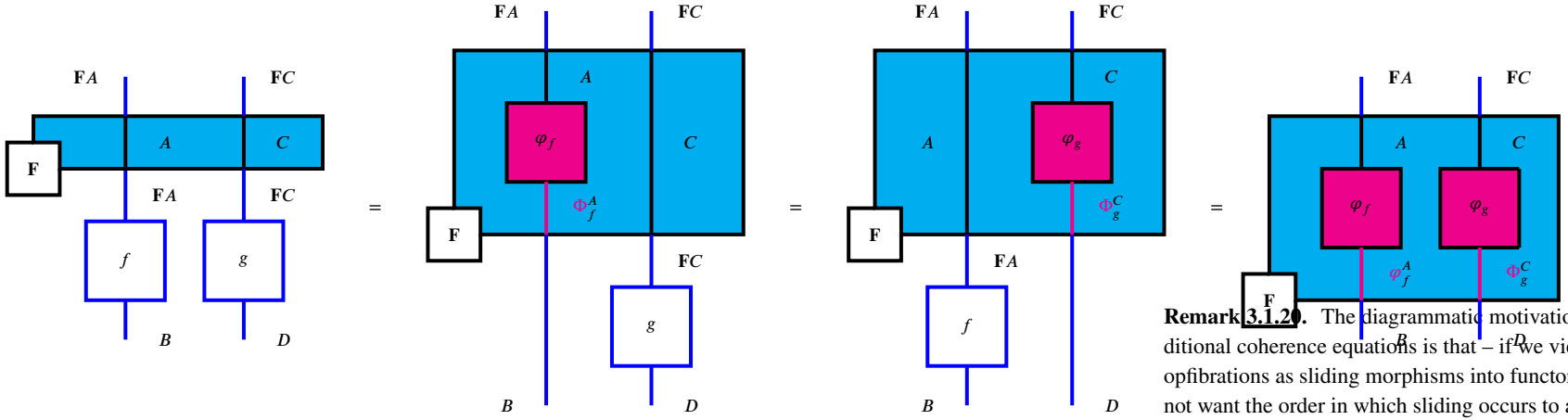
Diagrammatically, we can present all of the above as an equation reminiscent of sliding a morphism *into* a functor box from below.

$$\forall f : \mathbf{F}A \to B \in \mathcal{D}$$

$$\exists! \varphi_f : A \to \Phi_f^A \in \mathcal{C}$$



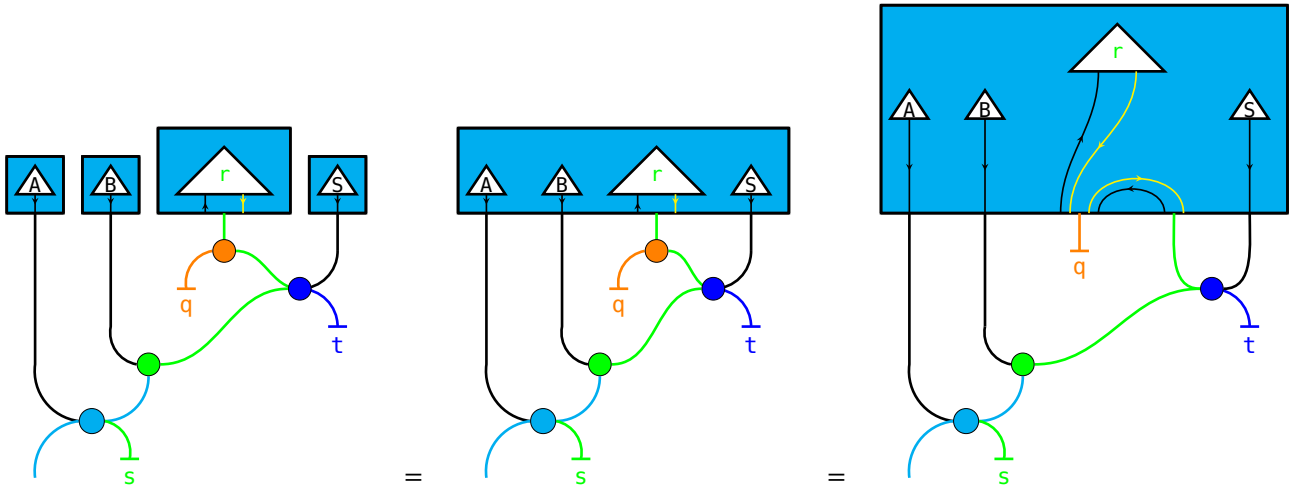
Definition 3.1.19 (Monoidal discrete opfibration)**.** We consider **F** to be a *(strict, symmetric) monoidal discrete opfibration* when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the following equations relating lifts to interchange hold:



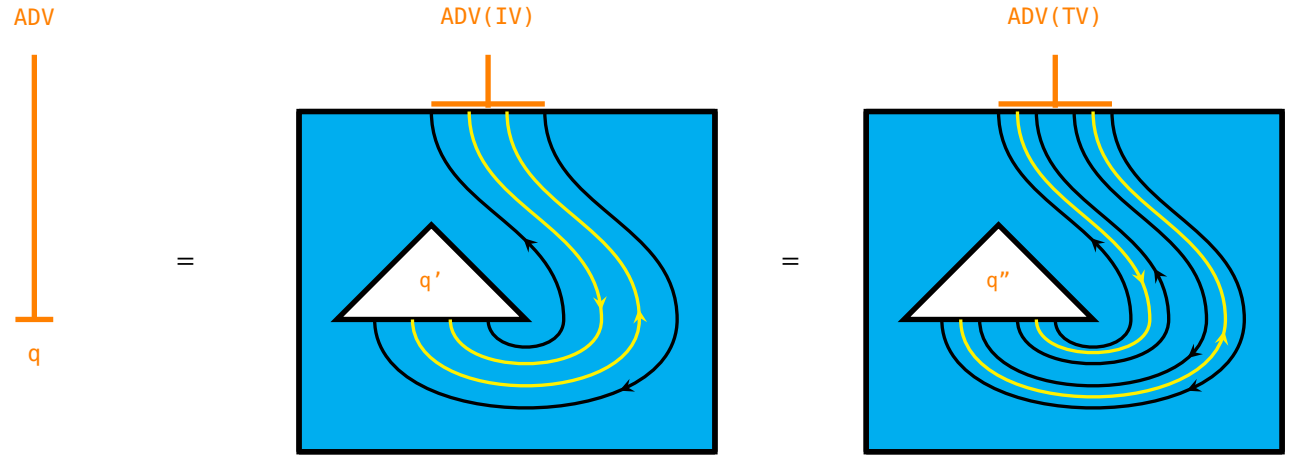
Remark 3.1.20. The diagrammatic motivation for the additional coherence equations is that – if we view the lifts of opfibrations as sliding morphisms into functor boxes – we do not want the order in which sliding occurs to affect the final result. In this way, lifts behave as 'graphical primitives' in the same manner as interchange isotopies and symmetry twists.

3.1.8 Relating the generative grammar to a pregroup grammar via a discrete monoidal fibration

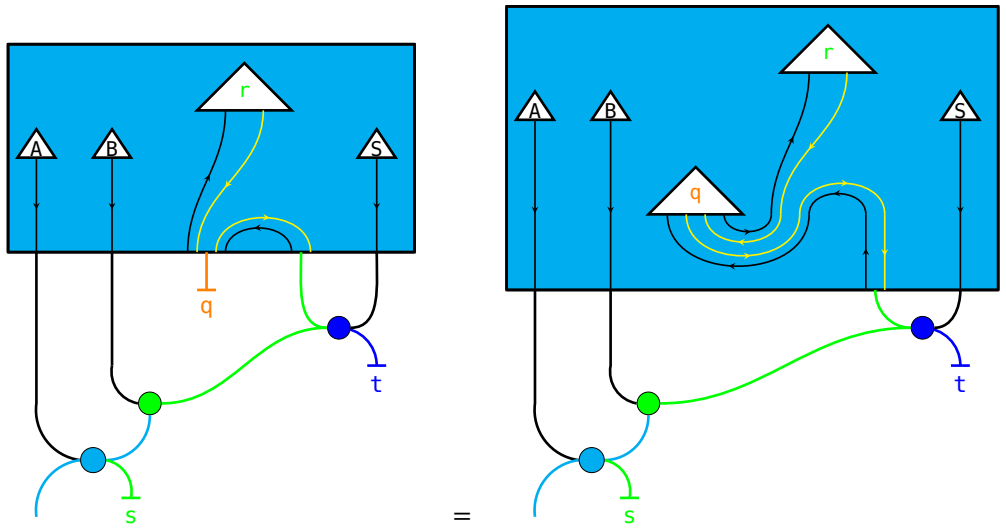
We merge the monoidal functor-boxes and we slide the bottom edge down using the fibration.



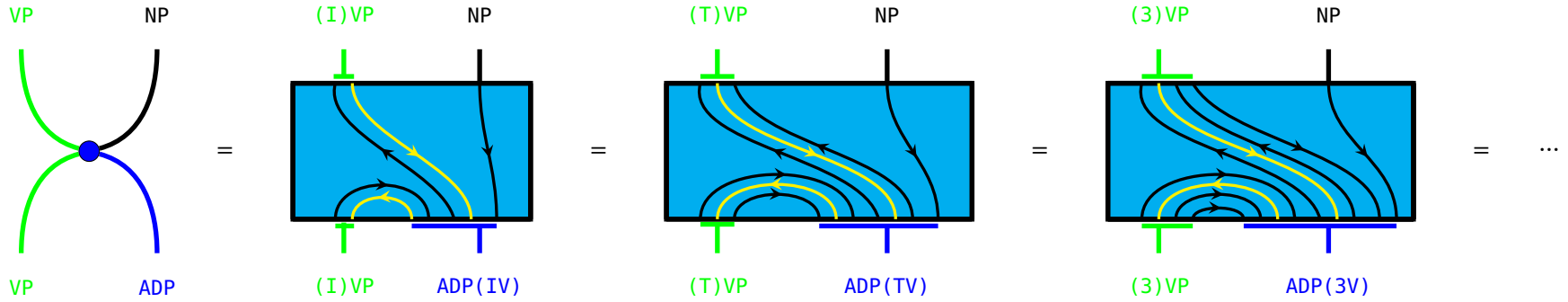
`quickly` could find itself modifying an intransitive (single noun) or transitive (two noun) verb. Suppose that it is the job of some process `q'` to handle intransitive verbs, and similarly `q"` to handle transitive ones. We use the functor for bookkeeping, by asking it to send both `q'` and `q"` to the dependent label $\bar{q}$. Diagrammatically, this assignment is expressed by the following equations:



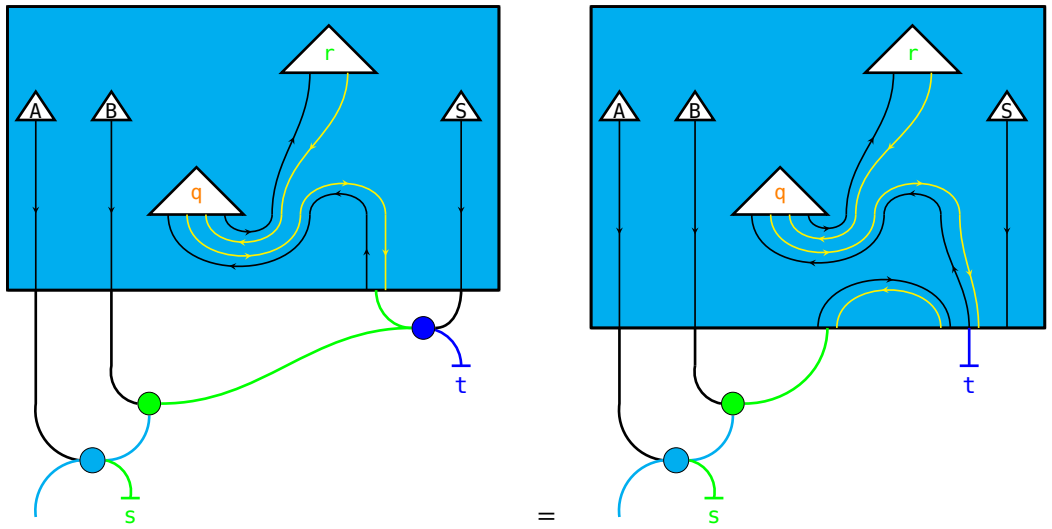
Since the functor is a monoidal discrete fibration, it introduces the appropriate choice of `quickly` when we pull the functor-box down, while leaving everything else in parallel alone.



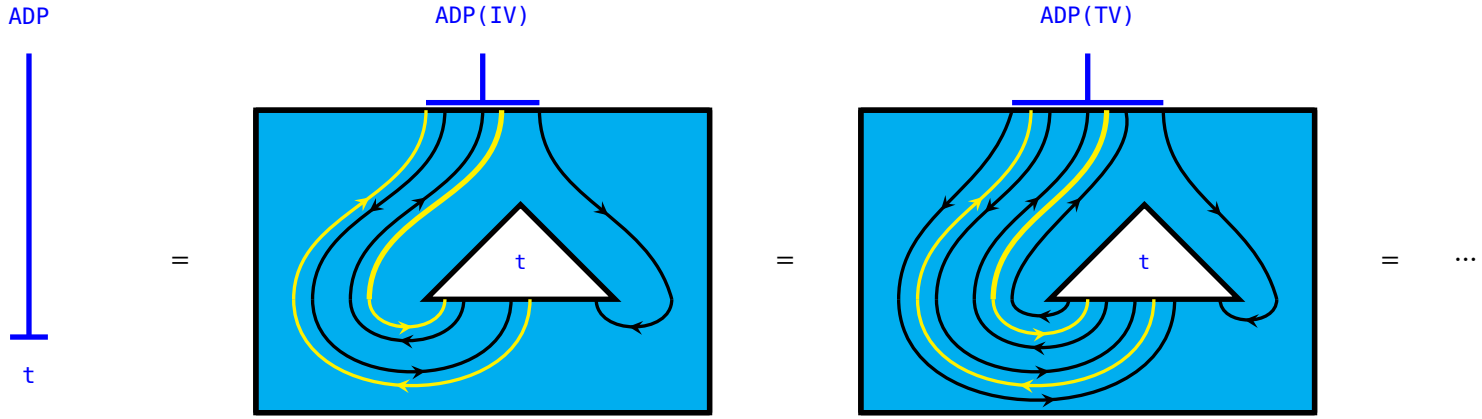
Adpositions can apply for verbs of any noun-arity. We again use the fiber of the functor for bookkeeping by asking it to send all of the following partial pregroup diagrams to the adposition generator. We consider the pregroup typing of a verb of noun-arity $k \geq 1$ to be ${}^{-1}n \cdot s \cdot \underbrace{n^{-1}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



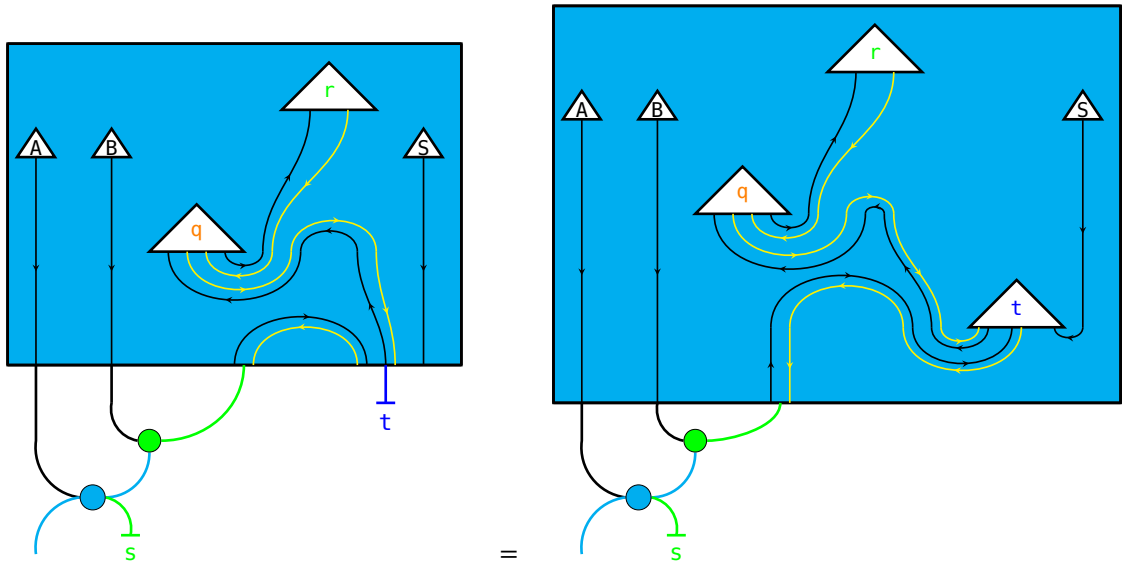
When we pull down the functor-box, the discrete fibration introduces the appropriate choice of diagram from above, corresponding to the intransitive verb case.



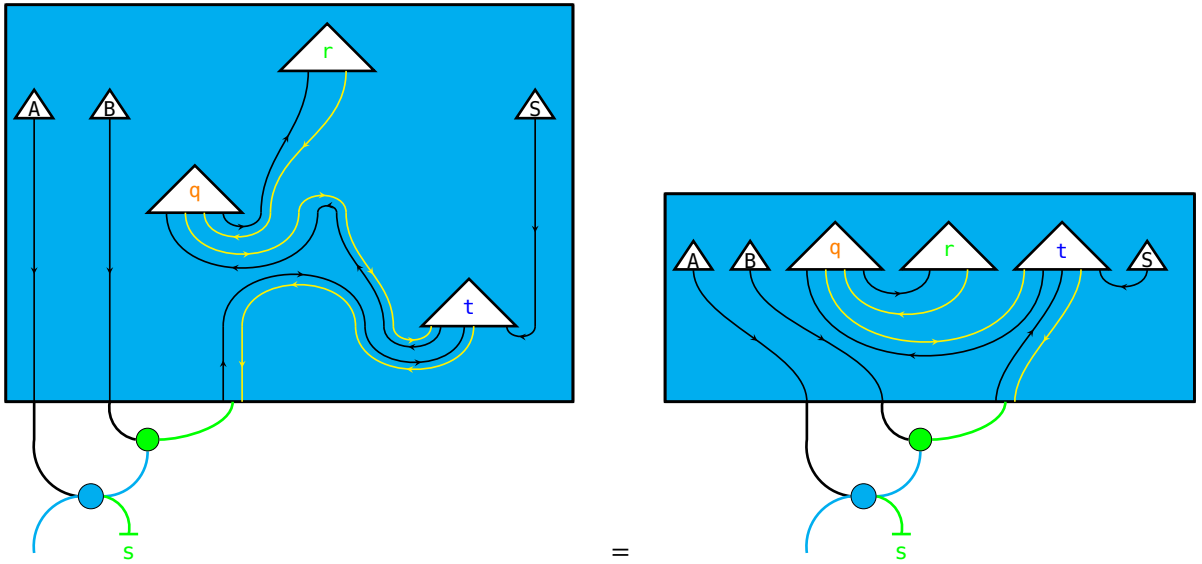
Similarly to `quickly`, we suppose we have a family of processes for the word `to`, one for each noun-arity of verb.



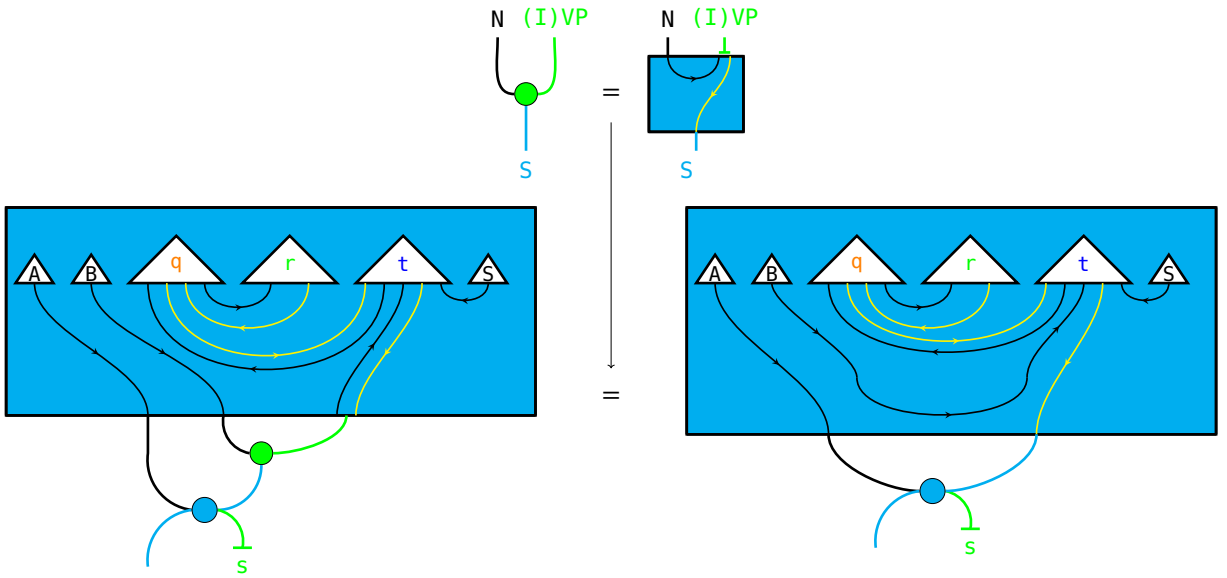
Again the discrete fibration introduces the appropriate choice of `to` when we pull the functor box down.



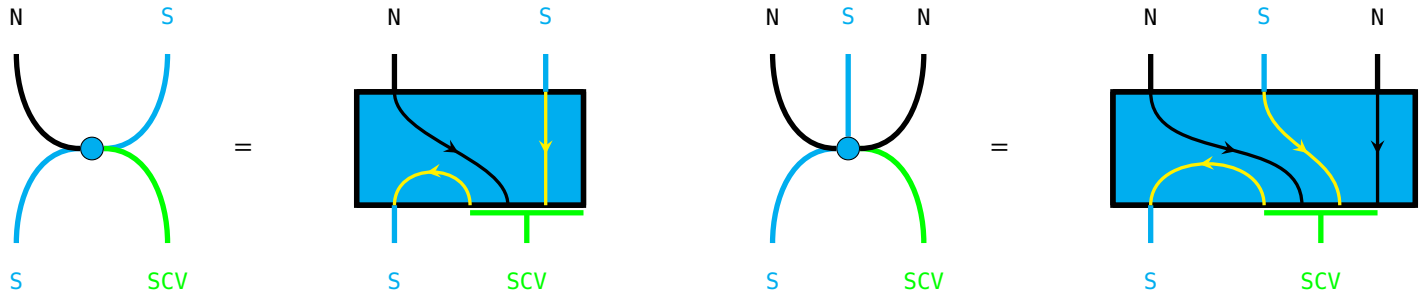
Now we visually simplify the inside of the functor-box by applying yanking equations.



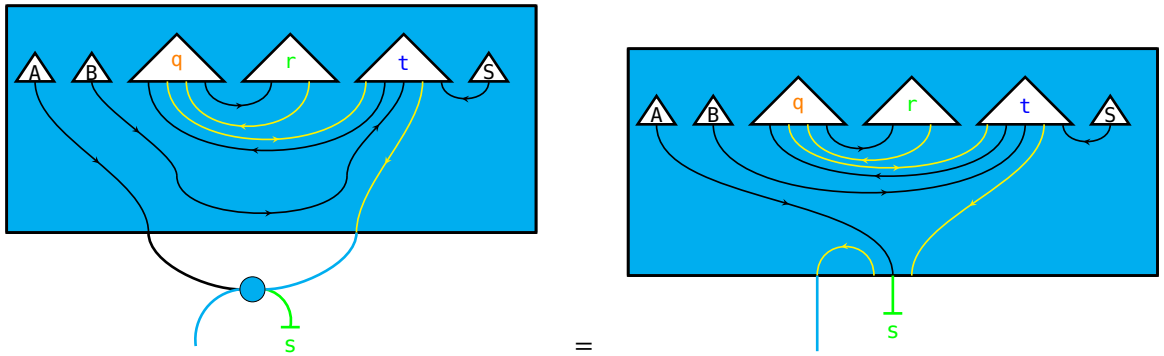
Similarly as before, we can pull the functor-box past the intransitive verb node. There is only one pregroup type ${}^{-1}n \cdot s$ that corresponds to the grammatical category `(I)VP`.



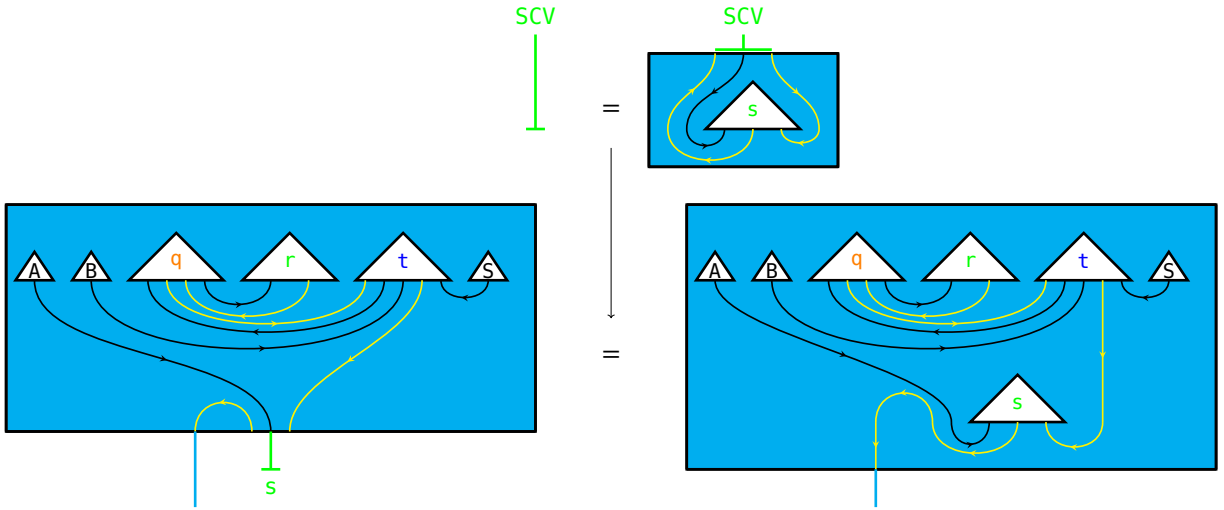
Proceeding similarly, we can pull the functor-box past the sentential-complement-verb node. There are multiple possible pregroup types for `SCV`, depending on how many noun-phrases are taken as arguments in addition to the sentence. For example, in `Alice sees [sentence]`, `sees` returns a sentence after 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:



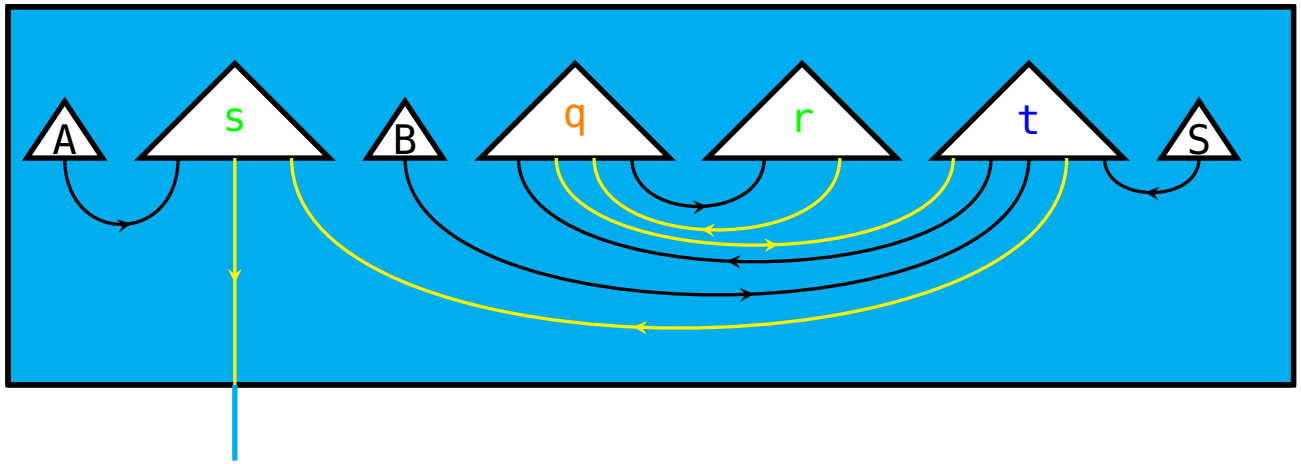
Pulling down the functor box:



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`.

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

Remark 3.1.21. Technical addendums.

The above construction only requires the source category of the functor to be rigid autonomous. Since no braidings are required, the free autonomous completion [Antonificaton] of any monoidal category may be used.

To enforce the well-definedness of the functor $\mathbf{F} : \mathcal{PG} \to \mathcal{G}$ on objects, we may consider the strictified "category of..." [ghicadiagrams]...

3.1.9 Discrete monoidal fibrations for grammatical functions

3.1.10 Discussion

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 3.1.22 (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 $\cdots \neg$`knows(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`

3.2 A hybrid grammar for text

WE NOW INTRODUCE OUR HYBRID GRAMMAR which is generative and aims to capture linguistic connectedness. We introduce the rules in the margin, saving the main body for worked examples. We develop the hybrid grammar in three main steps. First, we start with a context-sensitive grammar for simple sentences that only contain one verb, where the context-sensitivity is invoked for adpositions that modify verbs depending on whether they are transitive or intransitivity. Second, we introduce the notion of pronominal links, which identify recurring nouns, and pronouns with their referents. Further, we will introduce rules that allow us to fuse together simple sentences with recurring nouns via relative pronouns. Third, we introduce the notion of verbs that accept a sentential complement and phrase scope boundary. This expands our fragment to deal with compound sentences containing subphrases that are themselves sentences. Going forward, we refer to our hybrid grammar of this section as "hybrid grammar" or just "grammar" when there is no confusion.

OUR AIM IS TO BUILD A MINIMAL GRAMMAR THAT ALLOWS US TO GENERATE TEXT CIRCUITS FROM TEXT AND STATE CRISP MATHEMATICAL RESULTS. To this end, we use a Frankensteinesque hybrid of basic ideas from different formalisms: we use Chomsky's transformational phrase structure grammars, adjusted with features of Lambek's pregroups; pronominal links are inspired by discourse representation theory, and phrase boundaries are inspired by dependency grammars. Section **??** outlines these relationships in more detail. For the sake of clarity, we do not deal with some grammatical phenomena (like tense), omit certain grammatical patterns (we assume adverbs always come before the verb), and only deal with part of language (we do not consider determiners and quantifiers here, and we assume that ditransitive verbs have equivalent presentations as transitive verbs with adpositions.) This approach also has the usual shortcomings of formal grammars – such as not taking into account adjective order for purpose, origin, etc. in languages like English – that can be dealt with in the usual ways, requiring grammar to be mixed up with meanings. In fact, DisCoCat/DisCoCirc are all about combining grammar and meaning, and we (as many others) believe that ultimately they shouldn't exist independently, but should mutually inform each other. Practical NLP has empirically shown that this is essential for producing efficient tools. Pronominal link and phrase boundary data are not uniquely determined by text when given as just a string of words, without any further context – but then again, neither are grammatical types in many cases. The reason for including them is their necessity for obtaining *disambiguated* text structure which we can reason about mathematically.

WE MODEL PHRASE STRUCTURE USING A STRING-REWRITE SYSTEM. Phrase structure is what constrains the order of words in a sentence. String rewrite systems consist of (a usually finite collection of) *production rules* [1]:

$$\alpha \mapsto \beta$$

where α and β are strings of symbols. In our case, they may be phrase components, such as NP, or English words, such as BOB. We underline the latter in order to indicate that such symbols are *terminal* i.e. not rewriteable further by production rules.

ALL SUCH STRING-REWRITE SYSTEMS SPECIFY LANGUAGES. A language is viewed as a collection of strings of symbols

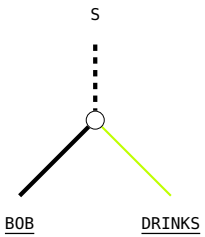


Figure 3.6: We will depict derivations of strings as planar "trees". The diagrams are read from top to bottom.

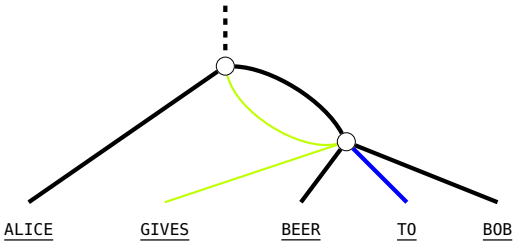


Figure 3.7: These "trees" may have multiple edges from a parent node to a child node. We drop symbolic labels for intermediate symbols, and replace them by coloured edges. For example, NP becomes a black edge and IVP becomes a green edge. As we introduce the rules, we will also keep to a coloring convention for typed wires, such that later on we may omit typings such as IVP from diagrams without confusion.

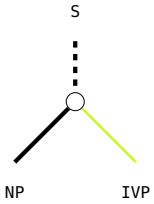


Figure 3.8: We now introduce a phrase structure grammar by giving the tree-fragments for the grammatical types, initially for what we call *simple* sentences, which have a single verb that does not take a sentential complement. A simple sentence may contain a single **intransitive** or **transitive** verb. In the former case, the sentence consists of a noun-phrase followed by a intransitive-verb-phrase (e.g. ALICE RUNS.). S $\mapsto$ NP $\cdot$ IVP

[1]

as follows. A special *start symbol* S is specified, and the language associated with the rewrite-system is the collection of all strings of terminal symbols that can be produced by (finite) applications of the production rules available, for example, given rules:

$$\mathtt{S} \mapsto \mathtt{NP} \cdot \mathtt{IVP} \tag{3.1}$$

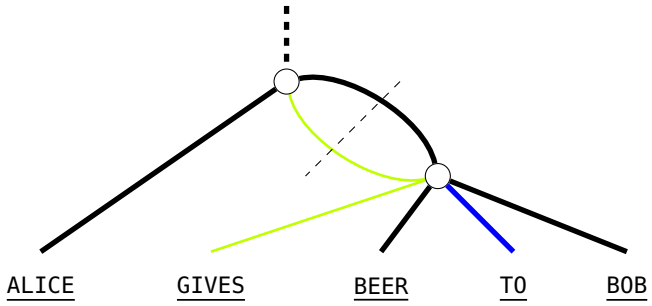
$$\mathtt{NP} \mapsto \underline{\mathtt{BOB}} \tag{3.2}$$

$$\mathtt{IVP} \mapsto \underline{\mathtt{DRINKS}} \tag{3.3}$$

where $\cdot$ is notation for string concatenation, we can produce simple sentences such as:

$$\begin{aligned} \mathtt{S} &\overset{(3.1)}{\mapsto} \mathtt{NP} \cdot \mathtt{IVP} \\ &\overset{(3.2)}{\mapsto} \underline{\mathtt{BOB}} \cdot \mathtt{IVP} \\ &\overset{(3.3)}{\mapsto} \underline{\mathtt{BOB}} \cdot \underline{\mathtt{DRINKS}} \end{aligned}$$

Example 3.2.1. First applying the TVP-production rule we obtain NP · TVP · NP, which corresponds to the grammatical structure of ALICE GIVES BEER. Then applying the TVP · NP-production rule, we transform GIVES BEER (i.e. both a green and a black wire) into GIVES BEER TO BOB. Altogether, we obtain the following "tree", where the dashed line indicates the two "subtrees":



WE CALL A SENTENCE WITH MORE THAN ONE VERB *compound*. We consider two ways in which compound sentences arise: relative pronouns, and phrase scope. We focus now on the first case. Relative pronouns fuse simple sentences together. We model the data of a pronoun using a **pronominal link**, which identifies nouns from possibly different sentences. We depict these as arrows under the trees, pointing at identified nouns.

Example 3.2.2. For example, in the text: ALICE IS SOBER. ALICE GIVES BEER TO BOB., we can identify the two pronominally-

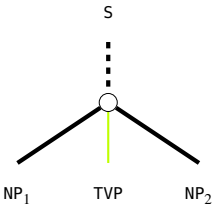
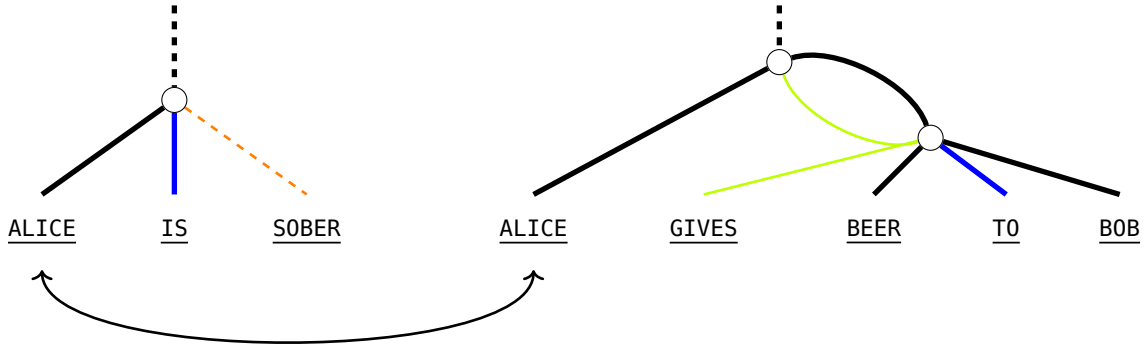


Figure 3.9: In the latter case, a sentence consists of a noun-phrase, transitive-verb-phrase, and another noun-phrase (e.g. ALICE LIKES BOB.). $\mathtt{S} \mapsto \mathtt{NP}_1 \cdot \mathtt{TVP} \cdot \mathtt{NP}_2$

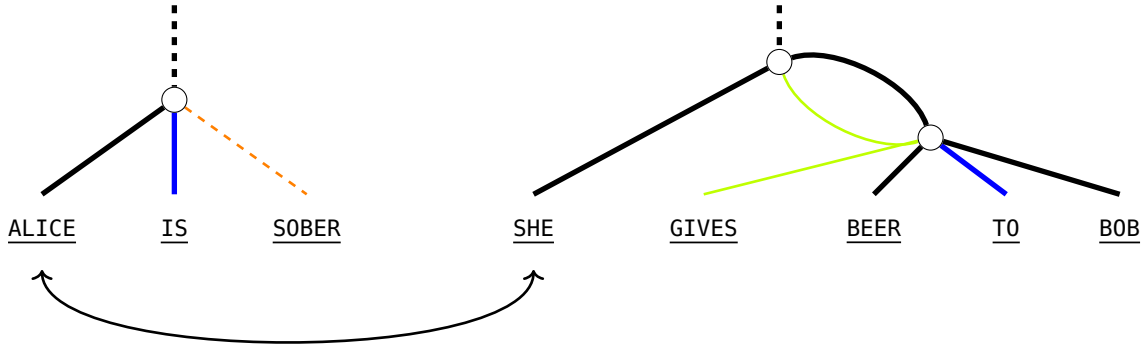
Figure 3.10: There also are the terminal rules for verbs, where the terminal symbols of the grammar are verbs of the appropriate type e.g. intransitive: $\mathtt{IVP} \mapsto \underline{\mathtt{IV}}$

Figure 3.11: Or transitive: $\mathtt{TVP} \mapsto \underline{\mathtt{TV}}$. Going forward, we omit the terminal rules in favour of giving examples of finished derivations, from which terminals can be inferred.

linked occurrences of ALICE.



In the presence of a pronominal link, later occurrences of a noun are interchangeable with a pronoun that refers to an earlier occurrence. We informally outline a convention here, stated in more detail later as Convention **??**. We assume that pronominal pointers in text always point from nouns in later text to nouns in earlier text. i.e. from right to left. We also assume that every noun has at most one outgoing pronominal pointer, and at most one incoming pronominal pointer.



RELATIVE PRONOUNS MAY BE DECONSTRUCTED IN TERMS OF PRONOMINAL LINKS BETWEEN SIMPLE SENTENCES. One way compound sentences emerge is by the use of relative pronouns, which allow a noun to be modified by more than one verb.

where the big triangle on the right hand side is a visual convention to indicate that the two sentences are 'fused' as a single sentence. A rewrite rule like (**??**) is called a *transformational grammar rule*. Transformations modify phrase structure trees. Below we encounter more examples of transformational rules for relative pronouns. In each example we will start with pronominally linked simple sentences to obtain a meaning-equivalent compound sentence with a relative pronoun. The important takeaway is that we can without loss of generality forget about relative pronouns by always considering their sense-equivalent in terms of pronominal links between simple sentences.

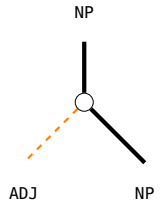


Figure 3.12: **Adjectives** can appear before a noun-phrase (e.g. DRUNK HAPPY BOB.) NP ↦ ADJ · NP.

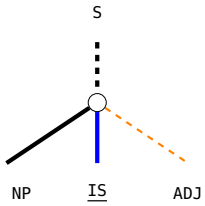


Figure 3.13: Or, using the copular IS considered as a verb, a single adjective can appear after a noun-phrase (e.g. BOB IS DRUNK.) S ↦ NP · IS · ADJ

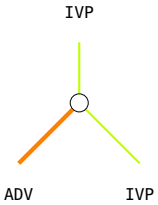


Figure 3.14: **Adverbs** can appear before a verb (e.g. ALICE QUICKLY HAPPILY RUNS.) IVP ↦ ADV · IVP

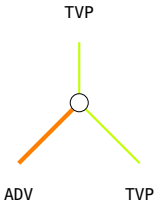
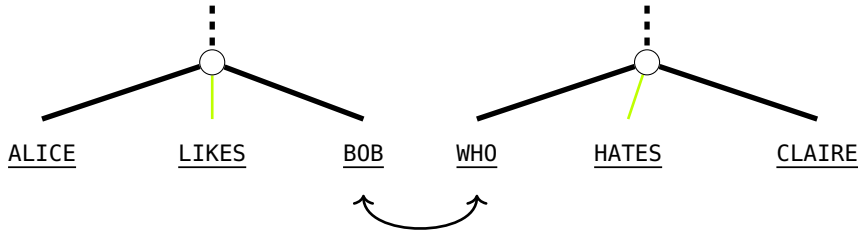


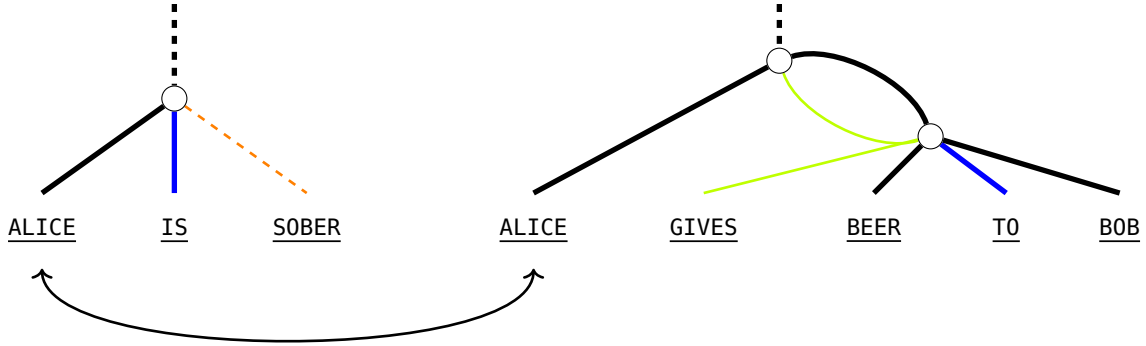
Figure 3.15: TVP ↦ ADV · TVP

SUBJECT RELATIVE PRONOUNS.

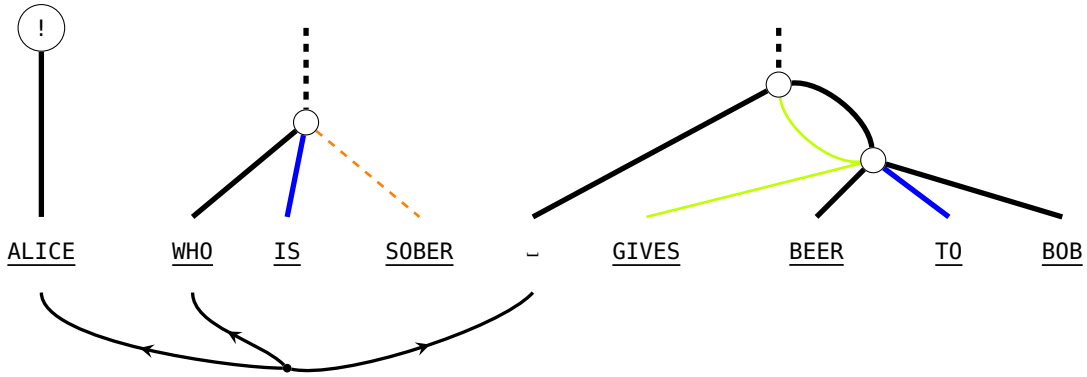
Example 3.2.3. For the text ALICE LIKES BOB. BOB HATES CLAIRE. we start with two trees. We identify the occurrences of BOB with a pronominal link. We can now fuse the trees by replacing the second occurrence of BOB with the relative pronoun WHO, yielding:



Example 3.2.4. We revisit a previous example:



By applying the special rule for subject relative pronouns, we obtain the following sentence:



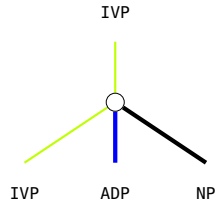


Figure 3.16: **Adpositions** can appear to the right of an intransitive-verb-phrase, followed by a noun-phrase (e.g. from ALICE RUNS. to ALICE RUNS TOWARDS BOB.). IVP ↦ IVP · ADP · NP.

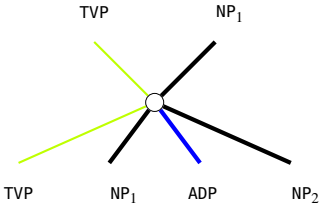


Figure 3.17: In the presence of a transitive-verb-phrase flanked by a noun-phrase to the right, we may add to the right an adposition followed by a noun-phrase (e.g. from ALICE THROWS BEER. to ALICE THROWS BEER TOWARDS BOB.) $\mathtt{TVP} \cdot \mathtt{NP}_1 \mapsto \mathtt{TVP} \cdot \mathtt{NP}_1 \cdot \mathtt{ADP} \cdot \mathtt{NP}_2$

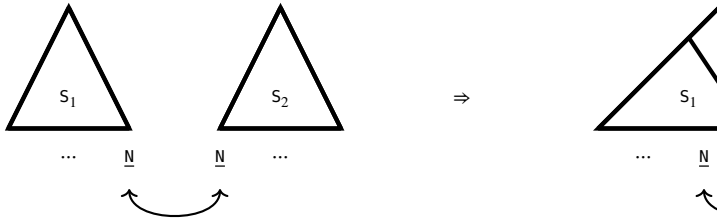
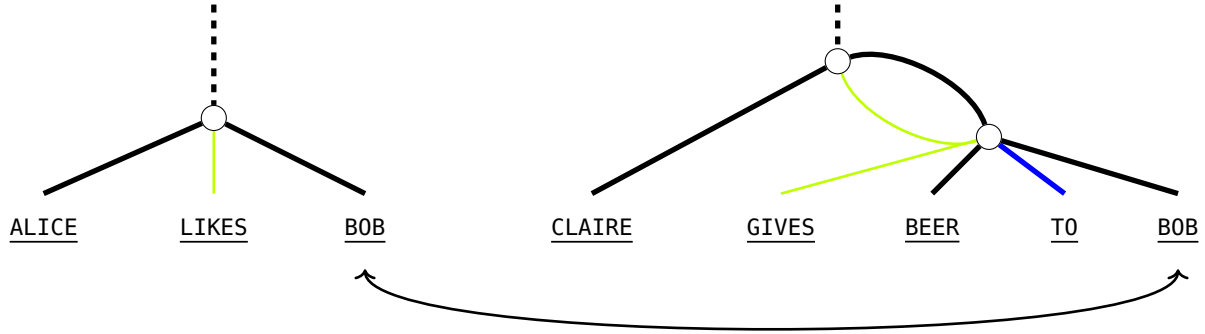


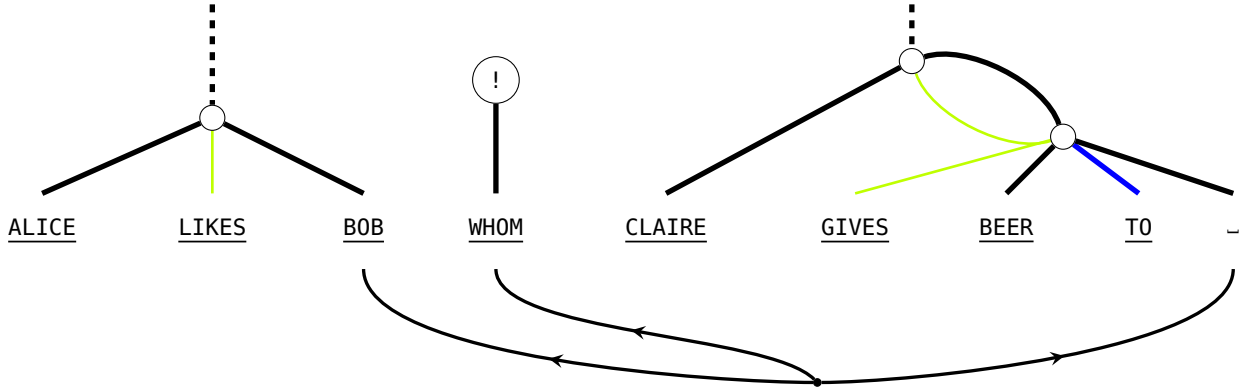
Figure 3.18: **Subject relative pronouns** replace the subject noun of a parse tree $\mathtt{S}_2$, and points to a noun in another, previous parse tree $\mathtt{S}_1$, usually the object noun.

We have not encountered an isolated noun – here indicated by ! – or blank labels _ before. Observe that these artefacts disappear when we treat relative pronouns as pronominally linked simple sentences.

Example 3.2.5. We start with the text ALICE LIKES BOB. CLAIRE GIVES BEER TO BOB.



Applying the rule for object relative pronouns, we obtain:



This concludes our tour of the first case of compound sentences. Now to the second case.

PHRASE SCOPE OCCURS WHEN A SUBPHRASE OF A SENTENCE IS ITSELF A FULL SENTENCE. We introduce two such cases. We first introduce these cases intuitively, then we introduce the refinement of *phrase scope* to eliminate a source of ambiguity.

Example 3.2.6. A sentence may consist of a noun-phrase, followed by a verb with sentential complement, followed by a sen-

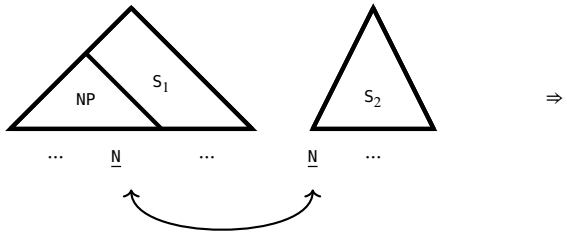


Figure 3.19: In English there is another special use of the subject relative pronoun to coordinate a single noun across two subsequent phrases. Given parse trees corresponding to sentences S_1 and S_2 in that order, this special case arises when the subject noun of S_2 points towards a subject noun in S_1. The result of the tree-transformation is that we have, in order: the noun-phrase (along with any adjectives) of S_1, followed by S_1 with the later noun replaced by the relative pronoun, followed by S_2 with its pointing noun removed. We use a multiarrow to point out more than two pronominally identified nouns.

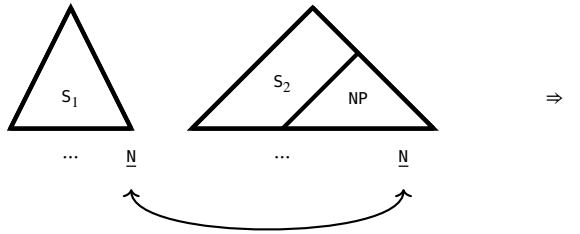
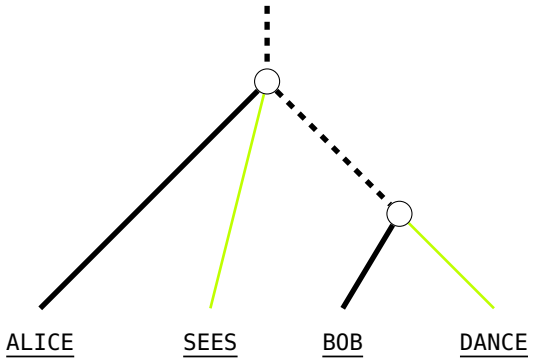
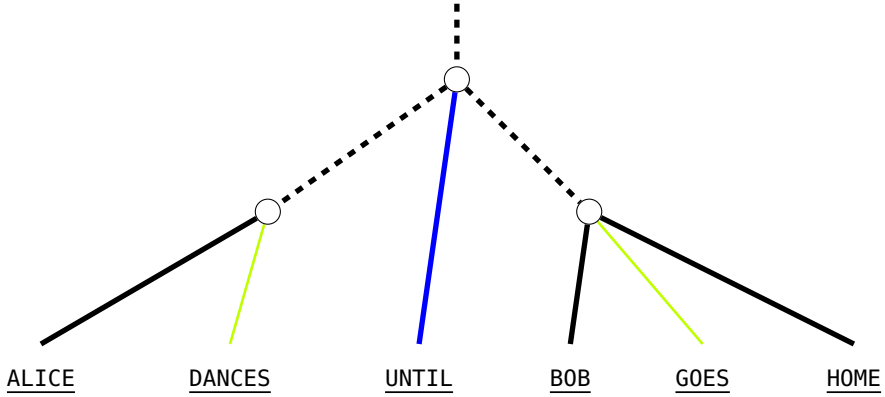


Figure 3.20: For **object relative pronouns**, we take consecutive sentences S_1 and S_2. If the object noun of S_2 points to the object noun of S_1, the object relative pronoun comes after the first occurrence, and the second occurrence of the noun is replaced by a blank.

tence (e.g. from BOB DANCES. to ALICE SEES BOB DANCE.).



Example 3.2.7. From BOB DANCES. and ALICE LAUGHS. to BOB DANCES SO ALICE LAUGHS.:



There is still a source of ambiguity to be addressed in the above formulation, which is where phrase scope comes into play.

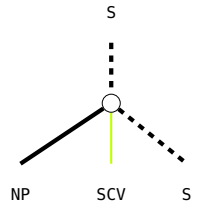


Figure 3.21: **Verbs with Sentential Complement** require phrase scope. We treat verbs with a sentential complement – such as to SEE or THINK – as their own grammatical class of verb.S ↦ NP · SCV · S

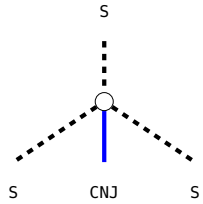
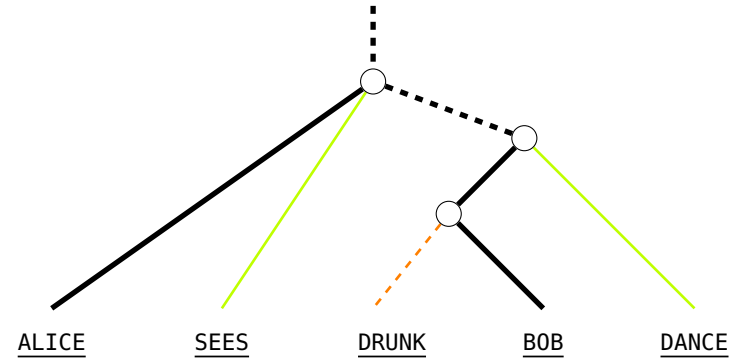


Figure 3.22: **Conjunctions** we treat similarly to verbs with a sentential complement
except with two phrase-bounded regions rather than one, on each side of a conjunction. S ↦ S · CNJ · S

For the sentence ALICE SEES DRUNK BOB DANCE. there is only one phrase structure:



which places all of DRUNK BOB DANCE(S) under the scope of what ALICE SEES. However, we wish to further distinguish between the following cases, where:

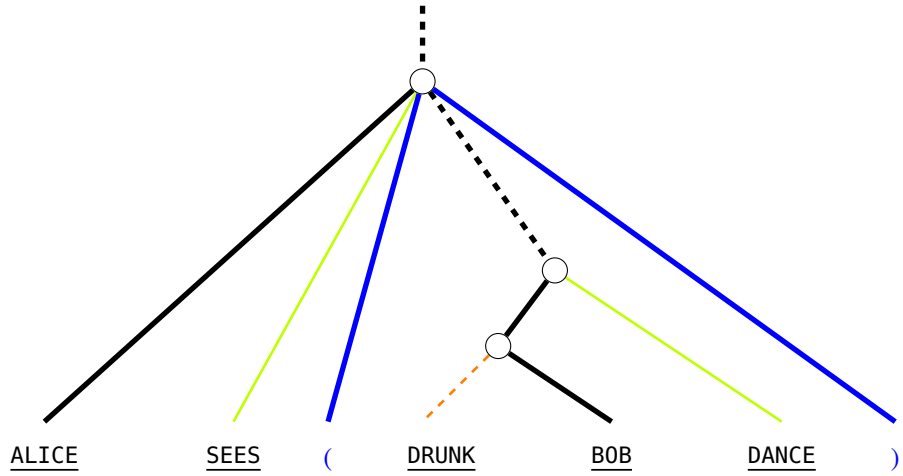
- Alice sees **both** that Bob is drunk and that Bob dances.
- Alice sees **only** that Bob dances, but not that Bob is drunk.

To make the distinction between these two cases we introduce two 'formal' types (and), which represent the left and right boundaries of phrase scope respectively.

Example 3.2.8. To disambiguate that in the sentence

ALICE SEES DRUNK BOB DANCE.

Alice sees **both** that Bob is drunk and that Bob dances, we have the following phrase structure:



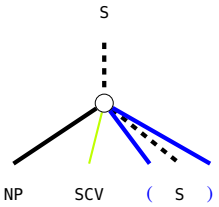


Figure 3.23: We re-express the rules for verbs with sentential complement and conjunctions to additionally express phrase boundaries. S ↦ NP · SCV · (· S ·)

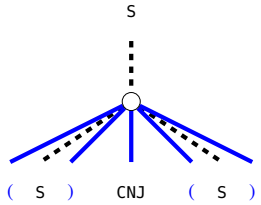


Figure 3.24: S ↦ (· S ·) · CNJ · (· S ·)

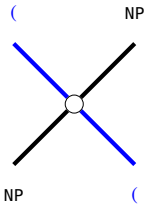


Figure 3.25: To model the permission of nouns to partially live outside the scope of a phrase as in the example above, we include the following rules that allow a NP to 'cross the border' of a phrase boundary. (· NP ↦ NP · (

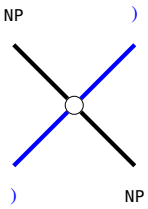


Figure 3.26: NP ·) ↦) · NP

To disambiguate that Alice **only** sees that Bob dances, and not that he is drunk, we have:

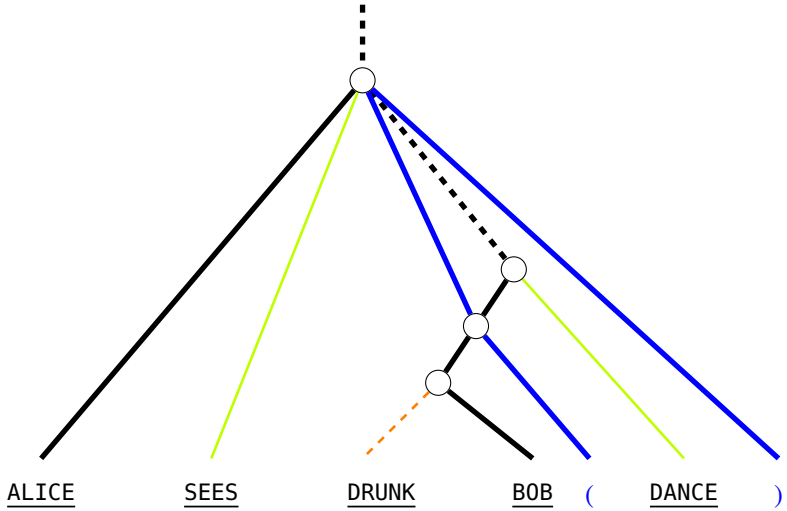
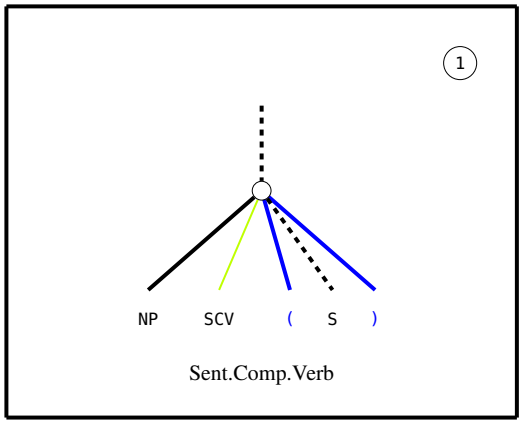
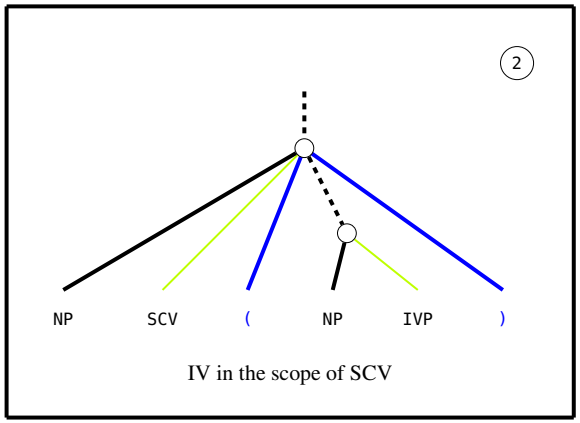
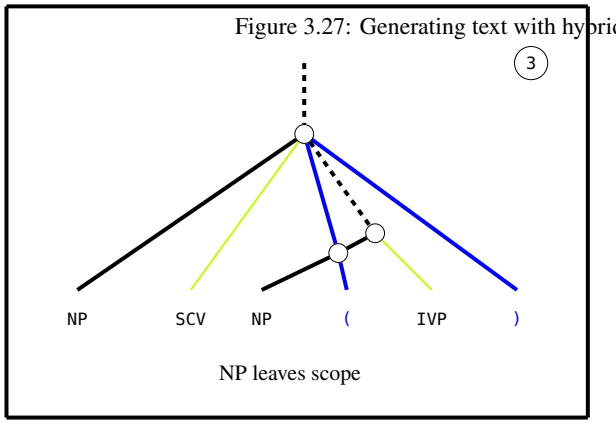
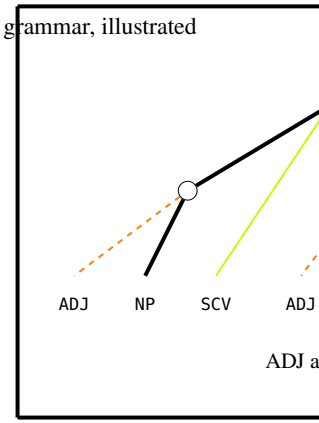


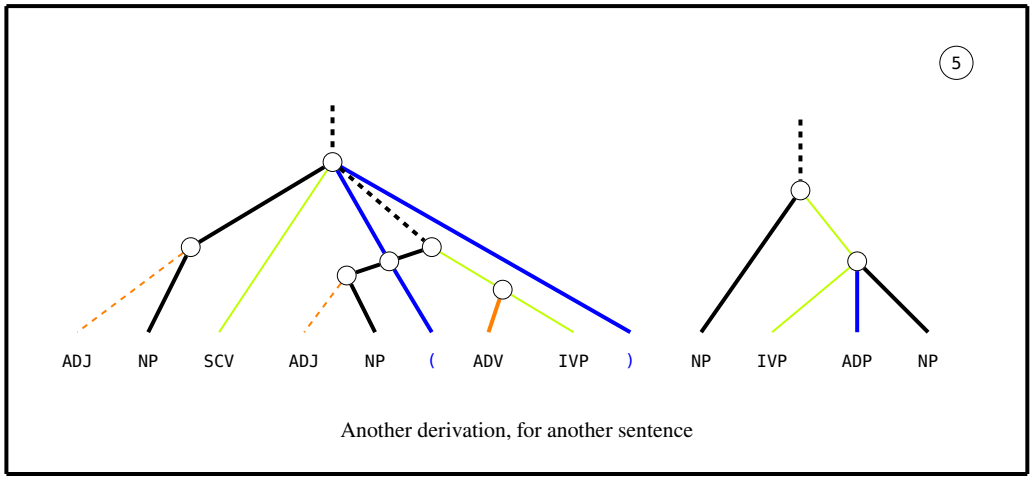
Figure 3.27: Generating text with hybrid grammar, illustrated

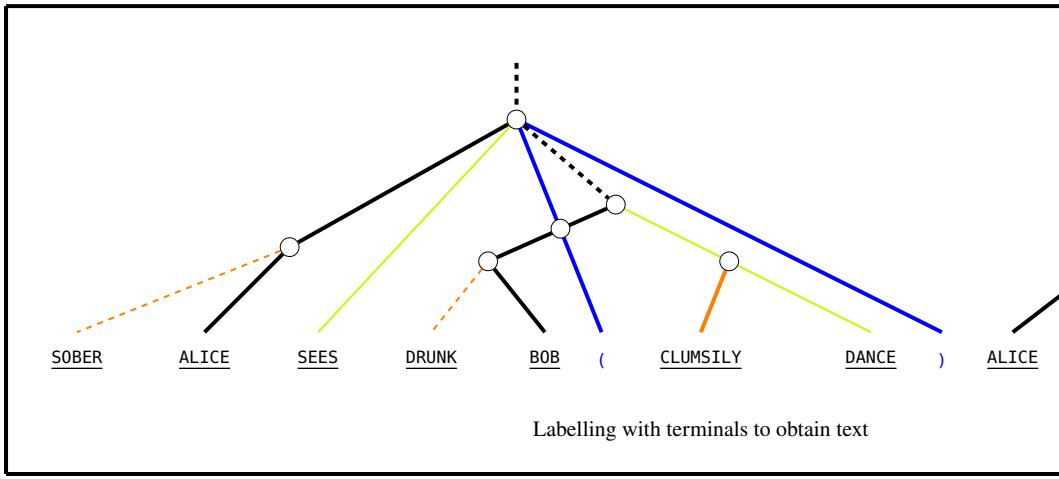


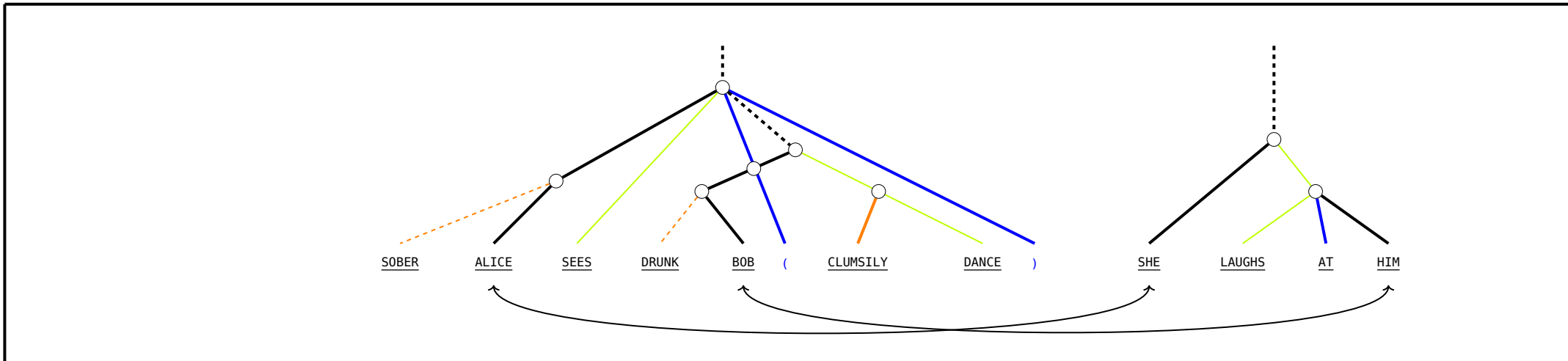










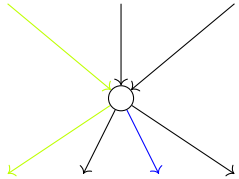


3.3 Text diagrams

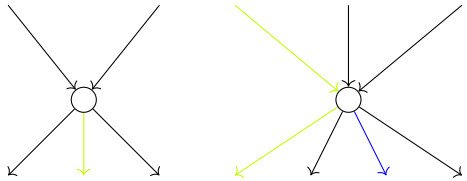
We now introduce *text diagrams*, which incorporate the grammatical phenomena of text with hybrid grammar (cf. pronominal links and phrase boundaries) into a single mathematical structure. We will do so by means of a systematic passage from text with hybrid grammar to text diagrams. Artefacts required to handle the behaviour of relative pronouns in hybrid grammar vanish in the setting of text diagrams.

3.3.1 Preliminaries

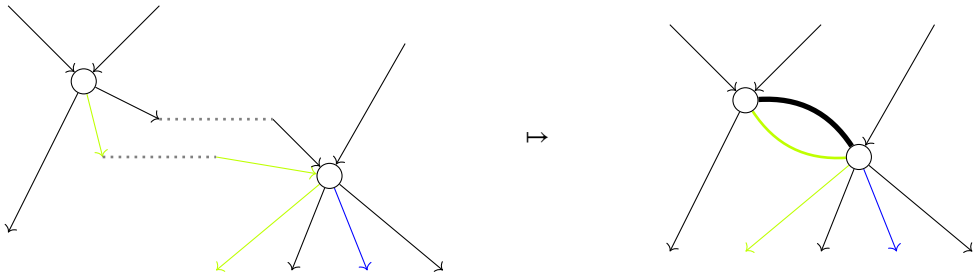
String diagrams [2] are a graphical mathematical framework for composing input-output boxes. For us the 'boxes' will have the following shape:



For composition one typically one distinguishes between parallel composition:



or sequential composition by plugging together matching outputs and inputs.



Despite the 1-dimensionality connoted by 'string', string diagrams in general also accommodate higher-dimensional structures such as planes and volumes [3].

[2] ; ; ; and

[3]

3.3.2 Simple sentences as text diagrams

The initial symbol for our phrase structure rules in the previous section was S. In the passage from hybrid grammar to text diagrams, systematically:

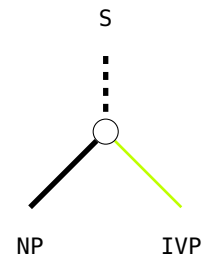
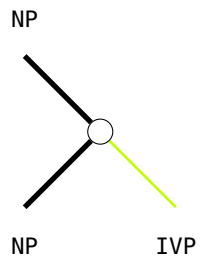
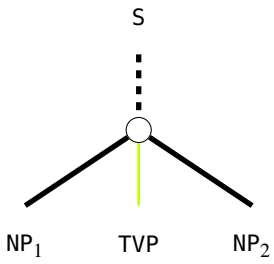
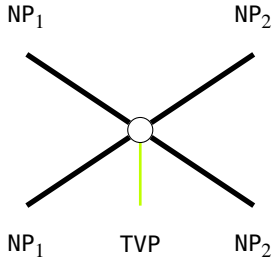
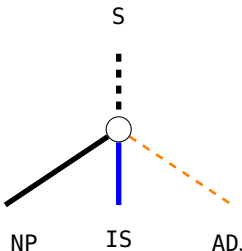
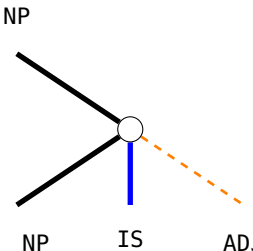
- We will replace the sentence type S with a sentence-dependent number of NP wires.

In this way, tree-shaped texts with hybrid grammar become string diagrams with one input NP wire for every pronominally distinct NP label in the text. This geometric change of perspective is what enables a passage to text circuits. To achieve this, we alter the rules of Section 3.2 as follows:

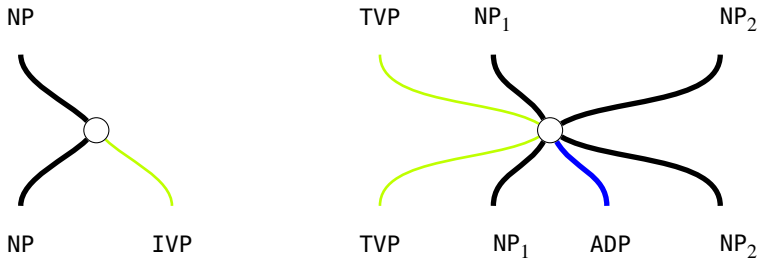
- We remove S types.

- Every rule must preserve the number of input and output noun wires, so phrase structure rules that introduce NP types must be altered to also take the same number of NP types as input.

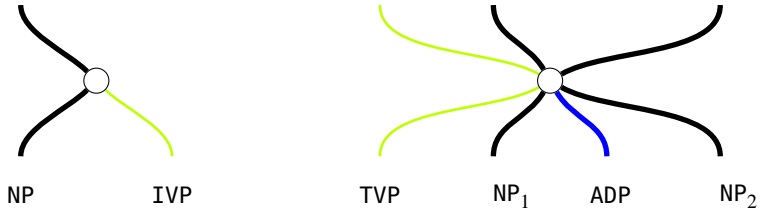
In the table below where we depict just those rules that are altered in this way. As can be seen, besides removing sentence-inputs and adding noun-inputs, subscripting NP where there are multiple, we also vertically align matching input-output pairs:

Grammar	Rule	Diagram
	Intrans.Verb	
	Trans.Verb	
		

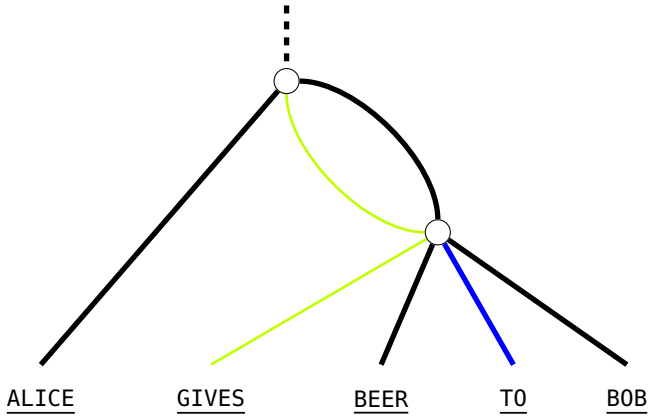
Similarly, when we draw text diagrams, as a visual convention we consistently arrange the diagram so that matching input and output NP wires align, and bend those a bit always pointing either upward or downward, to better reflect the notion that wires should be considered as inputs and outputs in the string diagram sense:



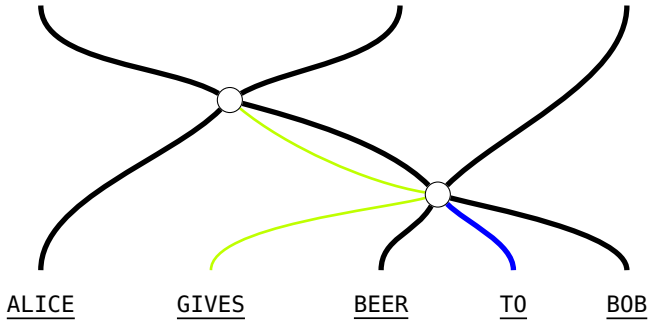
We can now also drop one of the doubly-appearing labels:



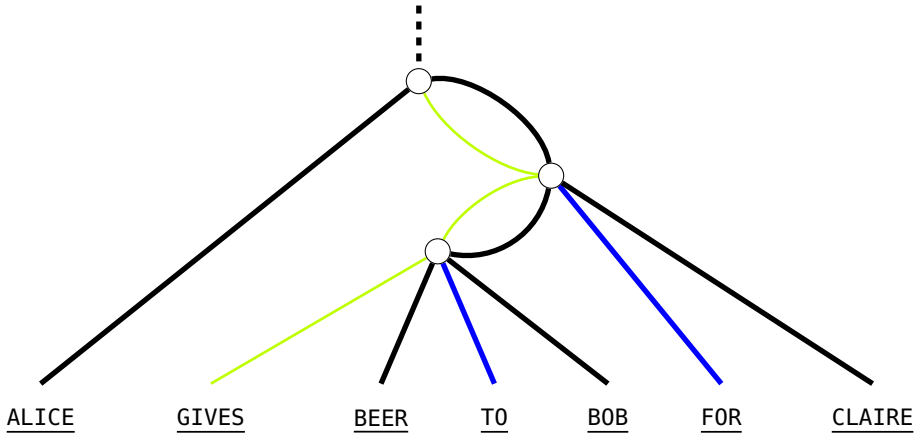
Example 3.3.1. For the simple sentence ALICE GIVES BEER TO BOB, the grammar is:



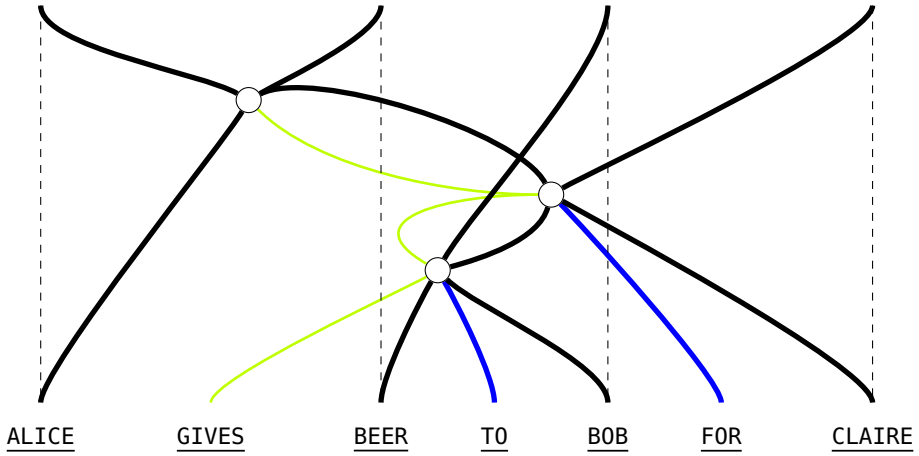
from which we obtain the following text diagram:



Example 3.3.2. We allow the new input NP wires obtained from grammar to diagram translation to cross other wires to reach the top of the diagram. For eample, in the sentence ALICE GIVES BEER TO BOB FOR CLAIRE the grammar is:



from which we obtain the following text diagram, where noun wires are vertically aligned:

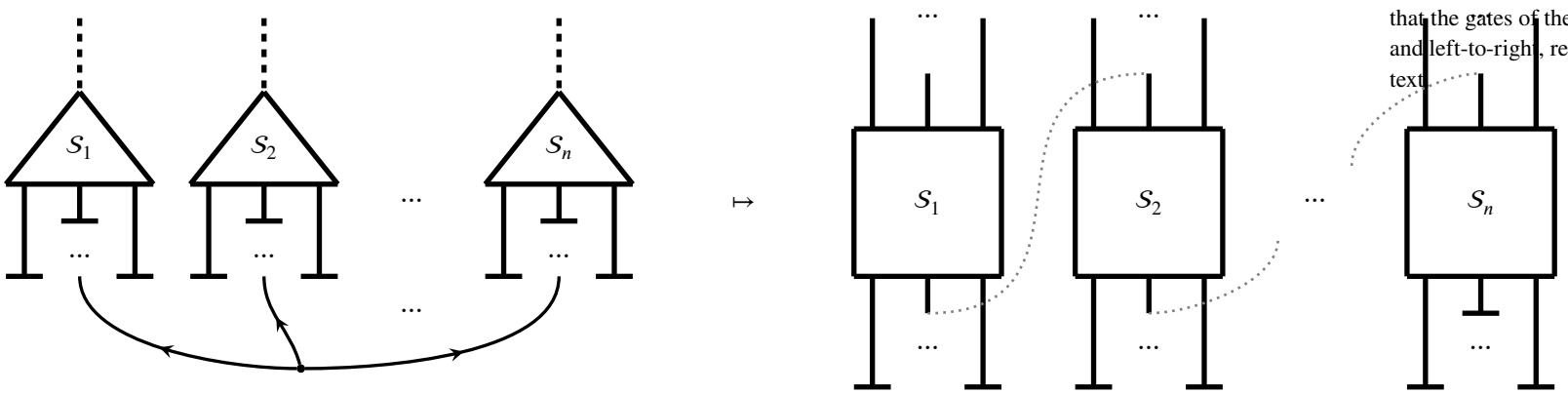


3.3.3 Rewriting text diagrams

When dealing with relative pronouns we have seen something analogous to Chomskian phrase-structure transformations: operations that fuse and more generally transform text diagrams. We formalise these as diagram rewrites, each of which replaces a diagram with another that has the same input and output wires as the first. We illustrate examples of such rewrites in the sections to follow.

3.3.4 Pronominal links in text diagrams

Pronominal links take a different shape in text diagrams than in grammar. The pronominal link arrows of **??** become wires that link output and input noun wires in a chain[4]:

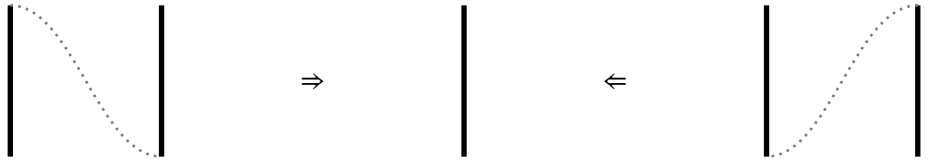


[4] We could have also had the pronominal links in the chain go from northwest to southeast. We choose this direction such that the gates of the resulting text circuit, read top-to-bottom and left-to-right, reflect the order in which words appear in text.

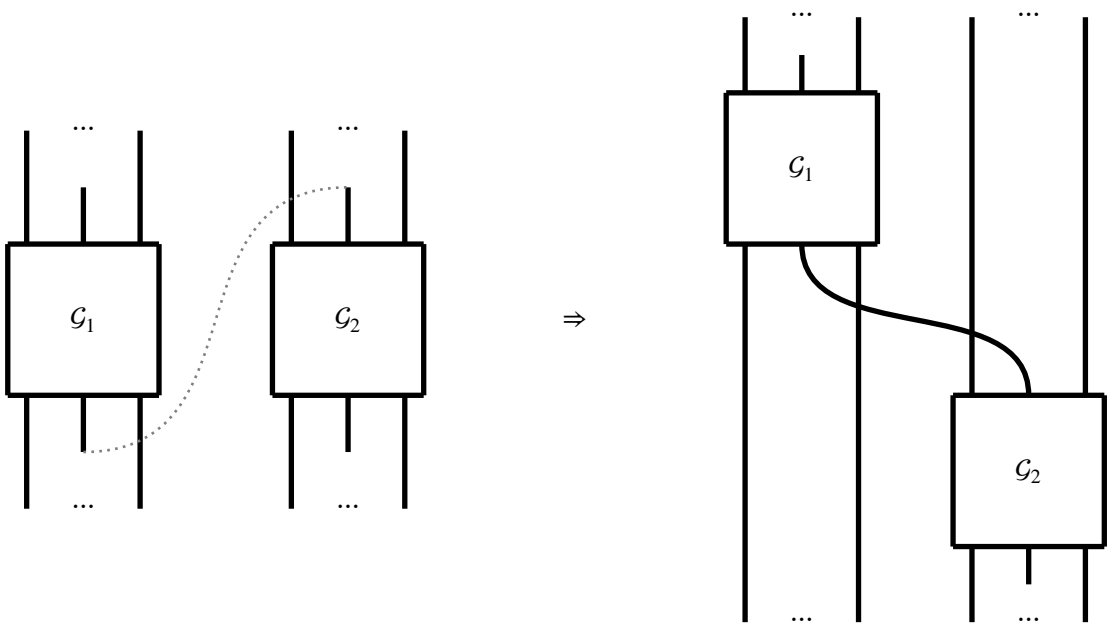
where the following pieces relate pronominal link wires to noun wires:

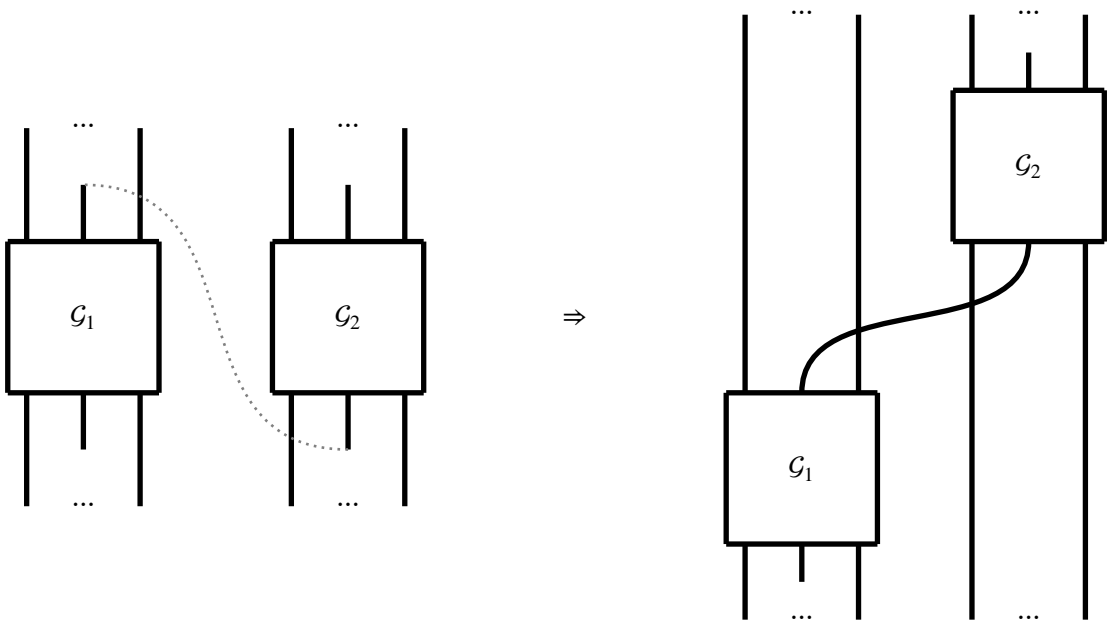
‘forward link’ pair ‘backward link’ pair

These pieces can vanish by the following link-elimination rewrites:



In particular, this means that where $\mathcal{D}_1$ and $\mathcal{D}_2$ are text diagrams, we have:





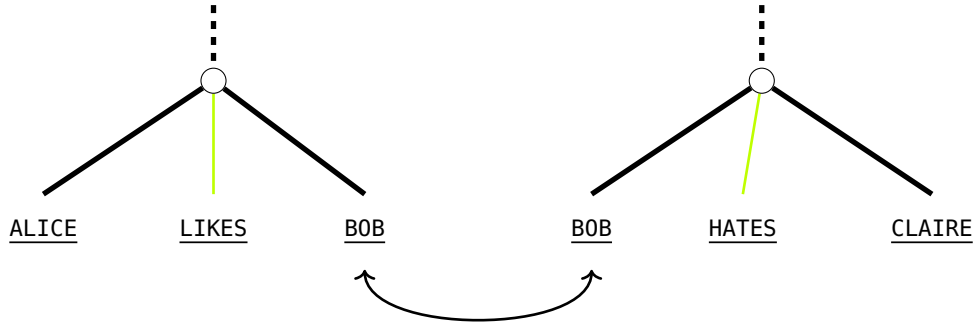
As we have seen now, wires may cross one another: this is an important quality of text diagrams.

Convention 3.3.3 (In text diagrams, only connectivity matters)**.** While hybrid grammar diagrams are planar, in text diagrams wires and labels may be freely induced to cross over one another, so long as input-output connectivity is preserved.

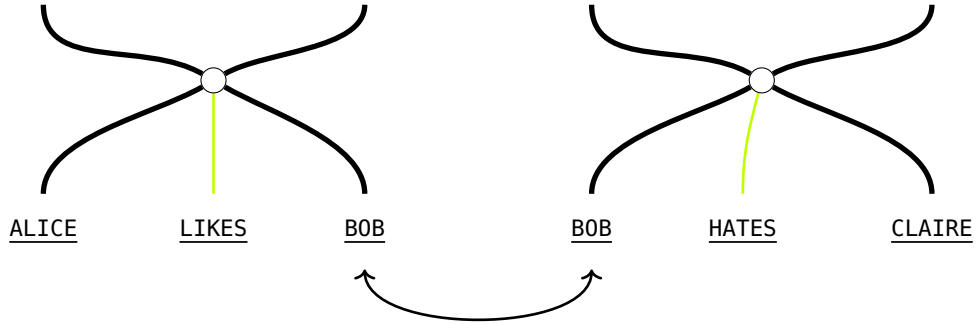
Link-elimination rewrites may induce twists in noun wires; a consequence of an overarching principle that only output-to-input connectivity matters. So we also have the following link-elimination rewrites:

We demonstrate by means of examples how these rewrites interpret pronominal links as composition of text diagrams.

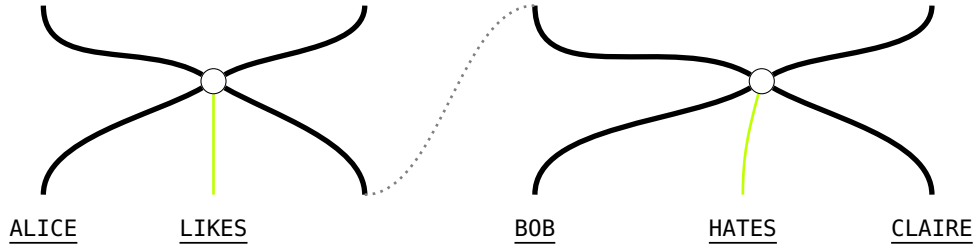
Example 3.3.4 (Subject relative pronouns revisited)**.** We start with the following text:



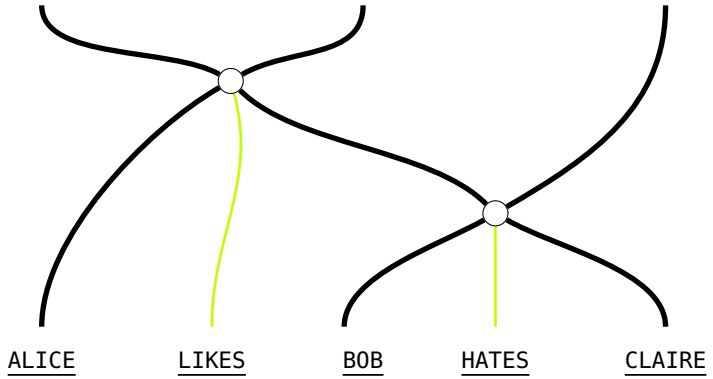
For which we can replace each tree with a text diagram:



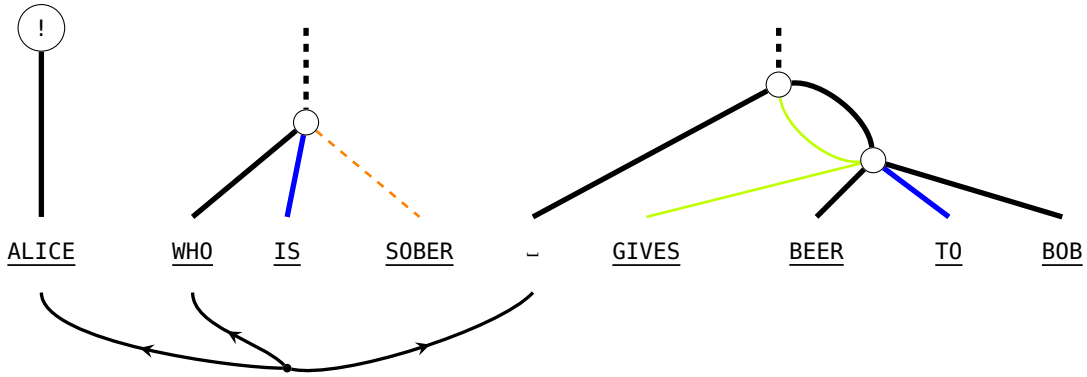
Now we can replace the pronominal link by its text diagram counterpart. Now `BOB` (which is a proper noun, rather than the pronoun `WHO`) labels the only available output wire:



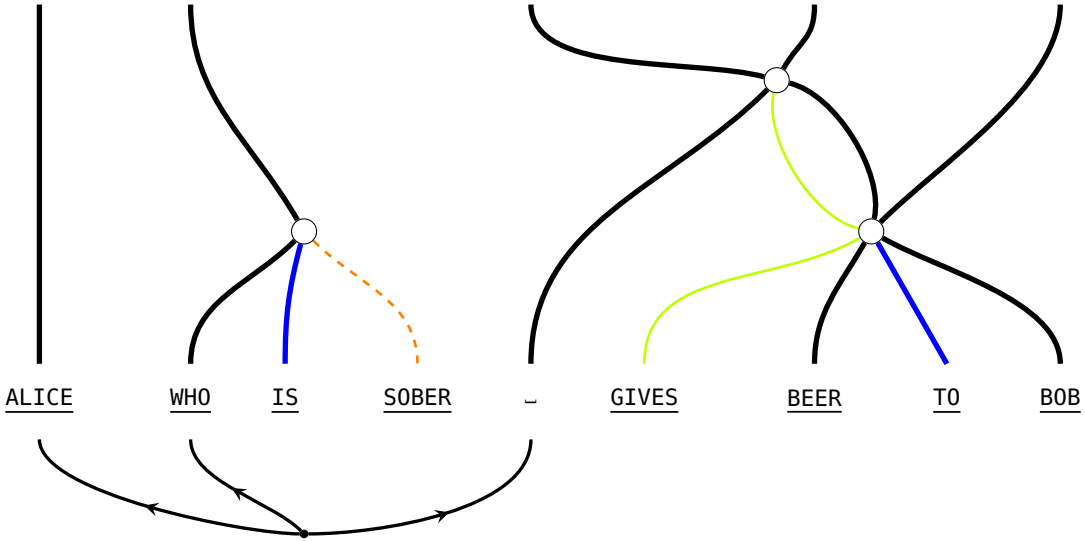
The pronominal link can now be rewritten away:



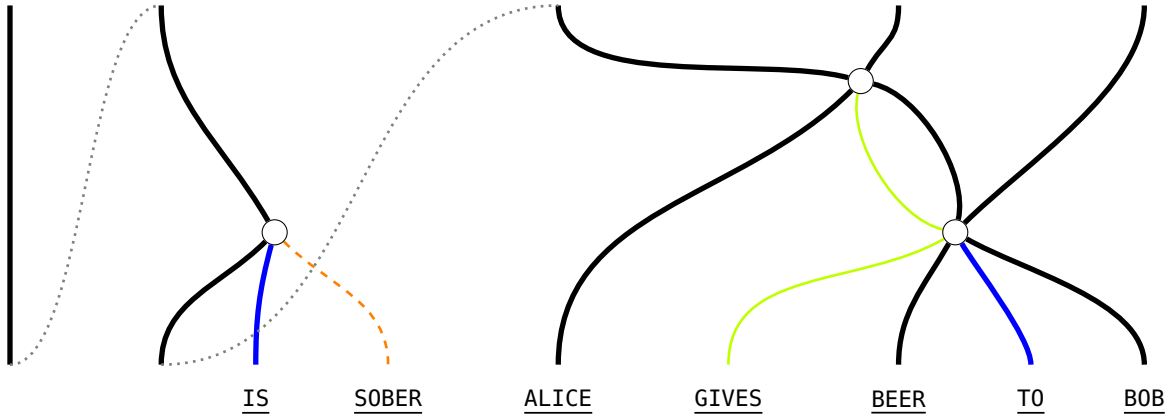
Example 3.3.5 (Subject relative pronouns revisited)**.** Pronominal links in the setting of text diagrams 'explain' the artefacts that were necessary in hybrid grammar to accommodate relative pronouns. Recall that for the special rule for subject relative pronouns, we obtain a structure with ! and _ artefacts:



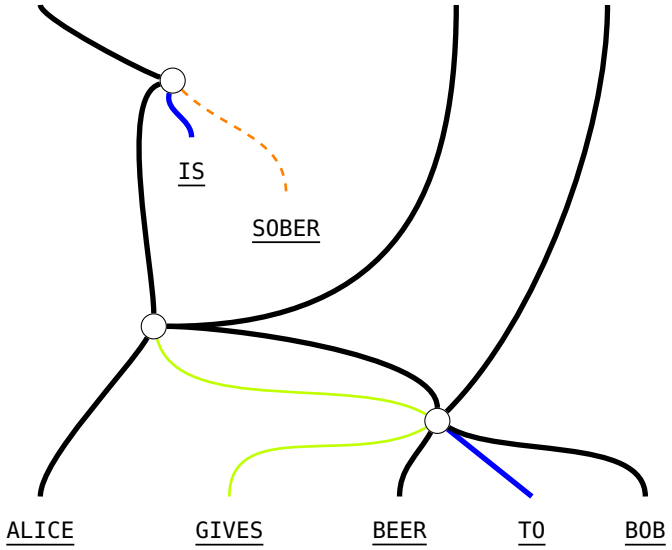
When we move to text diagrams, the ! artefact disappears:



Now we replace the pronominal links with their text diagram counterparts, obtaining:

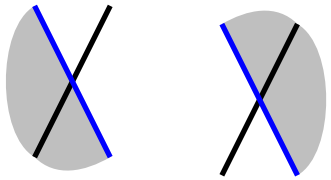


Rewriting away pronominal links, we have the following text diagram:

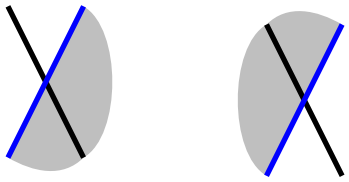


3.3.5 Phrase scope as phrase bubbles.

We will have to deal with compound sentences differently, because in the passage from text with grammar to text diagrams, we have replaced the sentence type S by a sentence-dependent collection of noun wires. This requires the use of string diagrams with regions.[5] We will introduce generators of text diagrams with regions that identify the boundaries that delineate a sentential subphrase. For verbs with a sentential complement that take a phrase to the right, we have diagram pieces that allow a noun-wire to enter a bounded phrase from the right, and exit on the left:

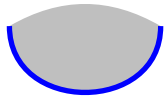


For conjunctions, which also take a phrase to the left, we have rewrites that allow a noun-wire to enter a bounded phrase from the left, and exit on the right:



[5] We forgo exposition here, but everything we present here can be expressed in the setting of associative *n*-categories , and in fact prototyped and implemented in a proof assistant . ; and

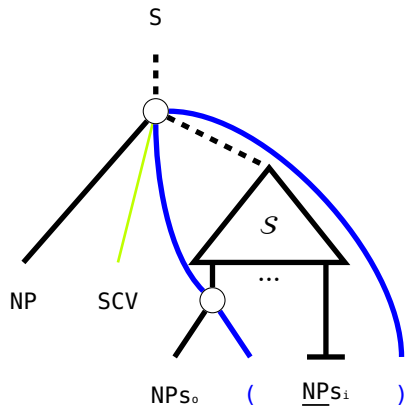
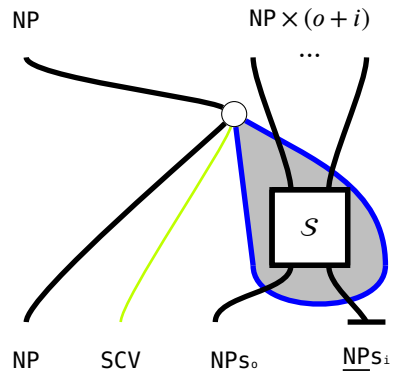
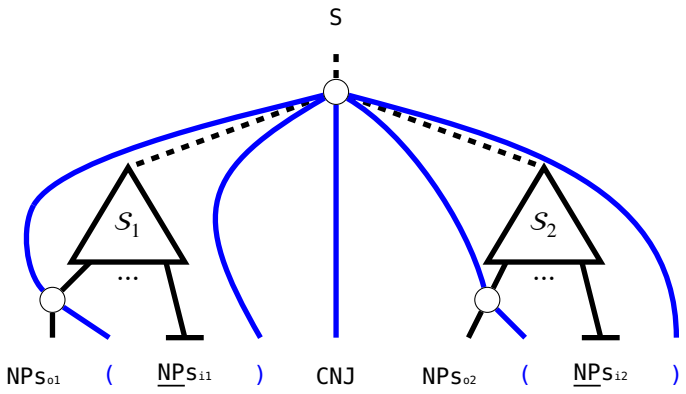
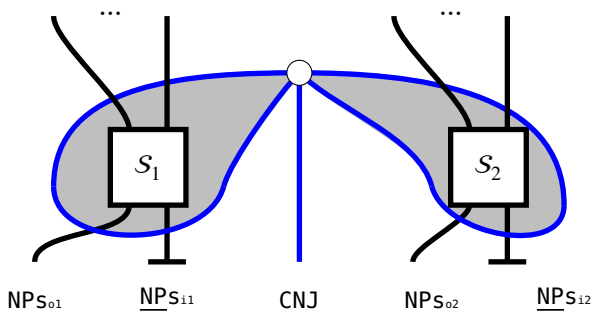
In both cases, we have one more rewrite to ‘cap off’ phrase regions. Formally we must distinguish between phrases-to-the-left and -to-the-right, which have differing caps, but in the graphical presentation we elide this distinction as it is visually evident:



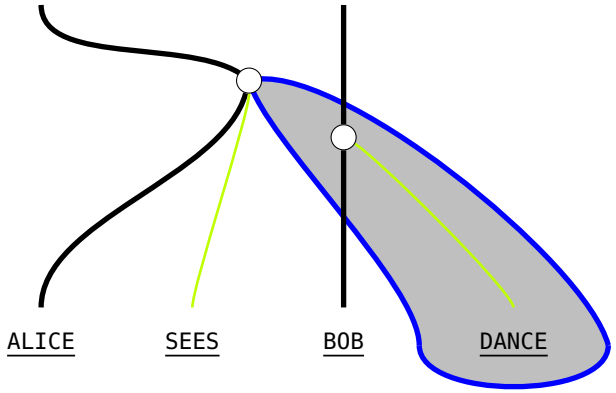
Convention 3.3.6 (Rules for phrase scope)**.** We state graphical conventions for phrase scope and phrase regions here. First, phrase scoped regions behave as planar obstacles in a text diagram, except for NP wires and pronominal links. This has two consequences: that nodes and wires occurring within a phrase scope are ‘trapped inside’, and also that multiple phrase scope constructions may nest iteratively, but no two overlap. Second, NP labels may only occur outside phrase scope.

For phrase scope constructions in grammar – verbs with sentential complements and conjunctions – in order to ensure that the resulting text diagram has an equal number of input and output noun wires, our correspondence must take into account the number of labelled noun wires (drawn with a ‘foot’) in the subsentence within the phrase scope. We subscript the noun labels with indices *i* and *o*, for ‘inside scope’ and ‘outside scope’. We have diagram counterparts for verbs with sentential

complements and conjunctions as follows:

Grammar	Rule	Diagram
	Sent.Comp.Verb	
	Conjunction	

Example 3.3.7. For ALICE SEES BOB DANCE we now obtain:



In this example, the sentential subphrase BOB DANCE(S) is ‘contained’ in a grey ‘phrase bubble’. Recalling Convention 3.3.6, although text diagrams in general are not restricted by planarity, we impose the condition that – apart from noun wires which have specific diagrams that allow them to – no wires penetrate the boundary of the ‘bubble’; in this way, the phrase regions capture the behaviour of phrase scope viewed as subtrees, or more generally as in dependency grammars.

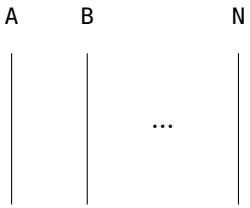
3.4 Text circuits

3.4.1 Definition

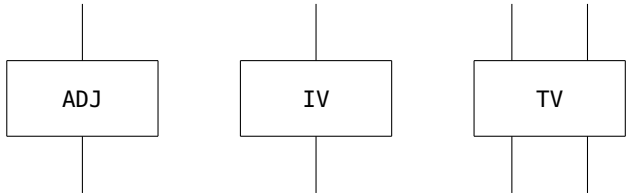
Our *text circuits* are made up of three ingredients:

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

Firstly, nouns are represented by wires, each 'distinct' noun having its own wire:



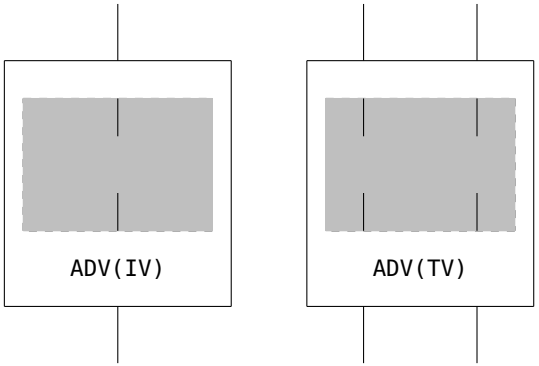
We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires:



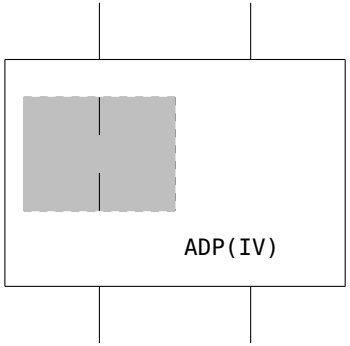
Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicat-

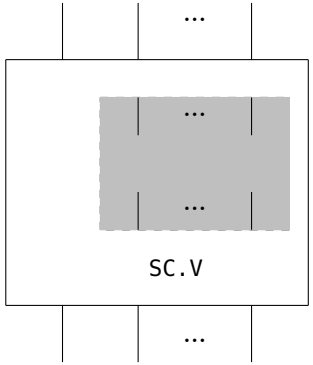
ing the shape of gate expected, and these should match the input- and output-wires of the box with the whole:



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

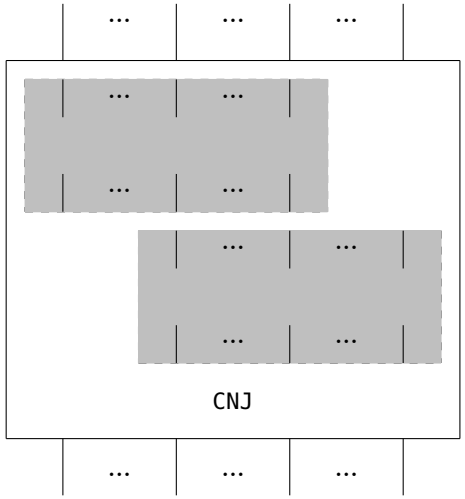


For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit:[6]

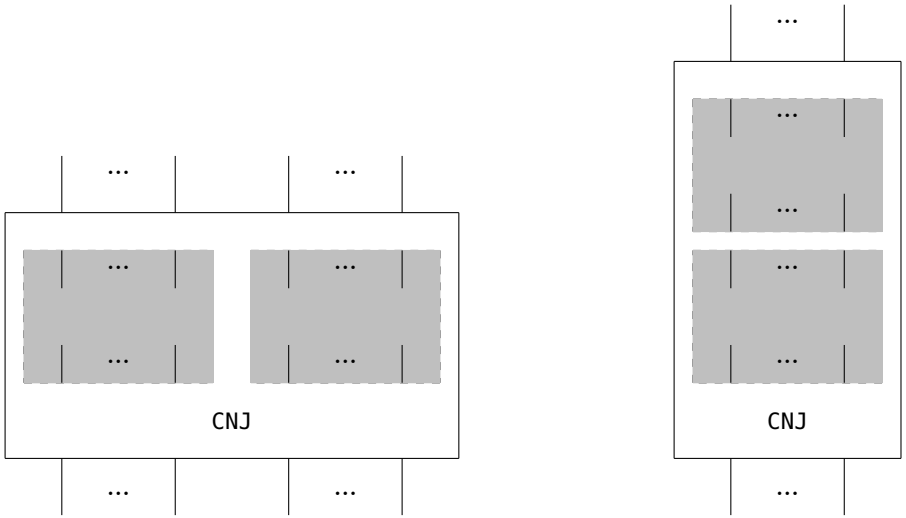


[6] The ‘dot dot dot’ notation within boxes is graphically formal . Interpretations of such boxes were earlier formalised in . ; ; ; and

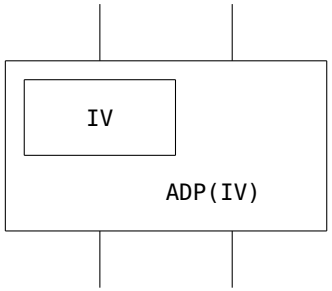
while conjunctions are boxes that take two circuits which might share labels on some wires:



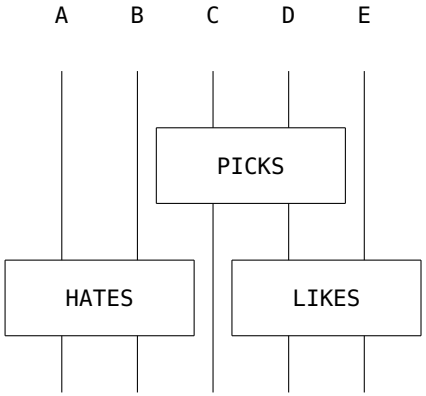
As special cases, when the noun-wires of two circuits are disjoint (left), or coincide (right), conjunctions are depicted as follows:



Of course filled up boxes are just gates

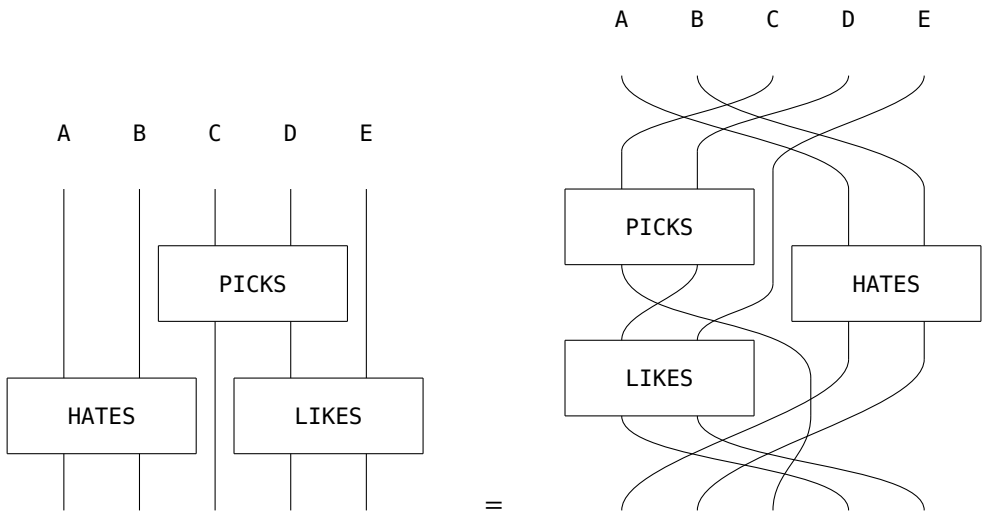


and we will now discuss how those compose. Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits, which by convention we read from top-to-bottom:

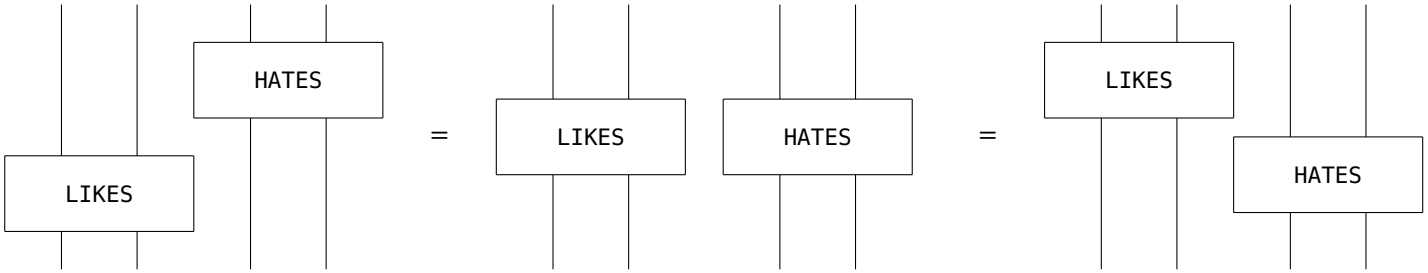


Convention 3.4.1. Sometimes we allow wires to twist past each other, and we consider two circuits the same if their gate-

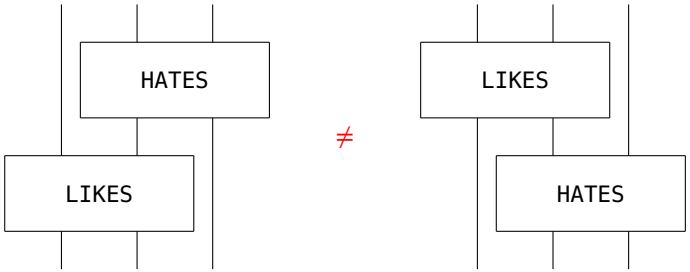
connectivity is the same:



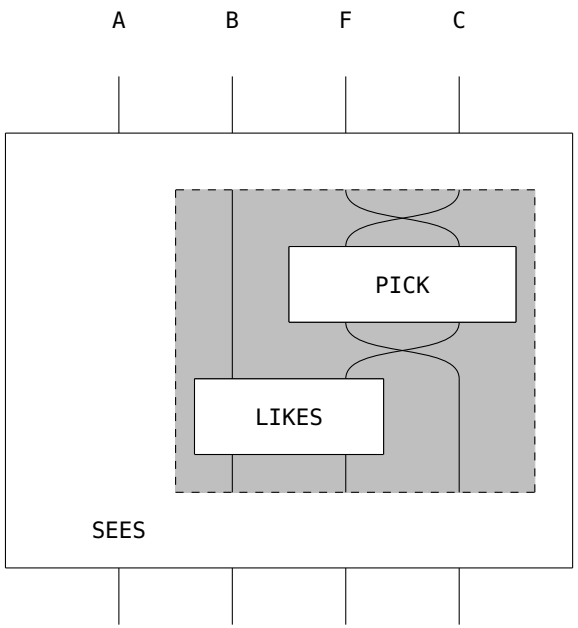
Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel:



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:

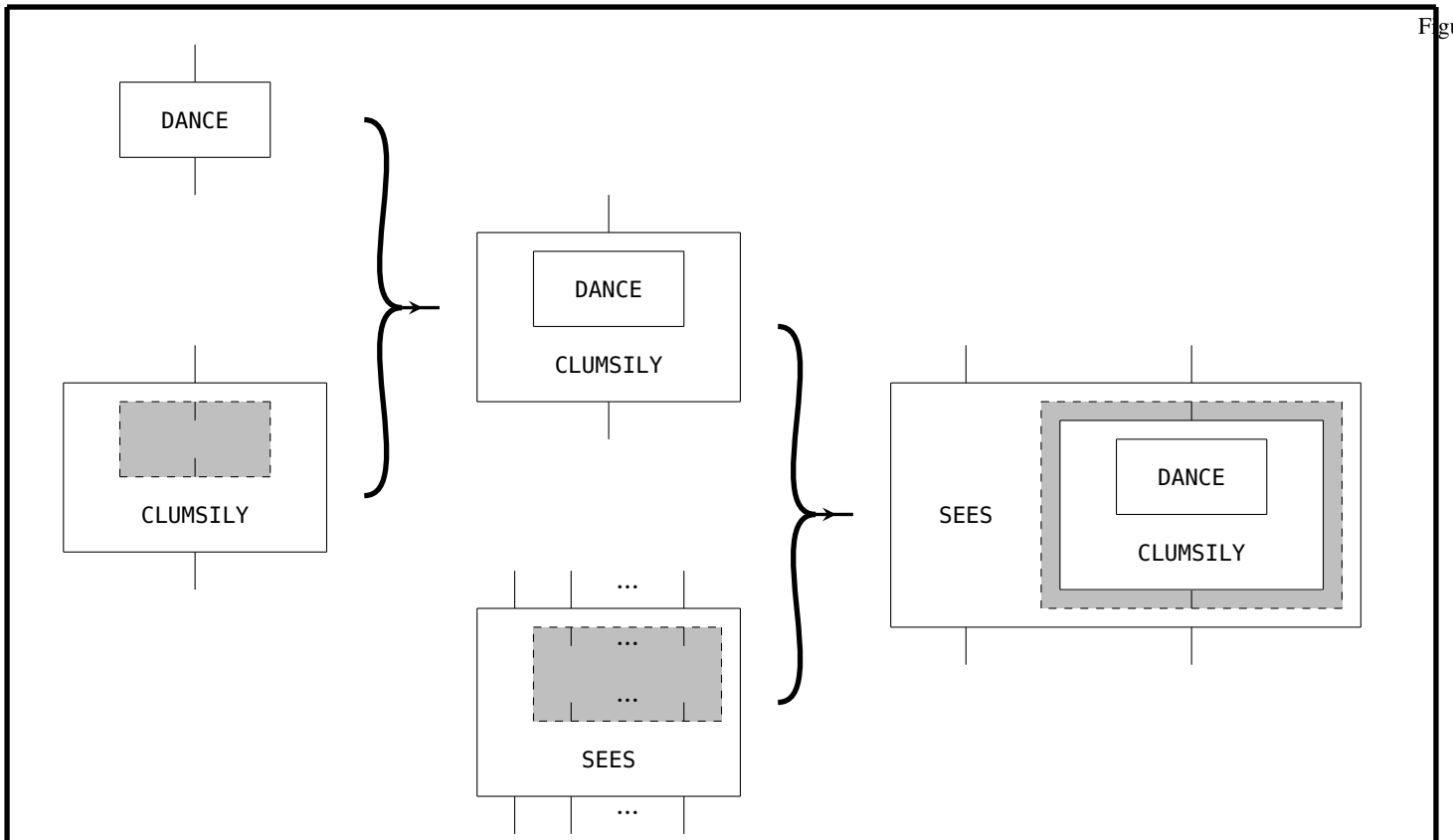


Example 3.4.2. The sentence ALICE SEES BOB LIKES FLOWERS THAT CLAIRE PICKS can intuitively be given the following text circuit:



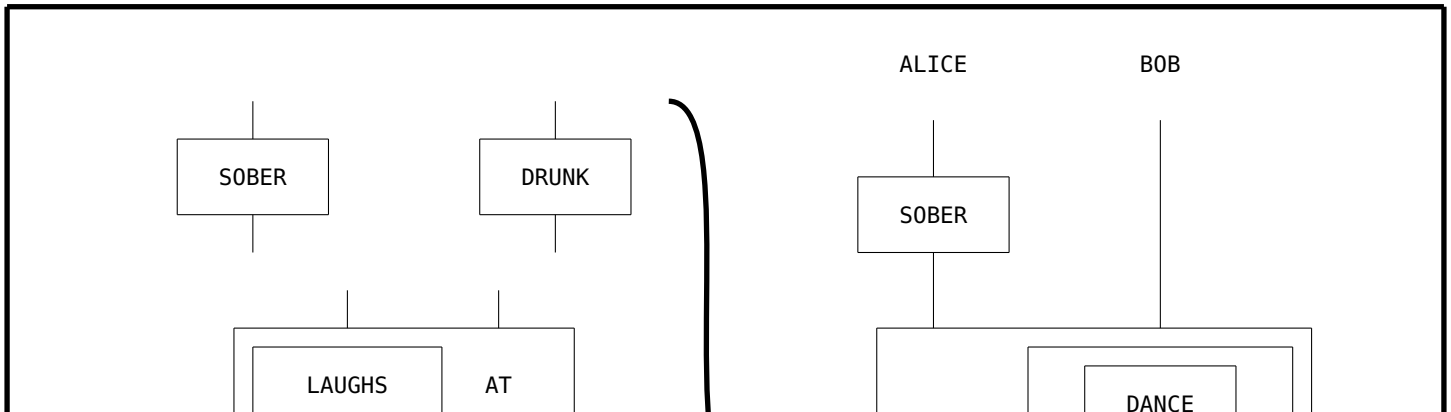
Example 3.4.3. Figure 3.28 illustrates how to directly generate text circuits, by providing an example analogous to the text generated in Figure 3.27 where we used our hybrid grammar.

Figure 3.28: Generating text circuits directly.



Inserting gates in boxes yields gates.

One can also take entire circuits to be gates.



3.5 Mathematical results

3.5.1 Main Text Circuit Theorem

We assume some finite vocabulary of words with grammatical types we have considered so far. Let $\mathfrak{T}$ denote the set of all raw text generated by our grammar. Let $\mathfrak{C}$ denote the set of all text circuits. We prove the following results:

1. We show that the translation rules of Section 3.5.3 algorithmically associate any text with grammar to a text circuit.

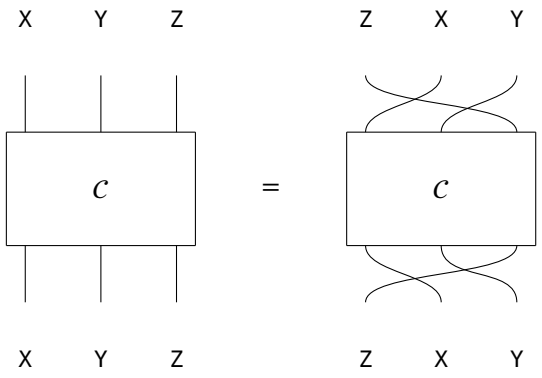
2. All text circuits are obtainable in this way.

Theorem 3.5.1. The translation rules of Section 3.5.3 define a surjection $\mathfrak{T} \twoheadrightarrow \mathfrak{C}$.

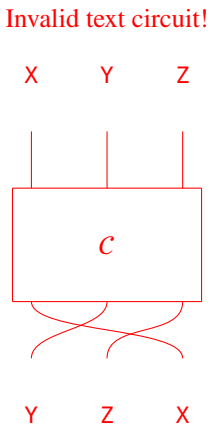
3.5.2 Refinements, extensions, conventions

There are clarifications to be made before proving the Theorem, which is a claim about *all* text circuits, and *all* raw texts generated by hybrid grammar. Convention 3.5.2 clarifies what exactly counts as a text circuit for our purposes. We address an edge-case circuit – where a wire appears in a box unconnected to any gates – in Refinement 3.5.3. For the proof-strategy of Lemma 3.5.15, which relies on a step that treats multiple gates in parallel as single sentences, we accommodate textual elements such as commas and 'contentless' conjunctions such as `AND ALSO` by means of a generic conjunction `[&]`. We note in Refinement 3.5.5 that the phrase-scope distinctions of hybrid grammar are properly expressible in raw text by means of a generic contentless complementiser `[THAT]`. Finally, we extend the definition of text circuits to include 'reflexive pronoun boxes' as structural components.

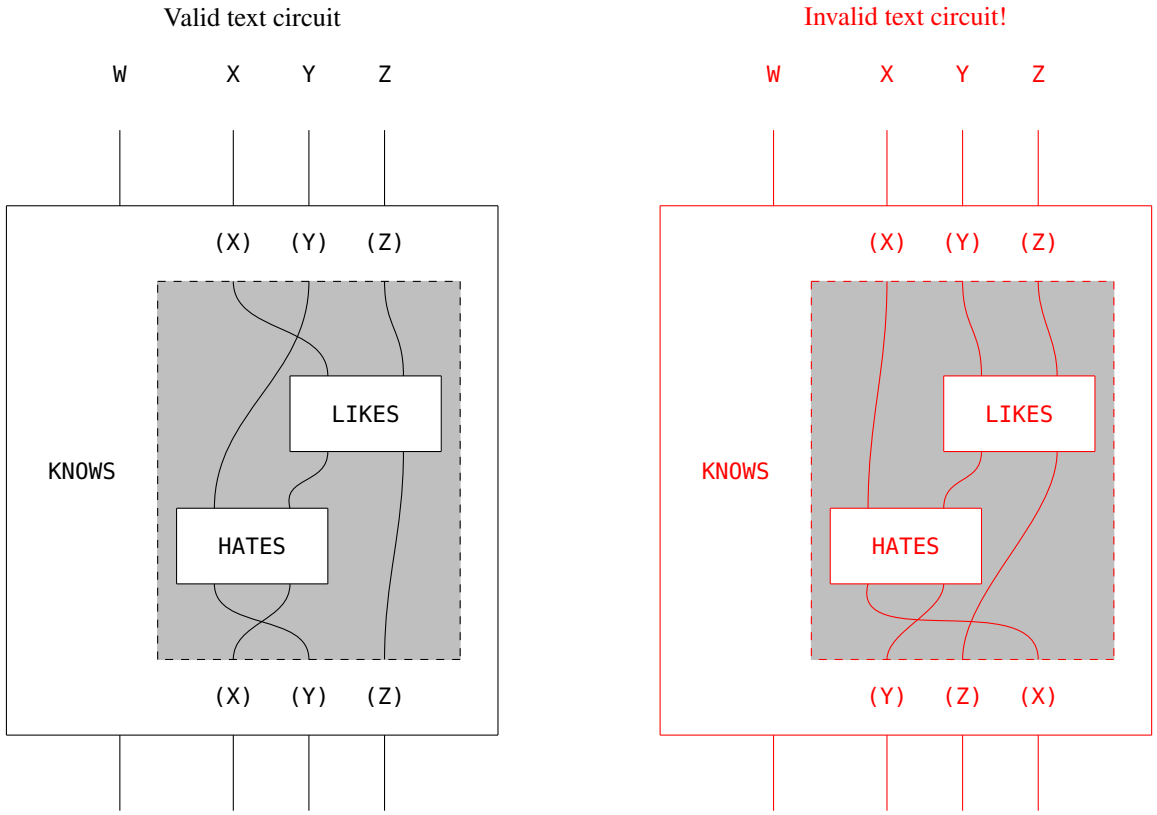
Convention 3.5.2 (Wire ordering)**.** We elaborate Convention 3.4.1 for depicting circuits. Every wire of a circuit is labelled with a noun, and when depicted these noun-labels are forced to take a left-to-right order. By convention, this order is kept the same for inputs and outputs of a circuit, by introducing wire twists where necessary. Note that two circuits with the same connectivity but different orders of noun-labels are still considered equal:



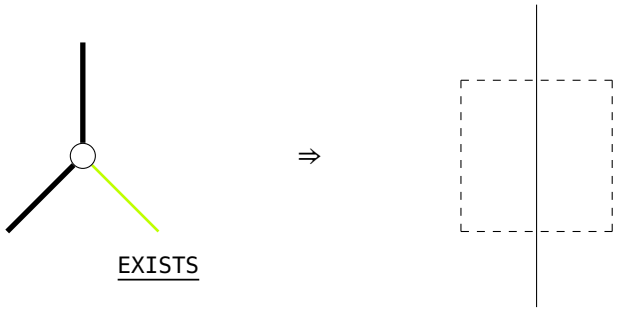
To give a negative example, the following diagram **violates** the visual convention, because the input and output wires have a different noun order:



The contents of a box are circuits as well, which must obey the same input-output agreement rules:



Refinement 3.5.3 (The verb `EXISTS`)**.** We introduce a special structural rule for an identity wire that does not participate in any gates. In these cases, we interpret the identity wire as the special transitive verb `EXISTS` at the level of raw text. The rewrite rule for `EXISTS` from text diagrams to circuits is as follows:



Refinement 3.5.4 (Sentence composition using `[&]`)**.** So far, we have mostly considered listing sentences as text to compose them, e.g.:

`ALICE RUNS. BOB DRINKS.`

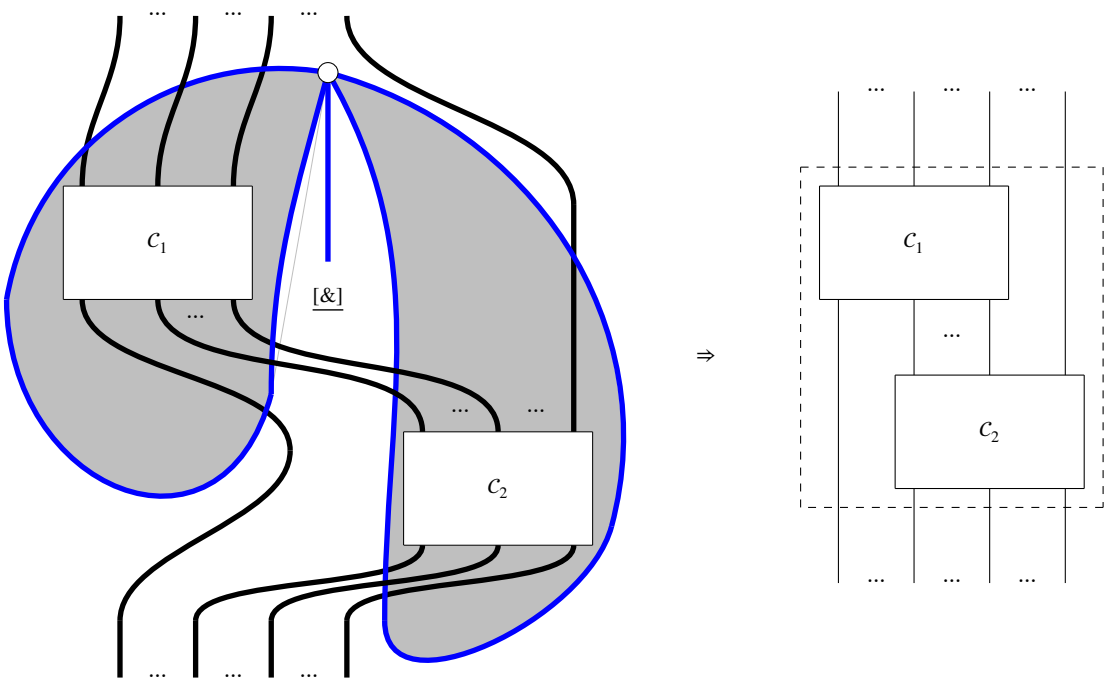
For the purposes of Lemma 3.5.15, we define a special conjunction `[&]` at the level of text, which allows us to consider multiple gates in parallel as arising from a single sentence in an unambiguous way. In English, there are multiple 'contentless' ways that two sentences may be composed as a single sentence. For example, by an ampersand:

`ALICE RUNS [&] BOB DRINKS.`

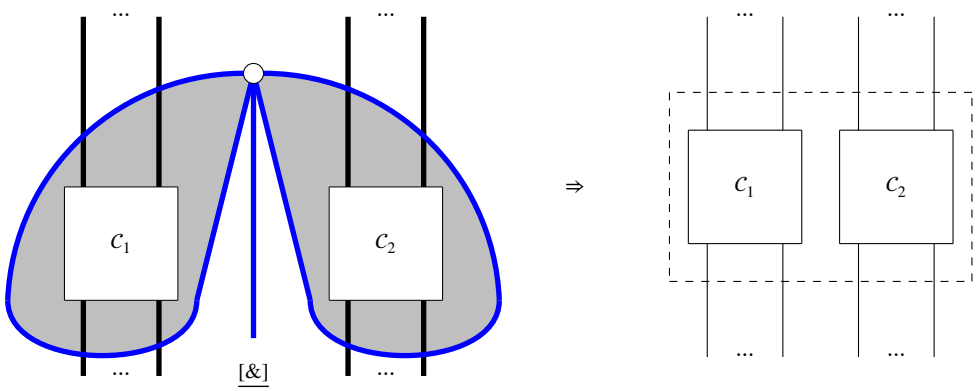
or by 'contentless' conjunctive phrases such as `AND ALSO` e.g.:

`ALICE RUNS AND ALSO BOB DRINKS.`

The special conjunction `[&]` is intended to be a catchall for these kinds of 'contentless' conjunctions. At the level of circuits `[&]` is interpreted as gate composition, and its rewrite rule from text diagrams to circuits is as follows:



Note that in the case where the noun wires are disjoint, the rewrite rule for [&] just yields parallel composition of circuits:



Refinement 3.5.5 (Sentence composition within phrase scope, and the complementizer [THAT])**.** Consider the sentence:

CLAIRE SEES ALICE RUNS [&] BOB DRINKS.

For us, when the raw text is equipped with hybrid grammar structure, there is no ambiguity as to whether Claire sees both that Alice runs and Bob drinks, or just the former. Recalling Refinement 3.5.4, neither reading of [&] as a comma or AND ALSO resolves this ambiguity. Since our broad claim is that we address a restriction of natural language, we have to justify that this distinction made by hybrid grammar structure is one that actually has a counterpart at the level of raw text. To disambiguate in a similar manner as our previous refinements, we introduce a complementiser [THAT], which behaves much like the phrase scope formal types (and) introduced in Section 3.2. Complementisers are words such as HOW or WHAT, which prefix sentential complements e.g.:

CLAIRE SEES HOW ALICE RUNS.

CLAIRE SEES WHAT BOB DRINKS.

We do not consider complementisers in general here. Instead we use just one 'contentless' complementiser [THAT], which we freely omit when it is implicit. Returning to the initial example, the presence of a complementiser allows us to distinguish, at the level of raw text, the case where Claire **only** sees Alice running:

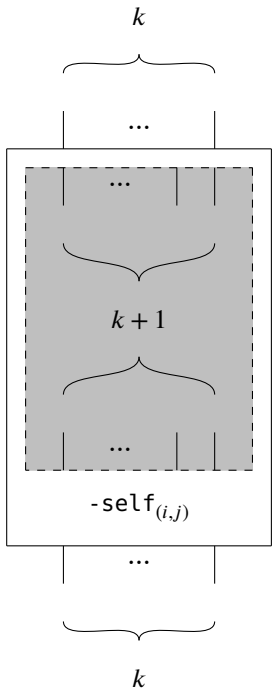
CLAIRE SEES [THAT] ALICE RUNS [&] BOB DRINKS.

from the case where Claire sees **both** that Alice runs and that Bob drinks:

CLAIRE SEES [THAT] ALICE RUNS [&] [THAT] BOB DRINKS.

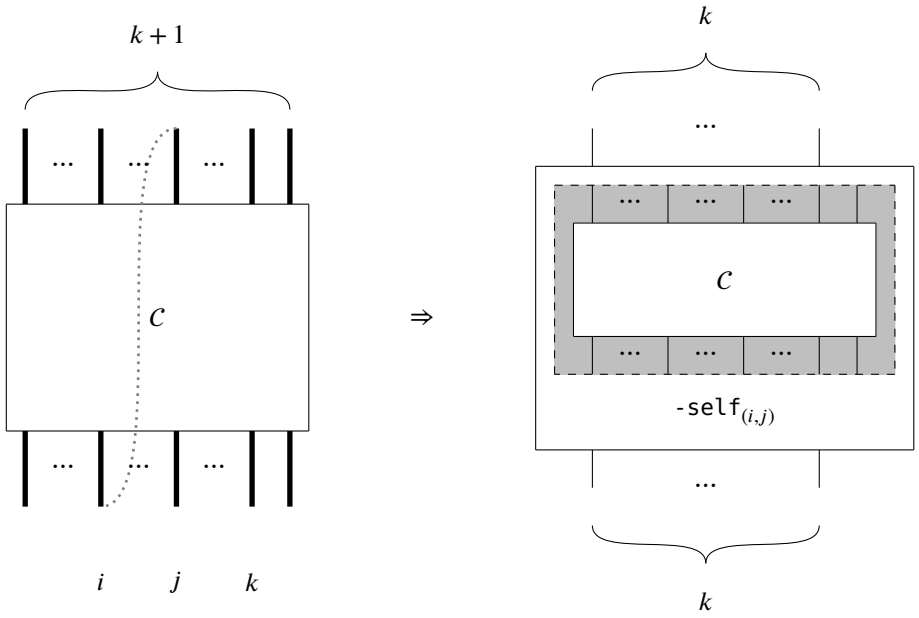
So, when we encounter the conjunction [&] within a phrase scope, we interpret it at the level of text as AND ALSO THAT. More generally, we use [THAT] to signify the open-bracket that begins a scoped phrase in raw text.

Convention 3.5.6 (Reflexive pronoun boxes)**.** We extend the definition of text circuits to accommodate reflexive pronouns, such as `HERSELF`, `ITSELF`. We treat pronominal links as gate composition in all other cases, but by definition, a reflexive pronoun in a circuit occurs when an output noun wire must compose with an input noun wire in its relative past, which cannot be drawn as a circuit. So, at the level of circuits, we assert a family of 'reflexive pronoun boxes'. For all $k > 1 \in \mathbb{N}$, and for all pairs of indices $1 \leq i < j \leq k$, we have a box:



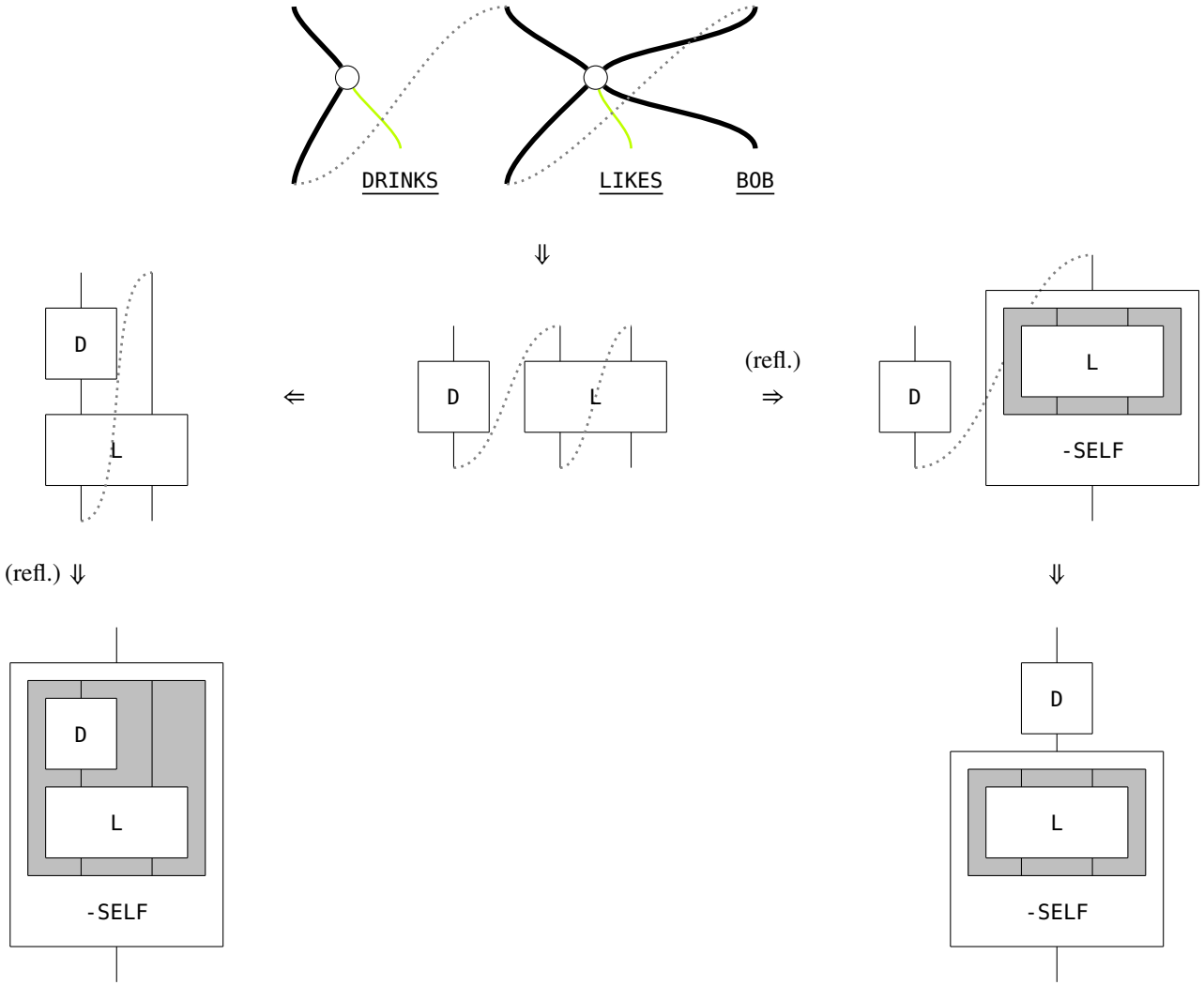
These boxes are the targets of rewrites from text diagrams when a pronominal link elimination would connect an output of a

circuit to an input, ‘doubling back’ a noun wire:



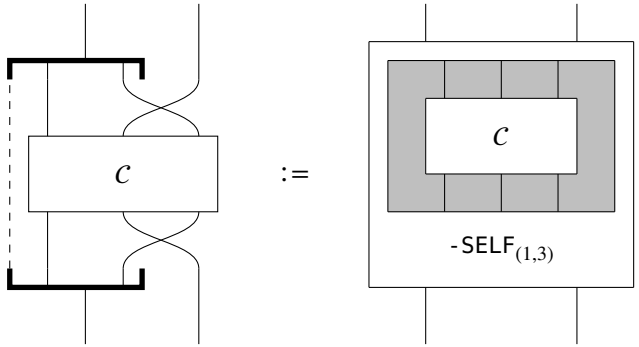
Refinement 3.5.7 (Coherence of reflexive pronoun boxes)**.** A *critical pair* in a rewrite system is a situation in which two rules are applicable to a term, and result in different final outcomes. Critical pairs are hence the only obstacle to the claim of Proposition 3.5.14. Reflexive pronouns are involved in all critical pairs that arise in rewriting text diagrams into circuits. Consider the following example, for the text diagram corresponding to the text:

BOB DRINKS. BOB LIKES BOB.

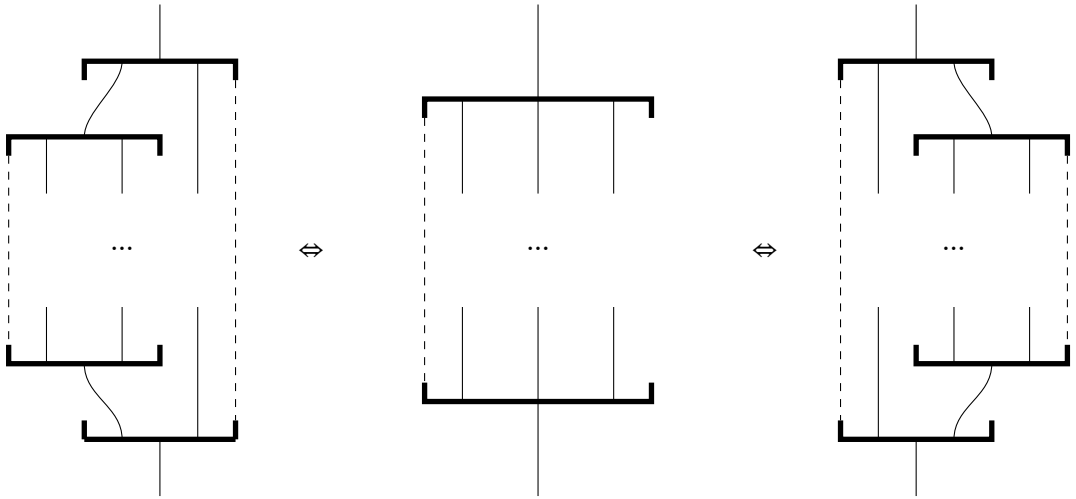


So the order of pronominal link eliminations and (their special case) reflexive pronoun box introductions matters. As we show in Corollary 3.5.9, we may generally deal with these critical pairs by a convention that all reflexive pronoun box introductions come before the elimination of other pronominal links. Recalling the footnote in Section **??**, a pronominal link in a text diagram can take two directions. Neither is strictly appropriate in the case of reflexive pronouns, where a pronominal link should *identify* two wires. So here we introduce syntactic sugar for the reflexive pronoun box, and introduce rewrite rules such that

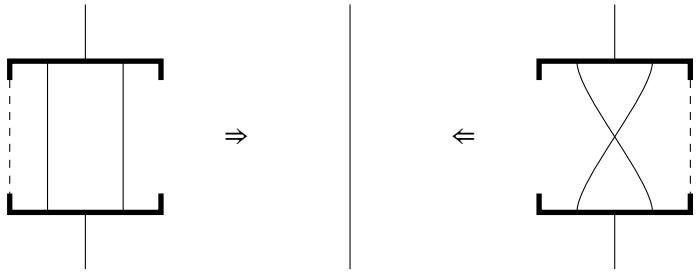
identification of wires is respected. We introduce the following shorthand for reflexive pronominal links:



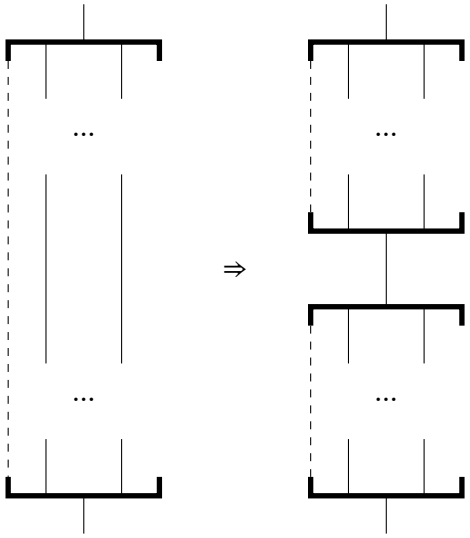
Identifying wires is *associative*, which we enforce by the following bidirectional rewrites:



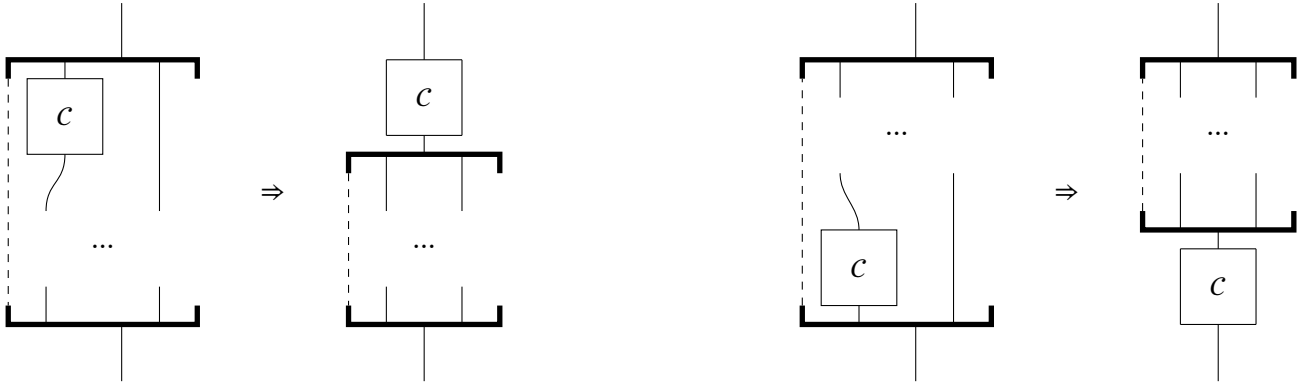
A reflexive pronoun box with only identity wires inside can be eliminated by an *identity* rule:



Conversely, we can introduce new reflexive pronoun boxes by *splitting* existing ones along identified wires:



And finally, a special aspect of reflexive pronoun boxes is that gates on one wire may *slide* out of them. This reflects the fact that sentences that only modify a single noun never require a reflexive pronoun, which would refer to another noun argument:



3.5.3 Text diagrams to text circuits

This section presents lemmas that together constitute the hard work in the proof of Proposition 3.5.14, which claims the existence of a function $\mathfrak{T} \to \mathfrak{C}$.

First, Lemma 3.5.8 and Corollary 3.5.9 constructively organise the rewrites of Refinement 3.5.7 into a function from text diagrams to text diagrams with no pronominal links. Second, Lemma 3.5.11 introduces new rewrite rules and ancilla types which

constructively constitute a function that takes text diagrams with no pronominal links nor phrase scoping into a normal form that corresponds to a single circuit gate. Third, Lemma 3.5.11 is extended by Lemma 3.5.12 to account for text diagrams with phrase scope as well. When taken together with the content of Section 3.3 – which provides a correspondence that sends every hybrid grammar text to a text diagram – the three stages above complete the description of a function from hybrid grammar text to text circuits.

Lemma 3.5.8 (Shrinking reflexive pronouns)**.** The associativity, identity, splitting, and sliding rewrite rules for reflexive pronouns suffice to recast text circuits in a form where every reflexive pronoun box contains exactly one gate.

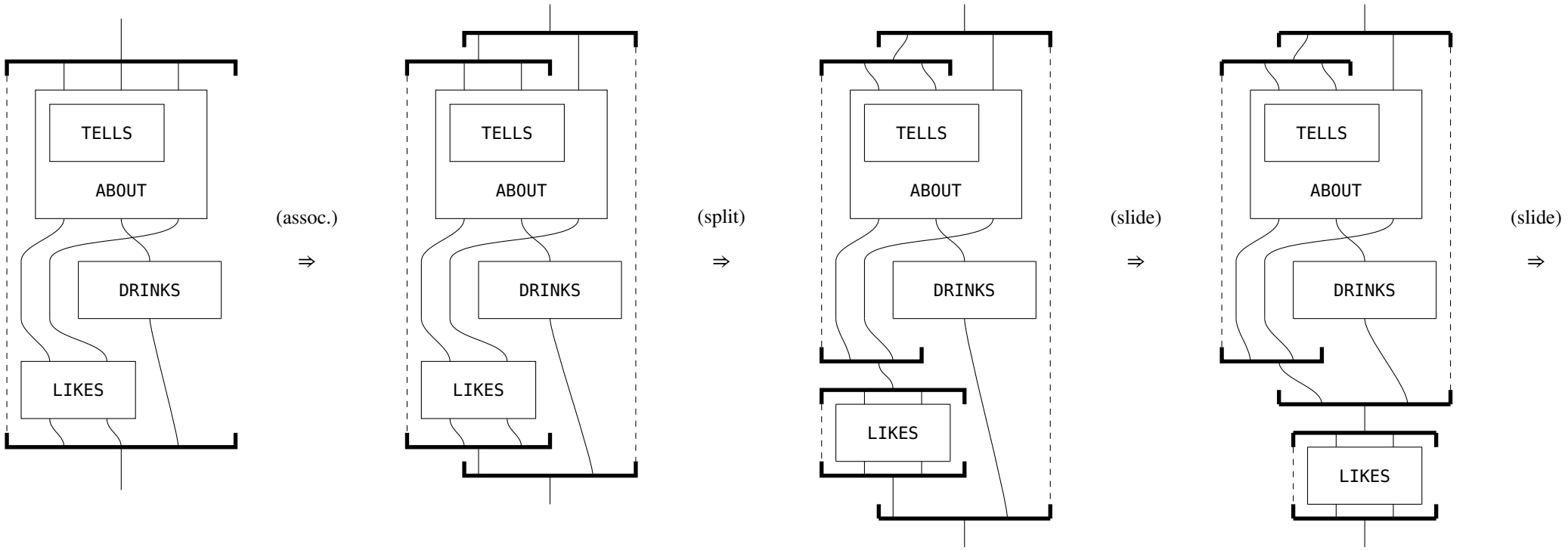
Proof. Nested reflexive pronoun boxes can be fused and rearranged according to the associativity rule. Any reflexive pronoun box containing two or more gates composed sequentially along an identified wire can be split by the splitting rule. The splitting and identity rules may be applied to eliminate twists in reflexive pronoun boxes containing a single gate, such that the inputs and outputs align. □

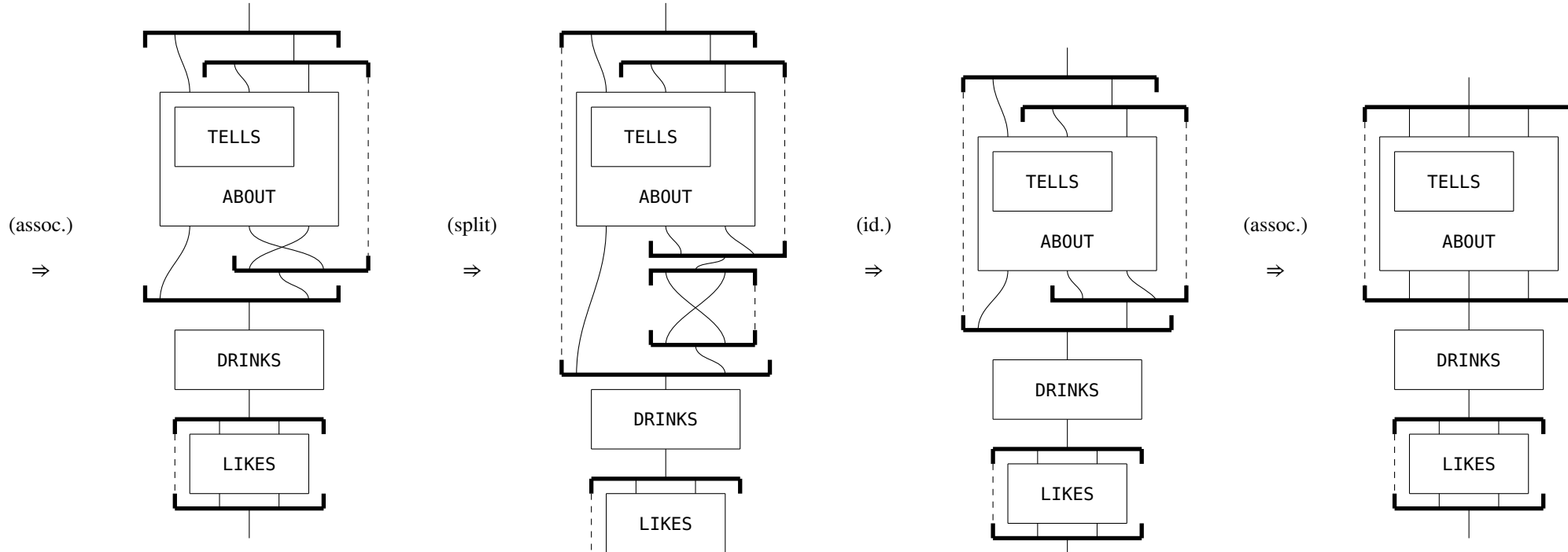
Corollary 3.5.9. Text circuits obtained by the shrinking lemma coincide with text circuits obtained from text diagrams where reflexive pronoun box reductions are applied before other pronominal link eliminations.

Proof. By construction, reflexive pronouns only occur within a sentence, and sentences correspond to individual gates. The rewrite rules of Lemma 3.5.8 preserve gate connectivity, which is determined by non-reflexive pronominal links. □

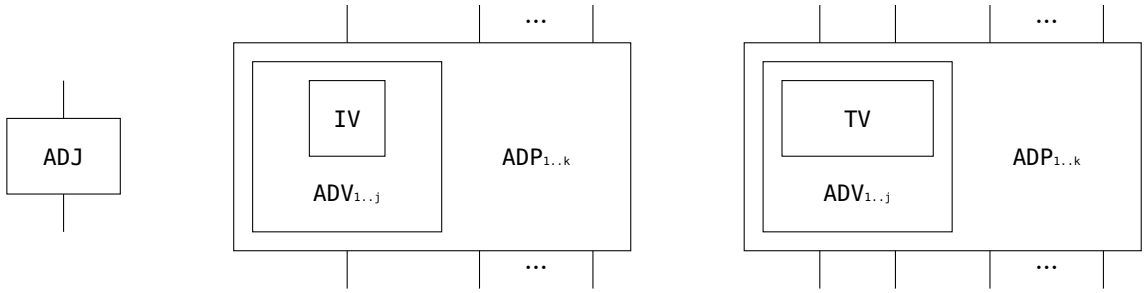
Example 3.5.10 (An application of the shrinking lemma)**.** Consider a text circuit that glosses as the text:

BOB TELLS BOB ABOUT BOB. BOB DRINKS. BOB LIKES BOB.

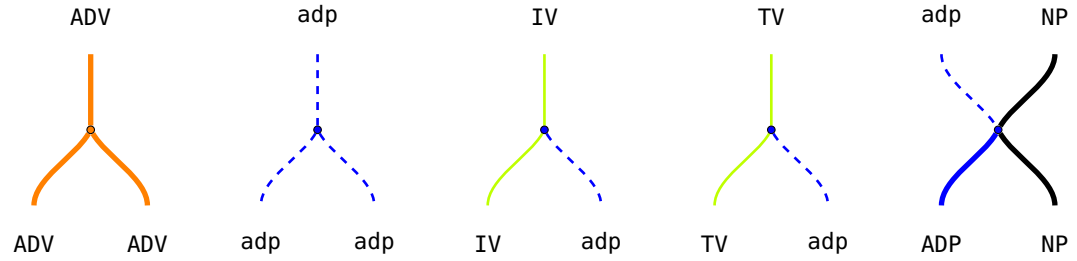

TELLS
ABOUT
DRINKS
LIKES
(assoc.)
⇒
(split)
⇒
(slide)
⇒
(slide)
⇒


(assoc.)
⇒
TELLS
ABOUT
DRINKS
LIKES
(split)
⇒
(id.)
⇒
(assoc.)
⇒

Lemma 3.5.11. We present ancillas and rewrites in Table 3.29 such that every text diagram without pronominal links and without phrase scope can be viewed as a unique text circuit constructed out of gates of the following forms:

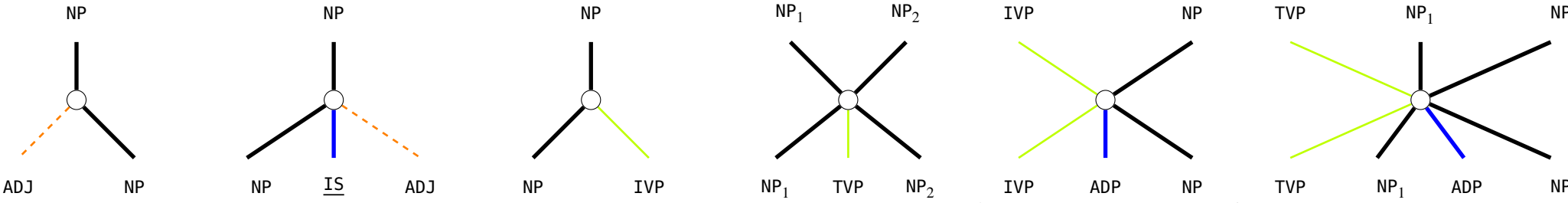


Proof. We introduce an ancillary wire type `adp`, along with the following ancillary text diagram components.



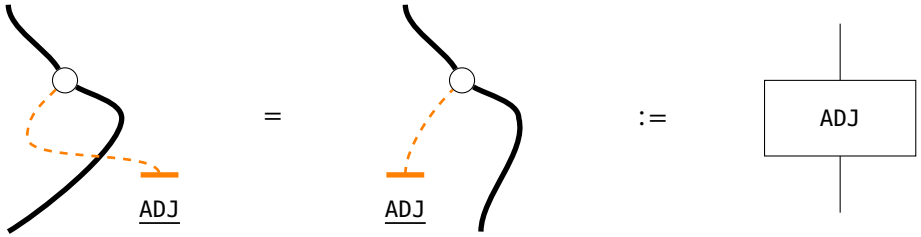
We present normalisation rewrite rules in the table of Figure 3.29.

The `NP` wires of a text diagram correspond to the noun wires in a text circuit. For a text diagram from a hybrid grammar structure without pronominal links and phrase scope, there are six non-label text diagram nodes that involve `NP` wires; each such node corresponds to either a gate or a box.



The `IS`-elimination rule recasts all instances of case (2) nodes as case (1) nodes. Recalling by Convention 3.3.3 that in text diagrams only connectivity matters, we may contract the ADJ label associated with each instance of case (1), interpreting as a

gate in a text circuit as follows.



Cases (5) and (6) are adpositions, which can only appear in the presence of an IV or TV wire respectively, which we refer to collectively as a V wire for simplicity. The only way such wires appear is by cases (3) and (4) respectively, and the only way they end is by V labels.

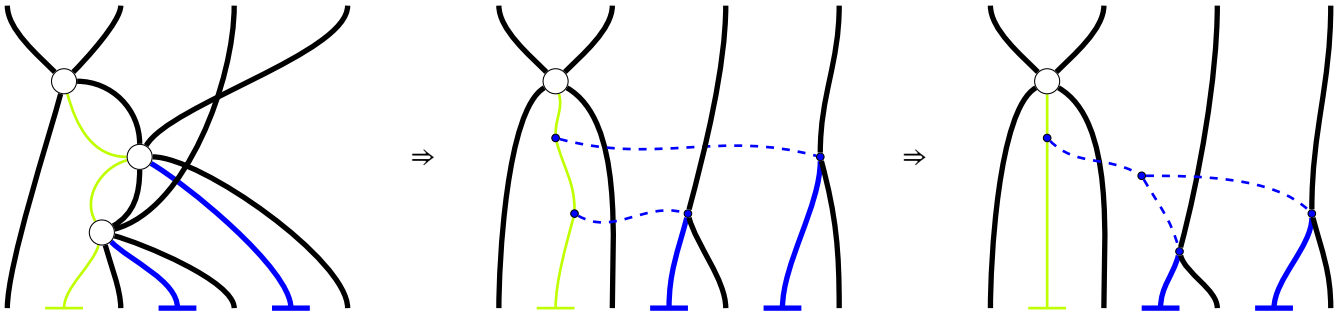
We cannot repeat the same contraction trick as in ADJ labels for V labels directly, because in between a node of case (3) or (4) and its associated V label, there may be ADV nodes, or ADP nodes as in (5) and (6). However, all such nodes have one input and one output V wire, so all ADV and ADP nodes, and their labels, can be unambiguously associated with a single V label.

The ancillas and rewrites are in service of obtaining a normal form for the ADV and ADP nodes and labels contracted to their parent V node of case (3) or (4). We analyse the three possible cases for a given V wire in the following order to illustrate the purpose of the rules; only ADV nodes appear; only ADP nodes appear; both kinds of nodes appear.

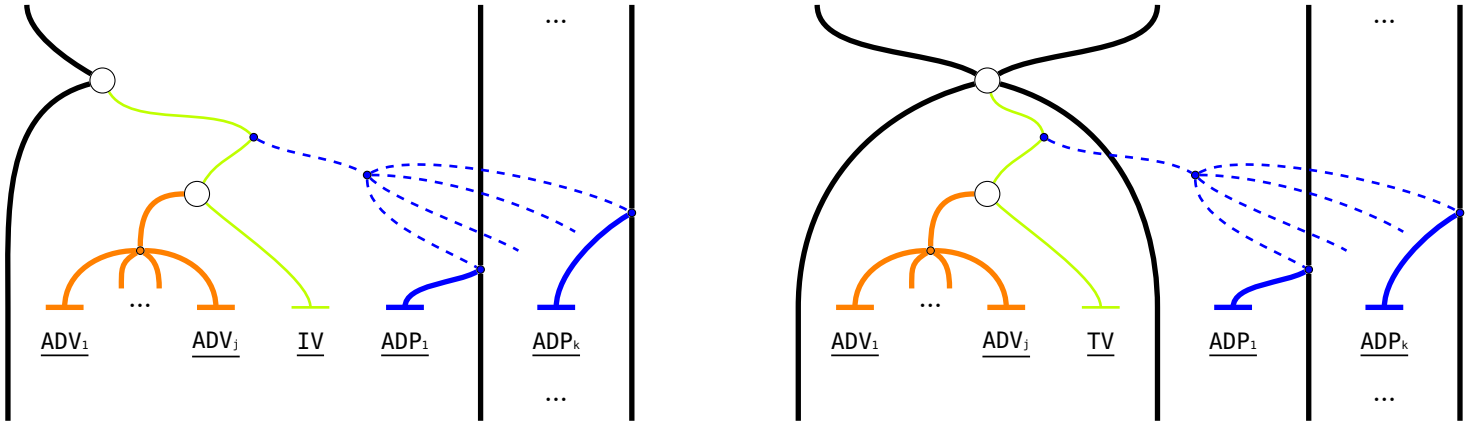
In the first case, when two ADV nodes appear adjacently on the same V wire, we apply the ADV-gather rule. The ADV-assoc bidirectional rewrite rule makes the order of ADV-gather applications irrelevant in the case of multiple adverbs for the same verb. Contracting labels, these rules allow re-expression of multiple ADV nodes on the same V wire as follows:

In the second case – which also addresses nodes of type (5) and (6) – when there are multiple ADP nodes, we first apply the ADP-ancilla rules to introduce the adp ancilla type. We may then apply adp-gather, mirroring the treatment for ADV wires: similarly, the adp-assoc rule makes the order of adp-gather rewrites irrelevant in the case of multiple adpositions on the same

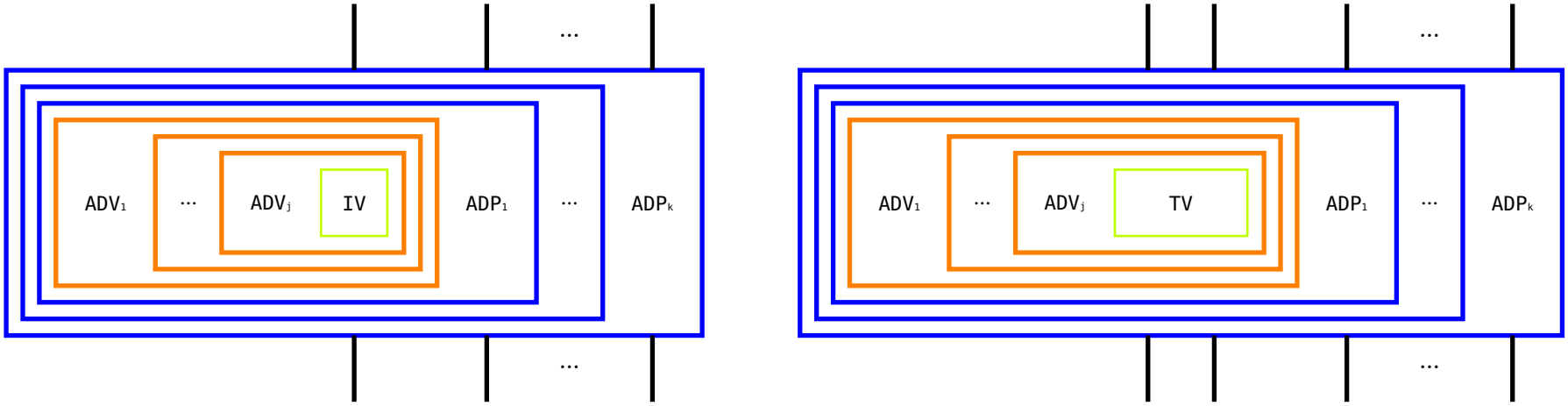
verb. These rules allow re-expression of multiple ADP nodes on the same wire as follows, which we illustrate for two ADP(TV).



In the third and final case where there are ADV and ADP nodes on a V wire, we first eliminate all ADP nodes of type (5) or (6) by ADP-ancilla rewrites. Then by applying the adp-ADV-order rule, we may arrange all adp nodes above all ADV nodes on the V wire, then we apply the treatment of the previous two cases of only ADV or only adp nodes. Contracting ADV and ADP labels, this procedure obtains the following normal forms for every node of type (3) and (4) respectively.



Which we may read as gates of the following form respectively, colour-coded for legibility.



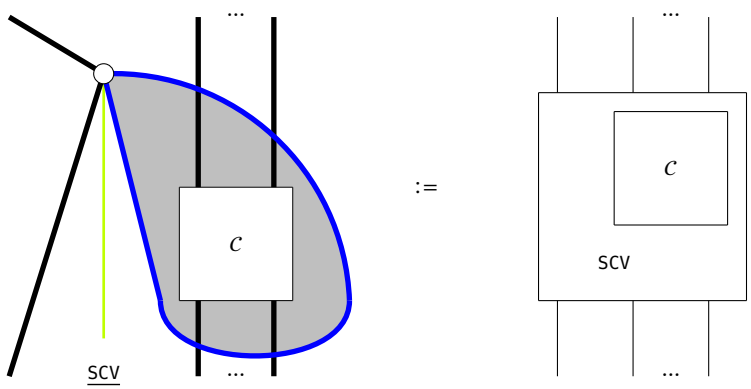
These are equivalent up to syntactic sugar to the last two gates in the claim. Uniqueness of the resulting circuit follows because none of the rules change the relative connectivity of nodes of type (1) through (4). □

Lemma 3.5.12. Every text diagram without pronominal links can be viewed as a unique text circuit.

Proof. We only need extend the claim of Lemma 3.5.11 to accommodate phrase scope. Recall by Convention 3.3.6 that phrase scope only allows NP wires and pronominal links to pass through, that NP labels may only occur outside of phrase scope, and that phrase scope nesting anywhere in the diagram is unambiguous. We prove the claim by (strong) induction, where the inductive hypothesis is that all text diagrams with phrase scope that nest at most k-levels deep can be viewed as a unique text circuit. For the base case, where there is no phrase scope, we are done by Lemma 3.5.11. For induction, assume we have a text diagram with phrase scope nesting of depth $k+1$. The topmost phrase scope node is either an SCV node or a CNJ node.

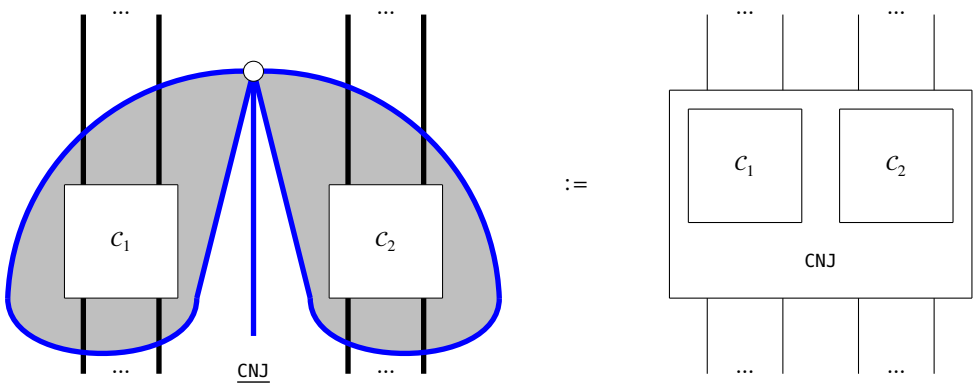
3.5.3.0.1 Verbs with sentential complement Since the same number of NP wires must enter as leave the phrase scope, by the inductive hypothesis we may express the content of phrase the phrase scope as a text circuit C, which we may unambiguously

read off as follows (modulo adverbs and adpositions for the SCV label as considered previously.)



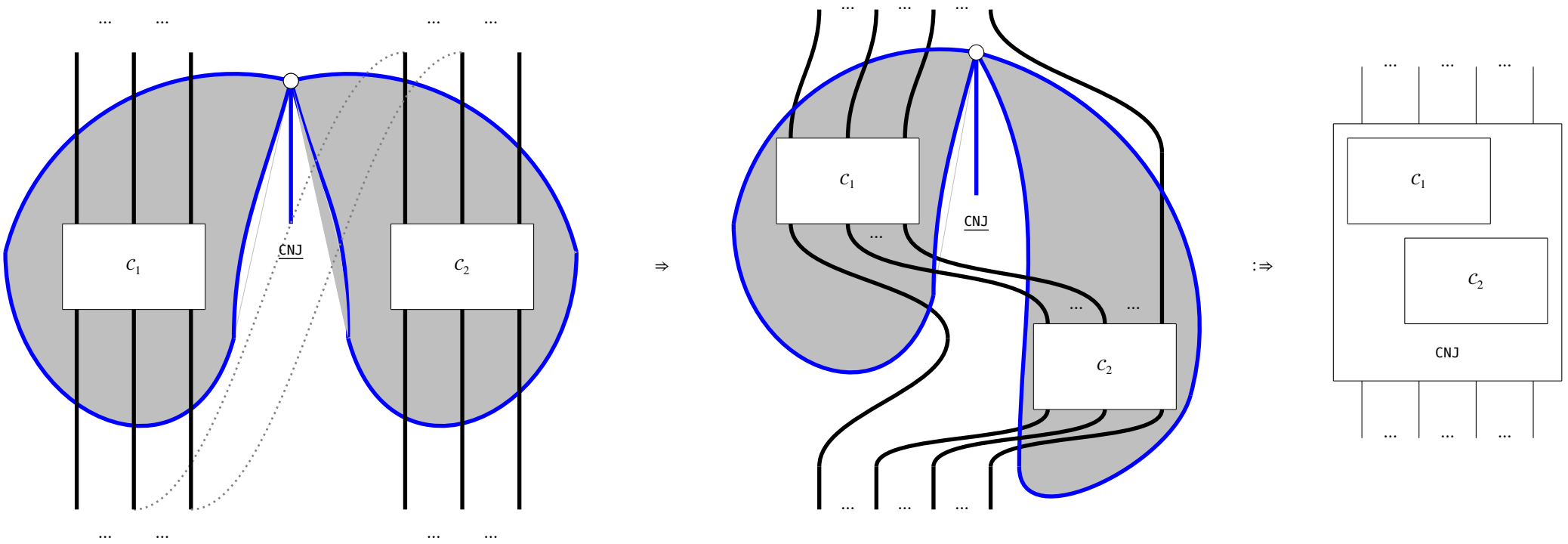
The remainder of the text diagram falls to the inductive hypothesis.

3.5.3.0.2 Conjunctions Similarly as before, we may express the contents of the left and right conjuncts as text circuits C_1 and C_2. There are two subcases we consider in order: where the conjuncts do not share NP wires, and when they do. In the former subcase, we obtain the following gate from a CNJ node.



In the latter subcase where the two conjuncts share noun wires – or, anticipating later work, when there are pronominal links between the conjuncts – we can introduce a conjugate box with overlapping arguments as syntactic sugar standing in for a

'normal' CNJ box inside reflexive pronoun boxes:



□

Example 3.5.13. Figure 3.30 illustrates how to obtain text circuits from text diagrams.

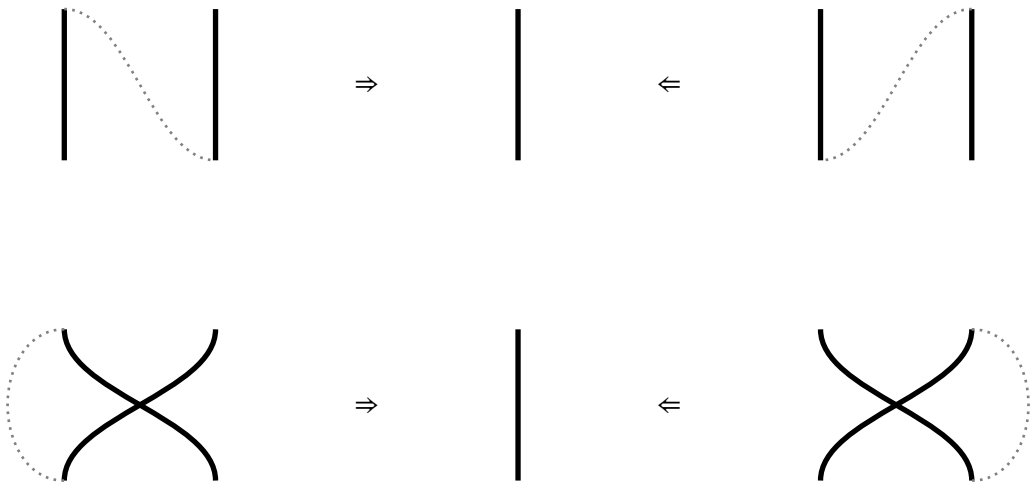
3.5.4 Proof of Theorem

It suffices to demonstrate the following. First, that the translation procedure from text to circuit given in Section 3.5.3 yields a function $\mathfrak{T} \to \mathfrak{C}$. Second, that this function is surjective.

Proposition 3.5.14. We have a function $\mathfrak{T} \to \mathfrak{C}$.

Proof. Lemma 3.5.12 handles all text diagrams obtained from text without pronominal links. It remains to be shown that pronominal links can be resolved uniquely in the setting of text diagrams. The following transformations eliminate pronominal

links.

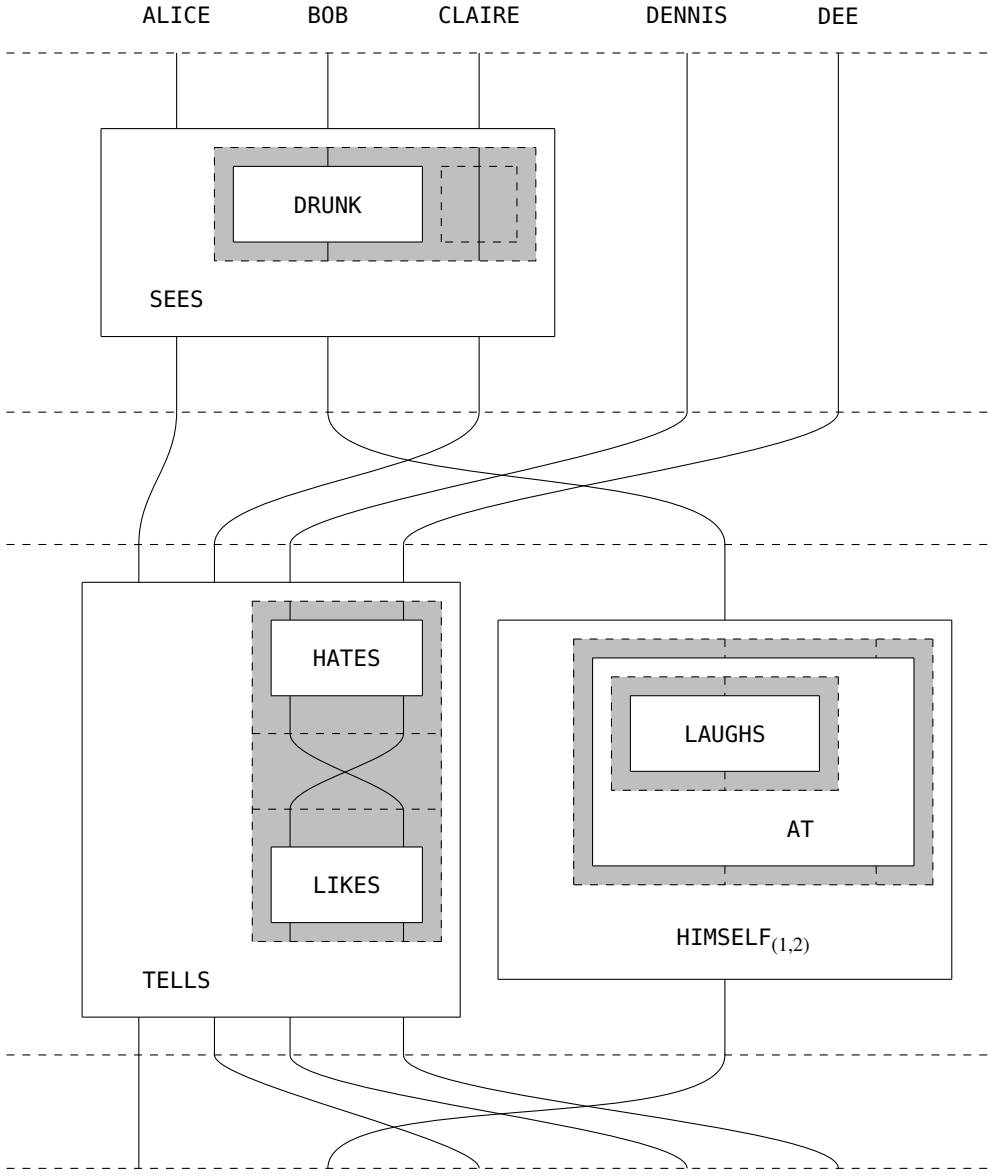


The only obstacle to interpreting these transformations directly as composition of text circuits is addressed by reflexive pronoun boxes, in Convention 3.5.6. The critical pairs that arise as a result of reflexive pronoun boxes are addressed by Refinement 3.5.7, and a normal form for reflexive pronoun boxes is constructively provided by Lemma 3.5.8. So the initial hybrid text determines the corresponding circuit. □

Proposition 3.5.15. $\mathfrak{T} \to \mathfrak{C}$ is surjective.

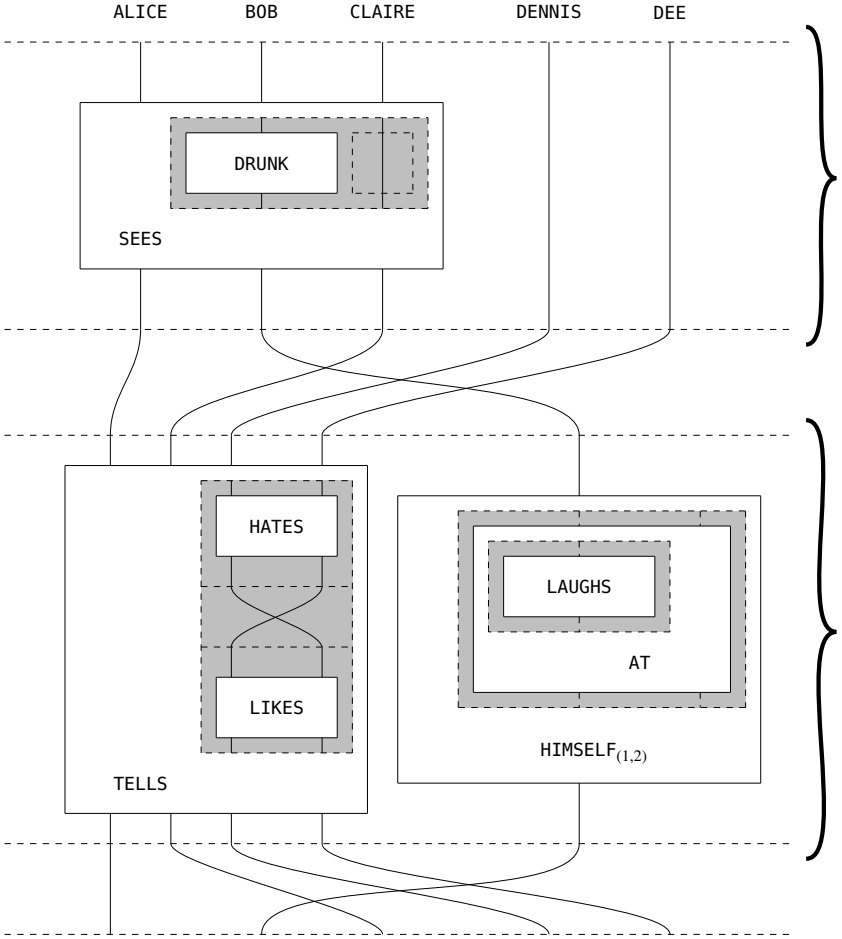
Proof. Surjectivity amounts to showing that an arbitrary circuit corresponds to a text that translates into that circuit; in other words, every text circuit can be textualised. First, we divide the circuit into horizontal slices, such that every slice either contains at least one gate, or contains only twisting wires. Here we exploit Refinement 3.5.3; when a wire inside the context of a

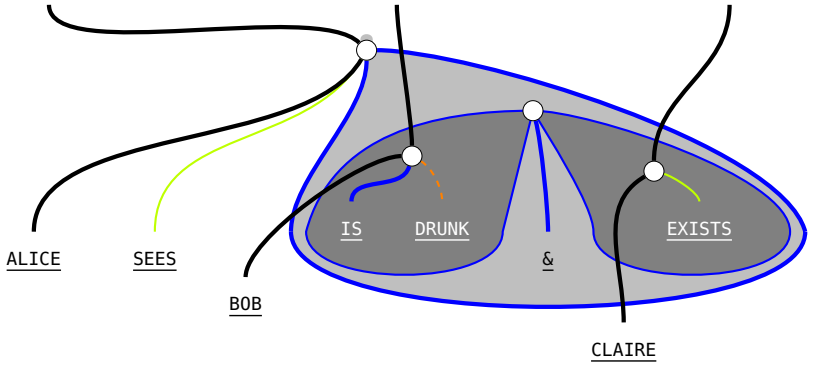
box does not connect to any gates, we assume it connects to an EXISTS gate. We iterate this slicing within the holes of boxes:

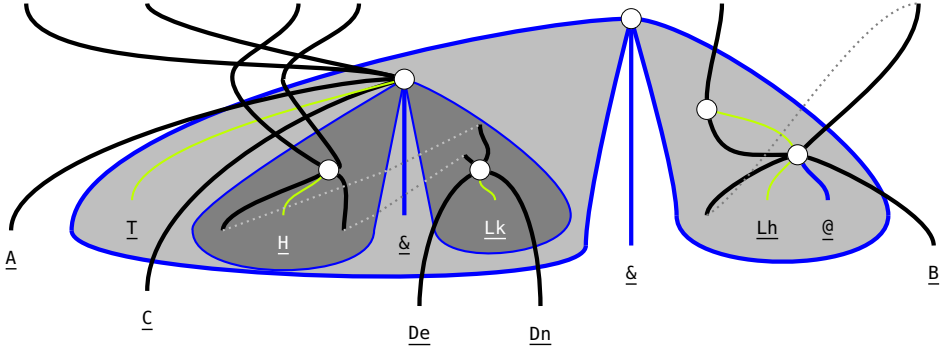


Now we exploit Refinement 3.5.4; every horizontal slice with gates obtained this way is a collection of gates composed in parallel, possibly with identity wires. Identity wires inside the holes of boxes that do not connect to gates we treat as connected to an EXISTS gate, as in the hole of SEES in the example. Other identity wires we do not need to mention in text. We deal with

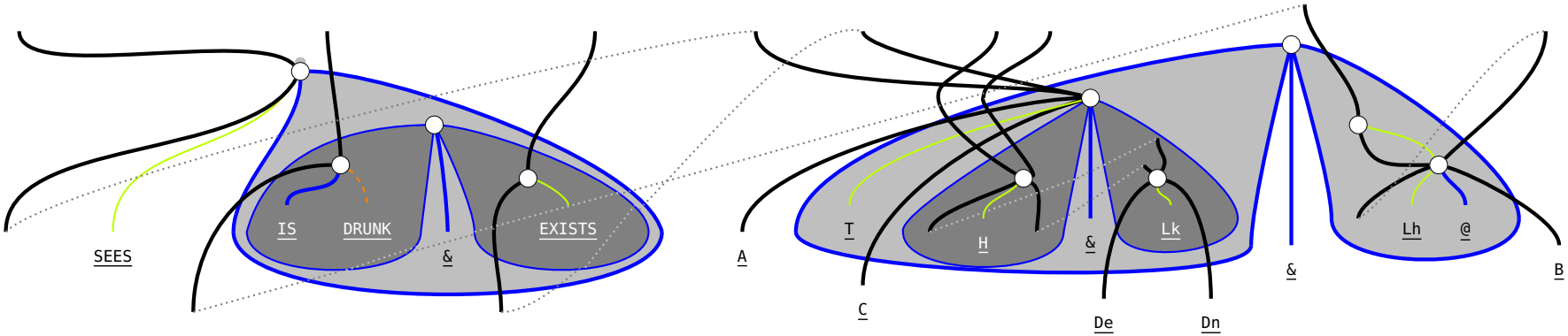
adjective gates by using the copular IS construction, for instance DRUNK in the example. We deal with reflexive pronouns by introducing the appropriate pronominal link, and just restating the noun, for instance HIMSELF in the example. We obtain a text diagram for each slice with gates:



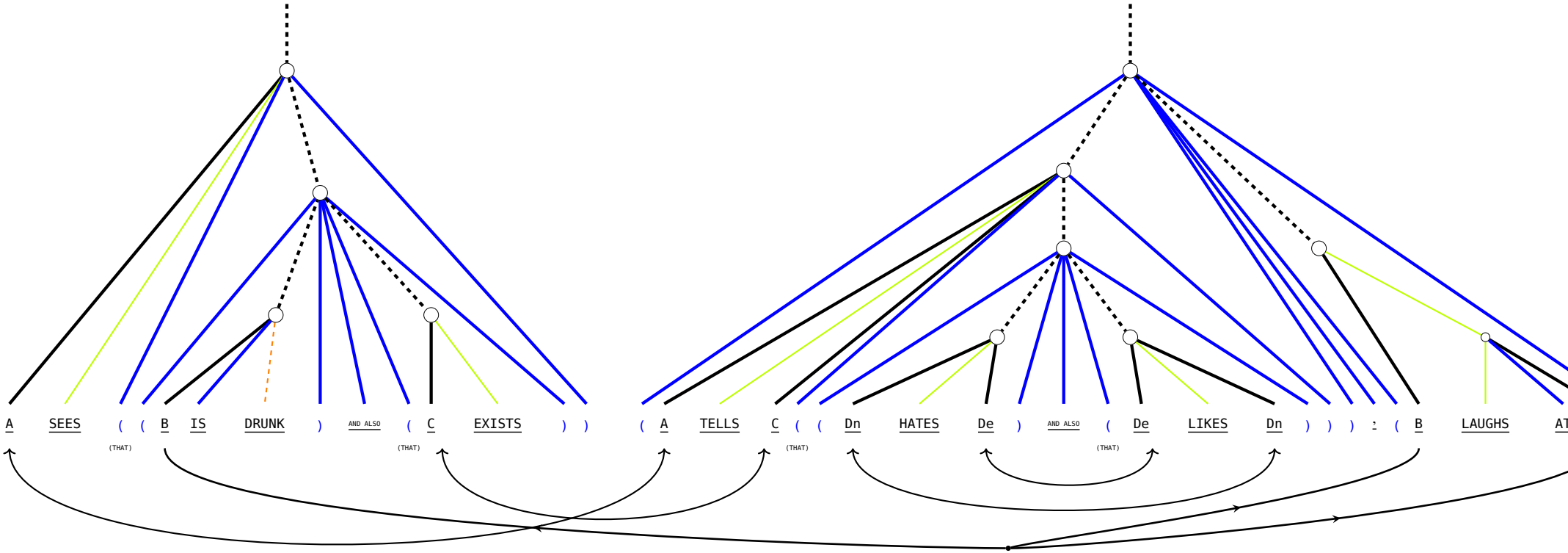




Then we join the diagrams of each slice with pronominal links that mirror the connectivity of the wire twisting slices:



The structure of pronominal links in text coincides with the compositional structure of circuits. By the rules of Section 3.3 relating text diagrams to hybrid grammar, and by the interpretation rules of Refinement 3.5.4, we obtain a hybrid grammar text:



ALICE SEES THAT BOB IS DRUNK AND ALSO THAT CLAIRE EXISTS.

ALICE TELLS CLAIRE THAT DENNIS HATES DEE AND ALSO THAT DEE LIKES DENNIS, BOB LAUGHS AT BOB.

By construction, our rewrite rules will turn the text back into our starting circuit. □

Example 3.5.16. In Figure 3.31 we give the example of textualisation of a text circuit that appeared in all of our previous such illustrations.

3.5.5 Extending grammar by means of equations

An immediate consequence of the Text Circuit Theorem is the existence of a nontrivial equivalence relation on parsed text. In prose: we consider two texts equivalent if they result in equal text circuits. So we may define equivalence of texts $\mathcal{T}_1 \equiv \mathcal{T}_2$ as follows:

$$\left.\begin{array}{l}\text{exists rewrite } \mathcal{T}_1 \Rightarrow \mathcal{C}_1 \\ \text{exists rewrite } \mathcal{T}_2 \Rightarrow \mathcal{C}_2\end{array}\right\} \text{ such that } \mathcal{C}_1 =_{\mathfrak{C}} \mathcal{C}_2$$

where $\mathcal{T} \Rightarrow \mathcal{C}$ denotes a rewrite of text with grammar $\mathcal{T}$ to text circuit $\mathcal{C}$, and $\mathcal{C}_1 =_{\mathfrak{C}} \mathcal{C}_2$ denotes equality of text circuits – cf. Convention 3.4.1.

We have seen so far how text circuits appear to capture the essential connectivity of meaning underneath the bureaucracy of natural language. By this consideration, the equivalence relation on text we have defined looks a lot like 'meaning equivalence'. Now we turn this observation into a guiding assumption.

Thesis 3.5.17 (Text Circuit Thesis)**.** Equal circuits stand for equal text meanings.

The thesis provides a recipe for engineering extensions of our grammar to accommodate grammatical phenomena beyond what we have considered here. We will explicitly demonstrate the inclusion of passive voice, possessive pronouns, and adjectivalisation of verbs using [ING] below.

When it is broadly agreed that $\mathcal{T}_1$ and $\mathcal{T}_2$ 'mean the same thing', then we will postulate that to be the case, and from this reverse engineer what the pieces of the text diagrams and corresponding rewrite rules of the grammar should be. If $\mathcal{T}_1$ makes use of a grammatical phenomenon $\mathcal{GP}$ beyond what we have considered thus far, while $\mathcal{T}_2$ is within our diagrams/grammar, then we make the circuits of $\mathcal{T}_1$ and $\mathcal{T}_2$ equal by adding the pieces of the text diagrams and corresponding rewrite rules for $\mathcal{GP}$. This procedure is best illustrated by the following examples, the first three of which are explored in [7].[8]

Example 3.5.18 (Passive voice)**.** In English, expressing a transitive verb in passive voice reverses the order of subject and object. For example, the following sentence in passive voice:

BOB IS LIKED BY ALICE

conveys the same factual data as the following sentence in active voice:

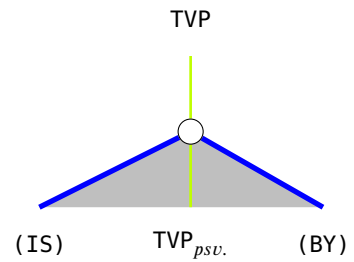
ALICE LIKES BOB

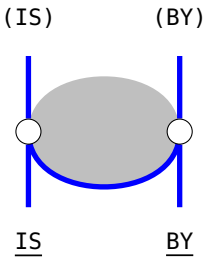
[7]

[8] In rather than extending diagrams/grammar by postulating equations like we do here, we used grammatical structure and so-called 'internal wirings' to derive equations between sentences. This was in fact the path we took in order to arrive at the results that we present here, and therefore the amount of credit we give here to those internal wirings. A forthcoming paper will be dedicated to this broad topic of 'internal wirings' and their relationship to the results presented here , and we also briefly address them in Section **??**.

; ; ; ; ; ; ; ; and

To accommodate the passive voice, we introduce a new wire type $\mathsf{TVP}_{psv.}$. In the figure below, on the left we introduce a string diagram that allows a TVP to become a $\mathsf{TVP}_{psv.}$ within a phrase scope, and on the right, we introduce a string diagram that closes the passive voice phrase scope by labelling with IS and BY:





We also ask that every label:

has a passive voice counterpart:

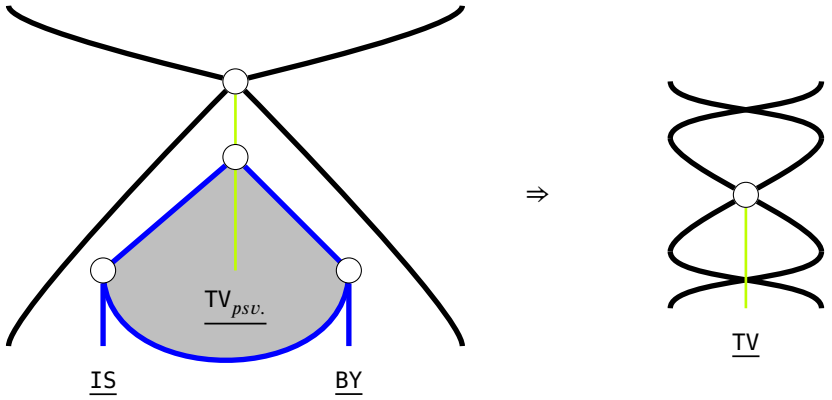
e.g. LIKES becomes LIKED. We also stipulate that, apart from the diagram on the left above, $\mathsf{TVP}_{psv.}$ mimics TVP for all other diagrams. This is so that we may further generate text grammar/diagrams within the passive voice scope to obtain, e.g.:

BOB IS DEEPLY LIKED BY ALICE

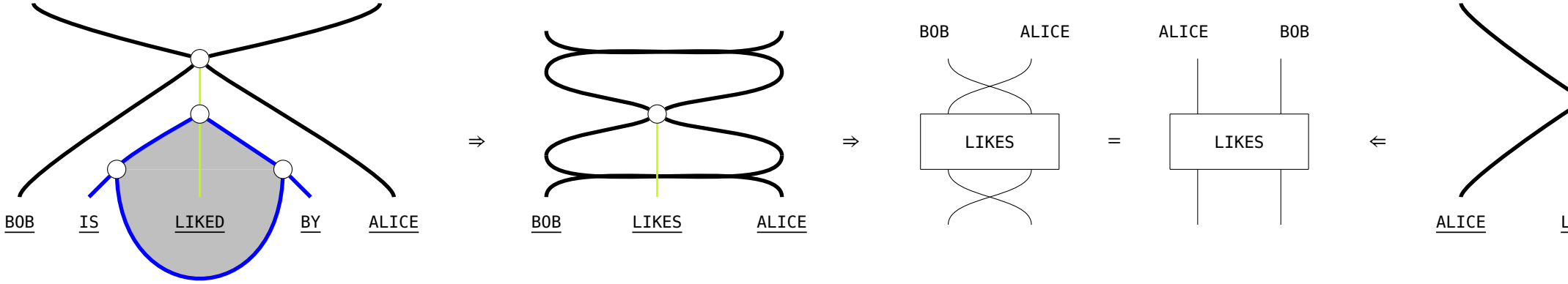
As a point of justification, the exception for the left diagram and having a separate $\mathsf{TVP}_{psv.}$ is necessary to disallow the passive construction to be applied repeatedly, which for example overgenerates the **ungrammatical**:

BOB IS IS LIKED BY BY ALICE

Returning to the example, we reformulate instances of passive voice into active voice by the following rewrite rule:



So we obtain the following proof of text equivalence:



Example 3.5.19 (Possessive Pronouns)**.** Consider the possessive pronoun HIS in the phrase:

BOB DRINKS HIS BEER

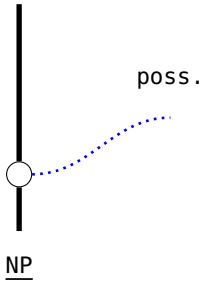
We can reformulate the noun phrase HIS BEER as the noun phrase BEER THAT HE OWNS without changing the core meaning. Applying this transformation, we obtain:

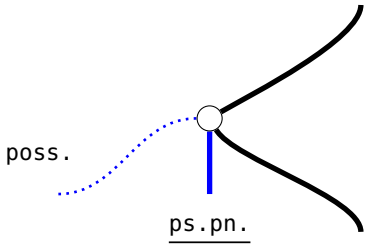
BOB DRINKS BEER THAT HE OWNS

We know that the relative pronoun THAT in this case gives us the same circuit as the text:

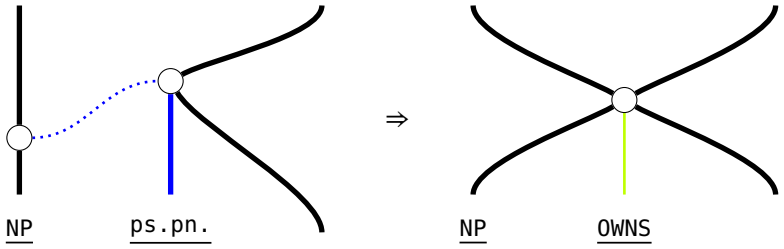
BOB DRINKS BEER. BOB OWNS BEER.

We can make this text equivalent to the original BOB DRINKS HIS BEER by introducing the following generation and rewrite rules for possessive pronoun labels. We introduce a dotted-blue possessive pronominal link type with two generators, so that the possessor is pronominally referred to by the possessive pronoun generator:

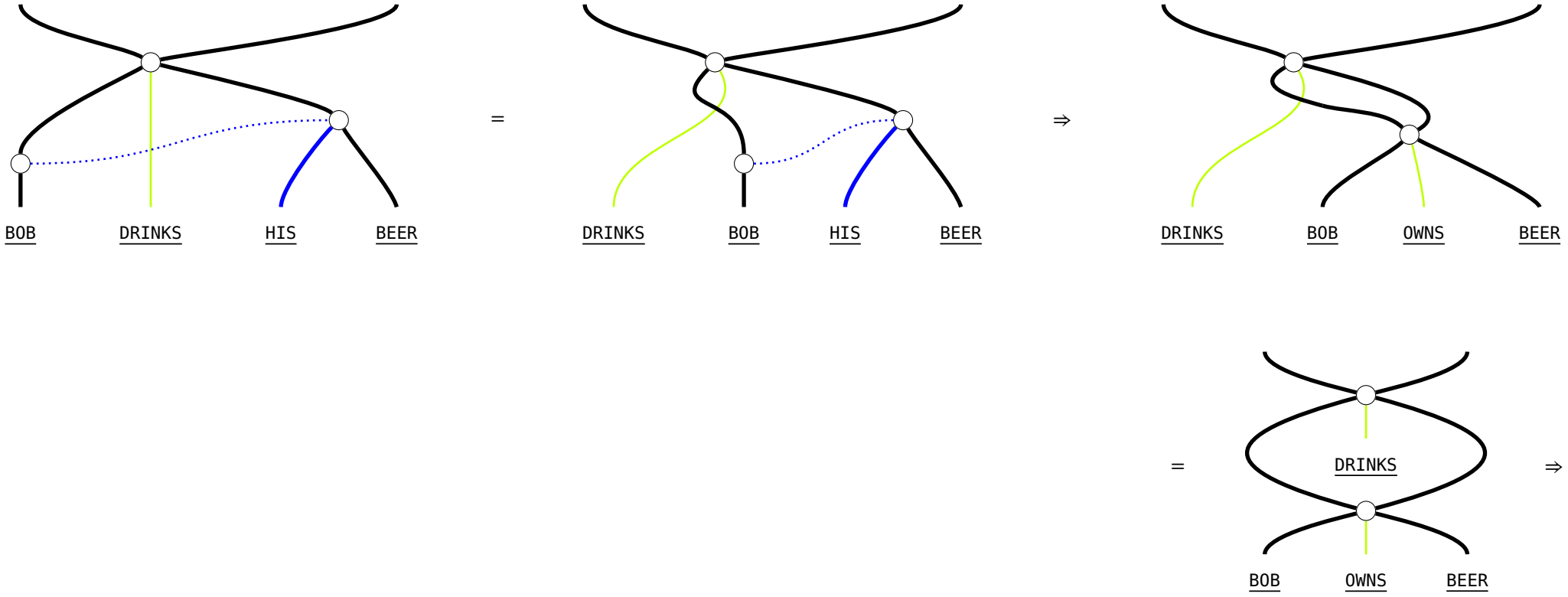




The rewrite rule that eliminates the possessive pronoun and the possessive pronominal is as follows:



So, returning to our sentence, we have a series of rewrites from text diagram to text circuit:



Note that while text diagrams from text are planar, in the process of rewrites, we may freely twist wires.

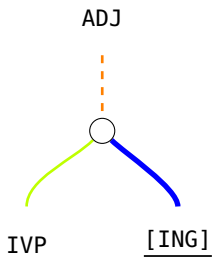
Example 3.5.20 (Adjectivalisation of Verbs by gerund `-ING`)**.** Appending `-ING` to a verb allows that verb to be used as if it were an adjective. For example, the noun phrase:

`DANC-ING ALICE`

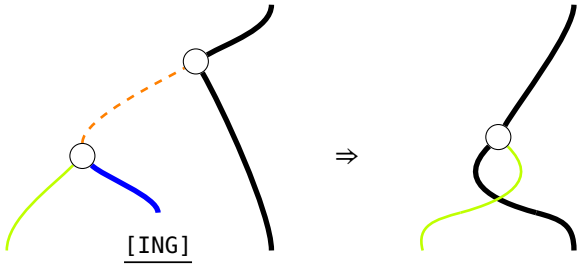
may be reformulated as the noun phrase:

`ALICE WHO DANCES`

We model [ING] as a generator that transforms an ADJ into a IVP.



The reduction rule is straightforward:



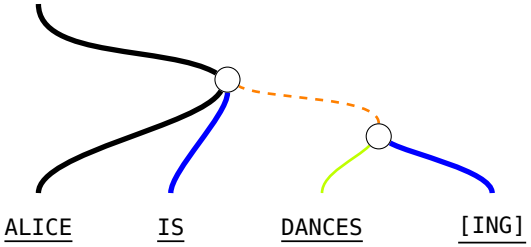
As a result, we obtain equivalences between the sentences:

- ALICE IS DANCING.
- ALICE DANCES.

and the noun-phrases:

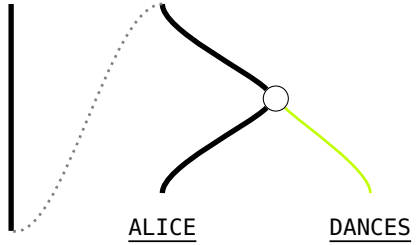
- DANCING ALICE
- ALICE WHO DANCES

ALICE IS DANCING.



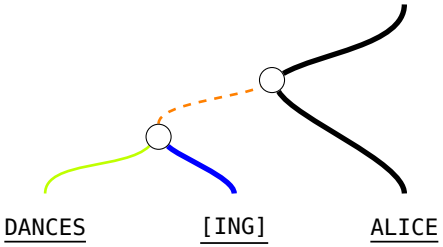
(IS elim.) ⇓

ALICE WHO DANCES



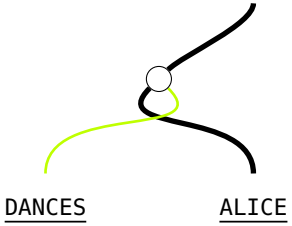
(pron. elim.) ⇓

DANCING ALICE



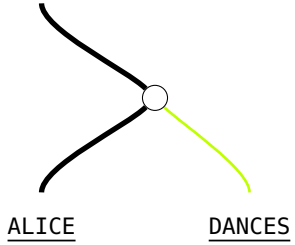
([ING] elim.)

⇒



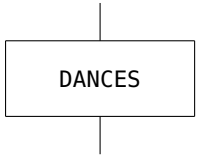
=

ALICE DANCES.



⇓

ALICE



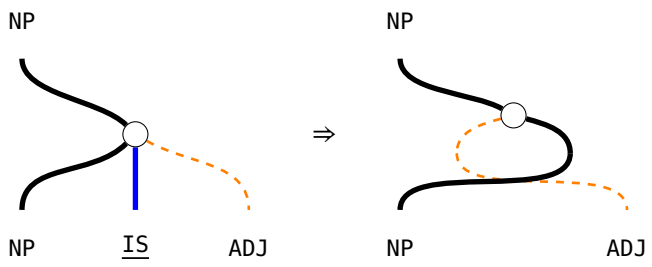
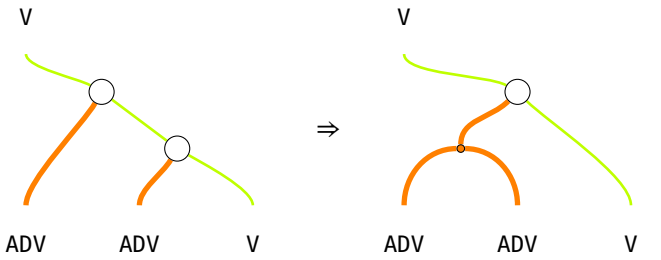
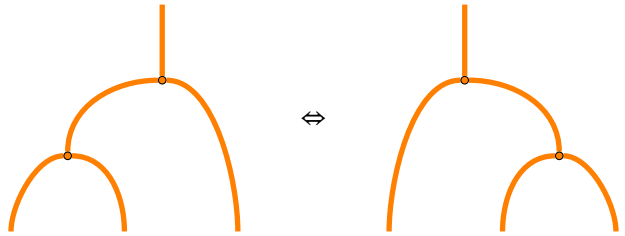
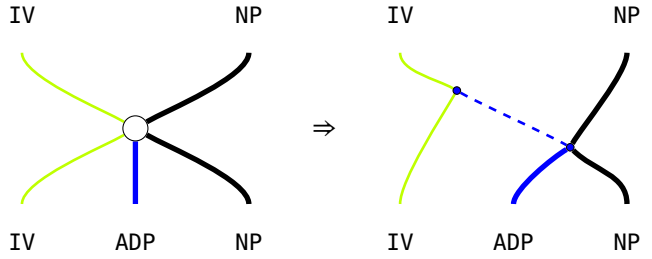
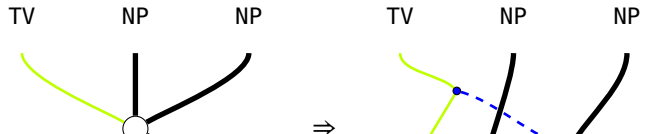
Rule name	Text diagram rewrite
IS-elimination	
ADV-gather	
ADV-assoc.	
ADP(IV)-ancilla	
	

Figure 3.29: Gate normalisation rewrites

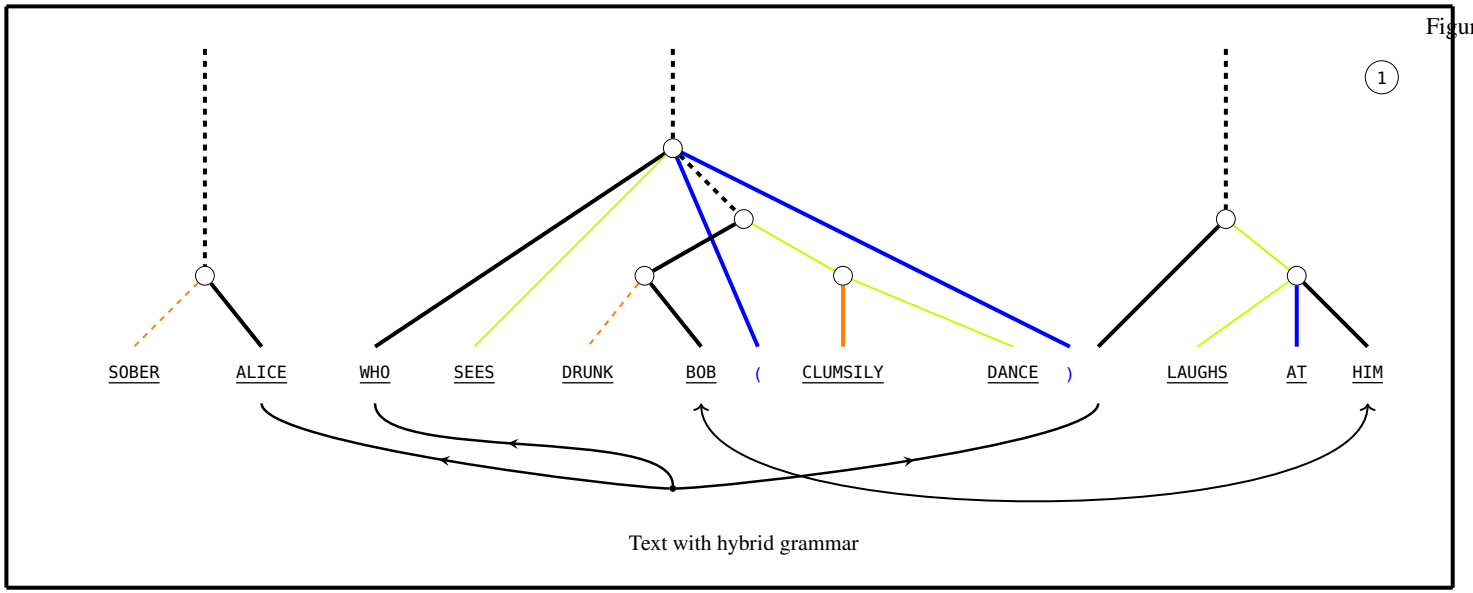
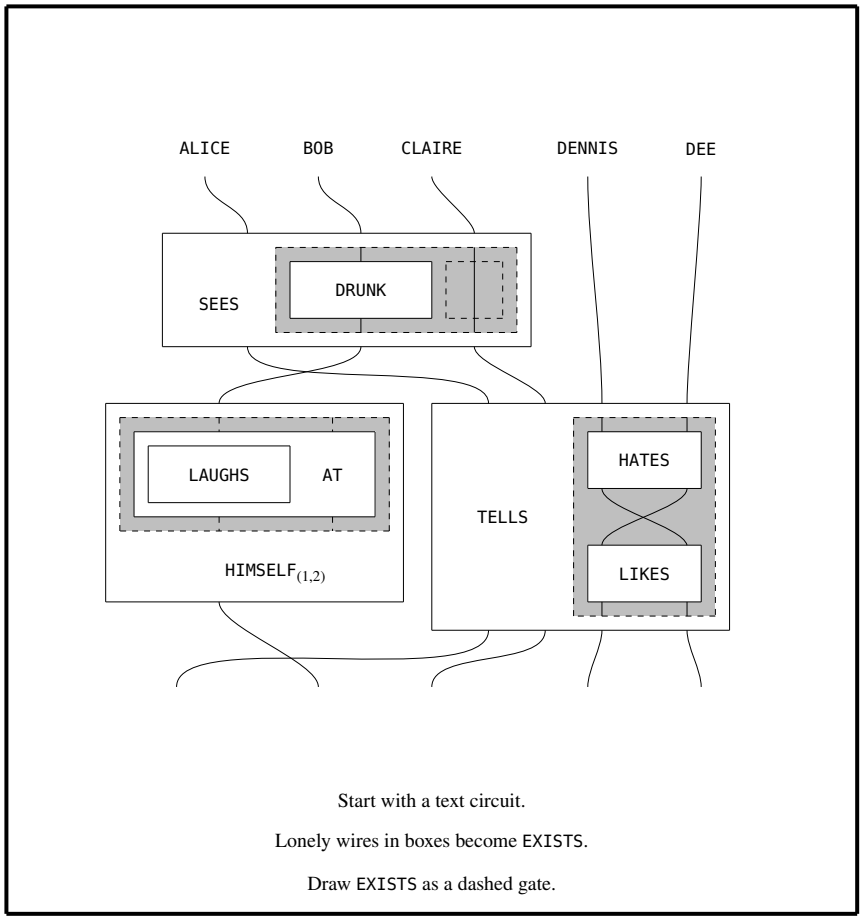


Figure 3.30: Obtaining text circuits from text diagrams.

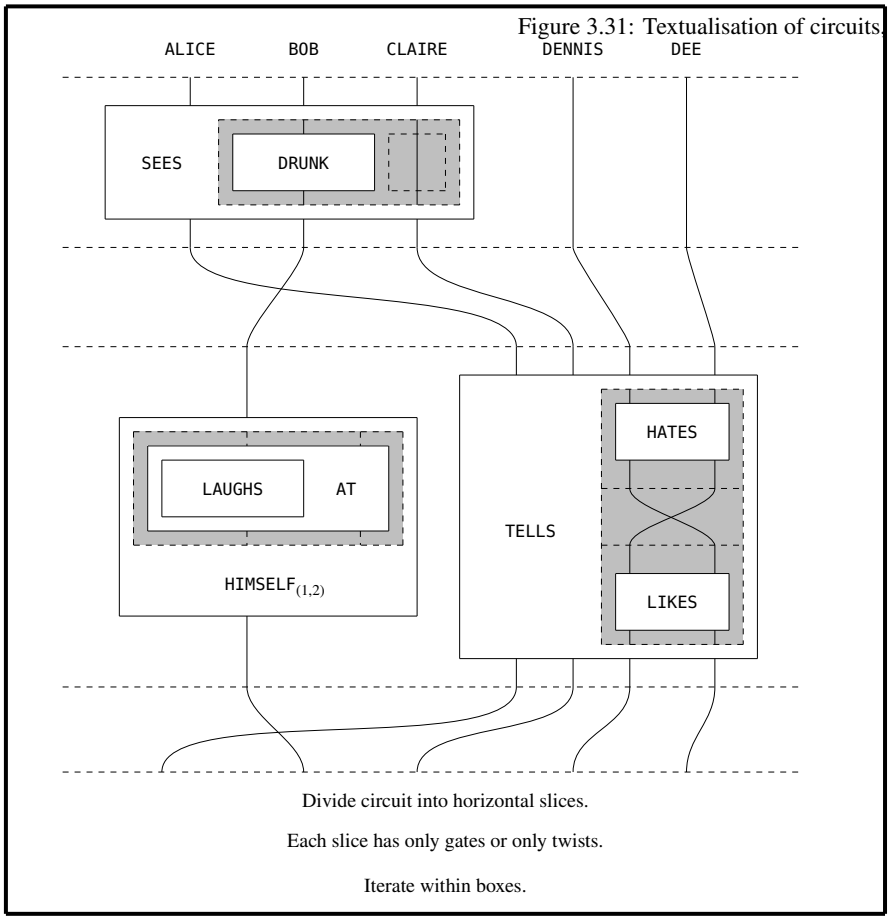
Figure 3.31: Textualisation of circuits, illustrated.



Start with a text circuit.

Lonely wires in boxes become EXISTS.

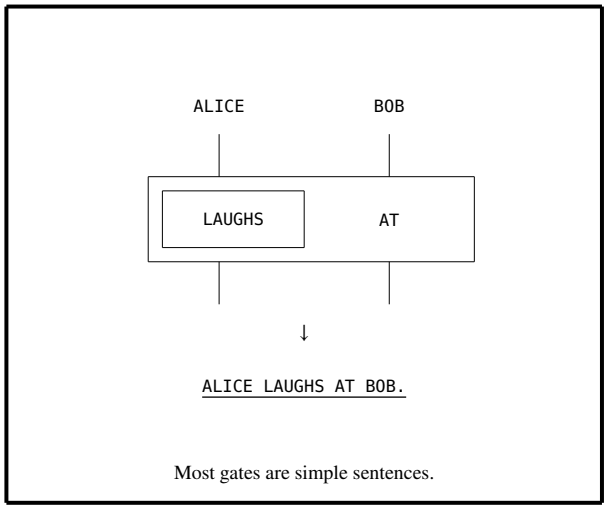
Draw EXISTS as a dashed gate.



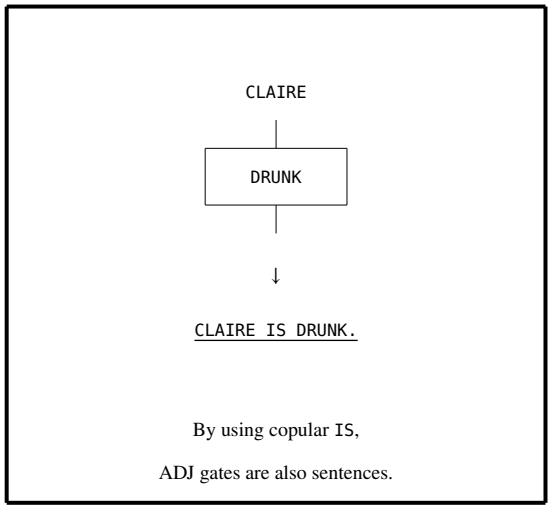
Divide circuit into horizontal slices.

Each slice has only gates or only twists.

Iterate within boxes.

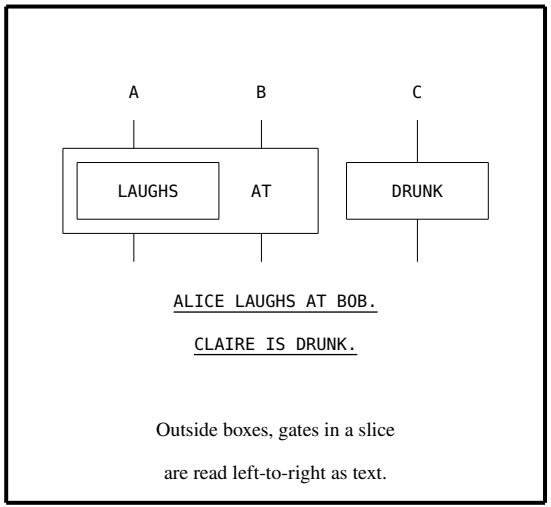


Most gates are simple sentences.



By using copular IS,

ADJ gates are also sentences.



Outside boxes, gates in a slice

are read left-to-right as text.

4

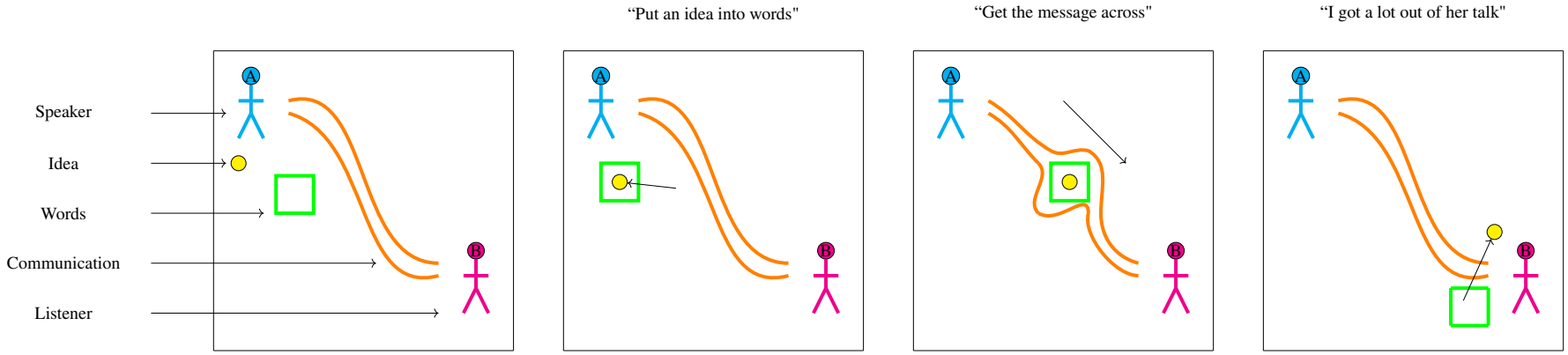
Continuous relations: a palette for toy models

4.1 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 4.5.5) 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 4.5.8 which provides a diagrammatic characterisation result that allows us to precisely identify when any idempotent in the category splits though a discrete topology. The discrete topology thus acts as a set of labels, where the (pre)images of each element under section and retract behave as the kind of shapes we wish to consider. As an interlude, we demonstrate how we may construct families of such idempotent-coherent special

commutative frobenius algebras on $\mathbb{R}^2$ to provide a monoidal generalisation (c.f. the monoidal computer framework in []) of **FinRel** equipped with a Turing object [], thus satisfying Justification **??**. Then we proceed to define configuration spaces of collections of shapes up to rigid displacement, and we develop a relational analogue of homotopy to model motions as paths in configuration space, along with appropriate extensions to accommodate nonrigid phenomena. If you are still reading and not already a category theorist nor just curious, I will drop the jargon now, because you are probably either questioning or deeply committed to your false ideals, both of which I respect, but just to spite the latter, I will proceed to do everything diagrammatically as much as possible.

4.1.1 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, because, as we have seen, this potentially allows whatever we construct in this setting to generalise to other process theories. Also, we want an excuse to build up and play with a symmetric monoidal category 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 concepts – however they are represented – and symbolic representation and manipulation? Here we sidestep the debate around what concepts actually *are*, aligning ourselves close to Gärdenfors: we assume that there are *conceptual spaces* that organise concepts of similar domain – such as colour, taste, motion – and that regions of these spaces correspond to concepts. Gärdenfors' stance is backed by empirical data [], but even if he is wrong, he is at least interestingly so for our purposes. As an example, the classic example of colourspace is also one of the best studied and implemented: there are many different embeddings of the space of visible colours in Euclidean space []. In this setting we can mathematically model the action of categorising a particular point in colour space as `blue` by checking to see whether that point falls within the region in colourspace that the symbol `blue` is associated with. So we find that this view is conducive to modelling concepts as spatial entities – a very permissive and expressive framework, which will allow us to calculate interesting things – whilst also having the ability to handle them using symbolic labels. The question remains: how do we model this association between space and sets of labels? This leads us to the following consideration: In a category for concept-compliant text-circuits $\mathcal{T}$, any conceptual space object Γ should possess a split idempotent through a set of concept-labels, thus encoding the association between concepts-qua-spaces and concepts-qua-symbols.

A further complication arises. Once we have a stock of symbols referring to entities in space, we can start talking about pairs of entities (e.g. `red or blue`), subsets of sets of entities (e.g. `autumnal colours`), arbitrary relations between the set of entities in one space and entities of another (e.g. how the colour of a banana relates to its probable textures and tastes). Given enough time and patience, we can linguistically construct – at least – any finite relation between finite sets. As we will elaborate in Section 5, the real challenge is that every time we define a new concept in this way, we can again treat it as a symbolic concept, that is, label it with a noun. Since nouns are first-class citizens that travel along wires in our framework, we are asking that any putative process theory that satisfies these considerations 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 following consideration: A category for 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 already know how to encode using split idempotents, so we can translate the two considerations of the last two paragraphs into one requirement. In prose; the first item gives us a method within the category $\mathcal{T}$ 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 and manipulate concepts however we would like to treat finite relations.

Requirement 4.1.1. 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:

Topology	**Type Theory**	**Conceptual Spaces**
Space	Type	Conceptual space
Point	Element of set	Copyable instance
Subset	Subset	Instance
Open set	Semi-decidable set	Concept
Closed set	Set with semi-decidable complement	-
Clopen set	Decidable set	A concept the negation of which is also a concept
Discrete topology	Type with decidable equality	A conceptual space where any collection of instances forms a concept
Haussdorf topology	Type with semi-decidable inequality of elements	A conceptual space where any pair of distinct instances can be described as belonging to two disjoint concepts
Compact set	Exhaustively searchable set, in a finite number of steps	A conceptual space $\mathfrak{C}$ such that for any joint concept R on $\mathfrak{C} \times \mathfrak{D}$, $\forall c_{\in \mathfrak{C}} R(c, -)$ is a concept in $\mathfrak{D}$

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 your concepts if your base space is discrete and finite.

This no-go just provides another perspective of the ancient observation that logical thought really does seem symbolic. It is also important to note that the essential idea of conceptual spaces is to circumvent this no-go; in our language, asking for split idempotents through (finite) discrete topologies is precisely what allows logical constructions to work on spatial representations of concepts, when those split idempotents exist. What this no-go does mean is that nobody has to waste their time looking for *precise* unifications of conceptual spaces and first-order logic-with-equality where every logical operator is viewed as a space-preserving transformation 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 reasoning. We want the objects of this category to be topological spaces because we want to model conceptual spaces and calculate interesting things. While the category **Top** of topological spaces and continuous functions is already symmetric monoidal with respect to categorical product, for linguistic and concept-related considerations we want finite relations to come into play diagrammatically, and because **Top** is cartesian monoidal it doesn't work for our purposes. So we construct **ContRel**.

4.2 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 4.2.4 (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 4.2.5. 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.

Reminder 4.2.1 (Topological Space)**.** A *topological space* is a pair (X, τ), where X is a set, and $\tau \subset \mathcal{P}(X)$ are the *open sets* of X, such that:

"nothing" and "everything" are open

$$\emptyset, X \in \tau$$

Arbitrary unions of opens are open

$$\{U_i : i \in I\} \subseteq \tau \Rightarrow \bigcup_{i \in I} U_i \in \tau$$

Finite intersections of opens are open $n \in \mathbb{N}$:

$$U_1, \cdots, U_n \in \tau \Rightarrow \bigcap_{1 \cdots, i, \cdots n} U_i \in \tau$$

Reminder 4.2.2 (Relational Converse)**.** Recall that a relation $R : S \to T$ is a subset $R \subseteq S \times T$.

$$R^\dagger : T \to S := \{(t, s) : (s, t) \in R\}$$

Reminder 4.2.3 (Continuous function)**.** A function between sets $f : X \to Y$ is a continuous function between topologies $f : (X, \tau) \to (Y, \sigma)$ if

$$U \in \sigma \Rightarrow f^{-1}(U) \in \tau$$

where f^{-1} denotes the inverse image.

Reminder 4.2.6 (Product Topology)**.** We denote the product topology of X^τ and Y^σ as $(X \times Y)^{(\tau \times \sigma)}$. $\tau \times \sigma$ is the topology on $X \times Y$ generated by the basis $\{t \times s : t \in \mathfrak{b}_\tau, s \in \mathfrak{b}_\sigma\}$, where $\mathfrak{b}_\tau$ and $\mathfrak{b}_\sigma$ are bases for τ and σ respectively.

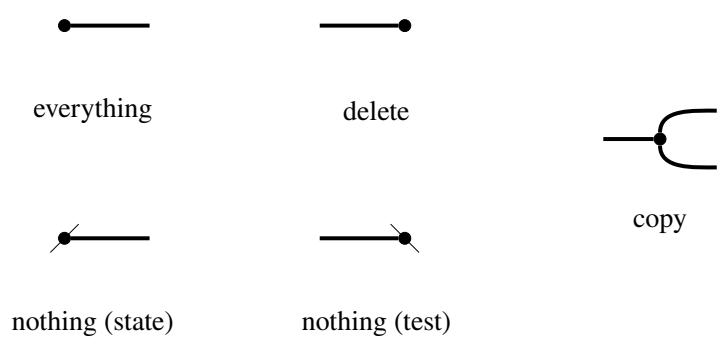
Reminder 4.2.7 (Product of relations)**.** For relations between sets $R : X \to Y, S : A \to B$, the product relation $R \times S : X \times A \to Y \times B$ is defined to be

$$\{((x, a), (y, b)) : (x, y) \in R, (a, b) \in S\}$$

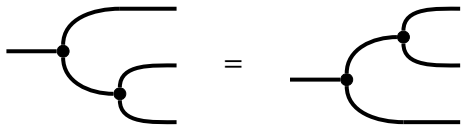
*4.3 **ContRel** diagrammatically*

4.3.1 Relations that are always continuous

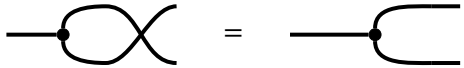
HERE ARE FIVE CONTINUOUS RELATIONS FOR ANY X^τ:



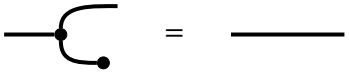
COPY AND DELETE OBEY THE FOLLOWING EQUALITIES:



coassociativity

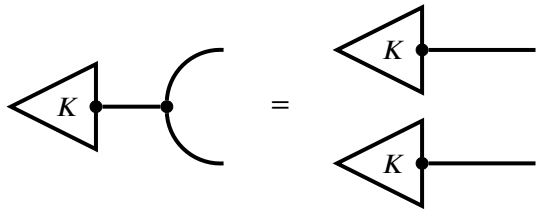


cocommutativity

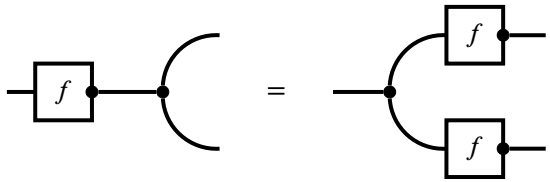


counitality

THE COPY MAP CAN ALSO BE USED TO DISTINGUISH THE DETERMINISTIC MAPS – POINTS AND FUNCTIONS – WHICH WE NOTATE WITH AN EXTRA DOT.

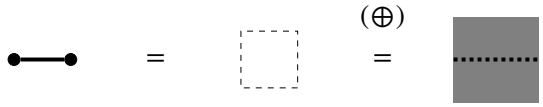


point

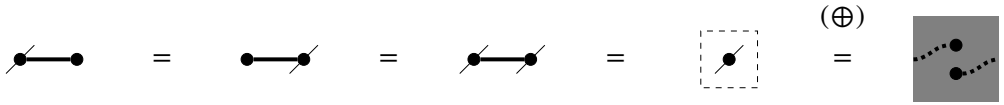


function

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 $\oplus$.

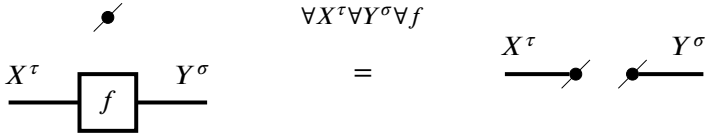


(b)one



zero

ZERO SCALARS TURN ENTIRE DIAGRAMS INTO ZERO MORPHISMS. There is a zero-morphism for every input-output pair of objects in **ContRel**.


X^τ
f
Y^σ
$\forall X^\tau \forall Y^\sigma \forall f$
=
X^τ
Y^σ

zero

4.4 Continuous Relations by examples

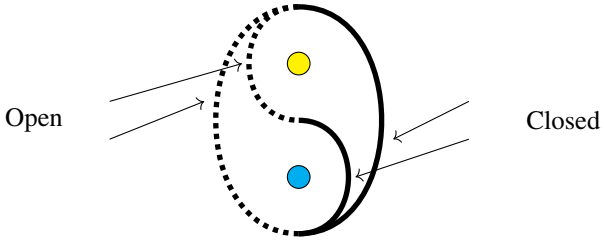
Let's consider three topological spaces and examine the continuous relations between them. This way we can build up intuitions, and prove some tool results in the process.

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

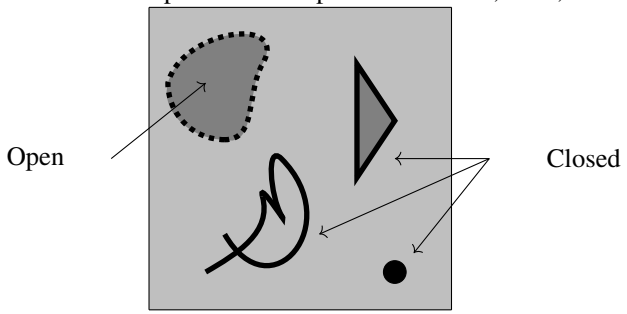
$$(\{\star\}\ ,\ \{\{\star\}, \varnothing\})$$

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 4.4.1, not all of

Example 4.4.1 (A noncontinuous relation)**.** The relation $\{(0, \bullet)\} \subset \mathcal{S} \times \bullet$ is not a continuous relation: the preimage of the open set $\{\bullet\}$ under this relation is the non-open set $\{0\}$.

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 4.4.3 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 4.4.4 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 4.4.6. *Proof.* The preimage of the empty relation is always $\varnothing$, which is open by definition. □

FULL RELATIONS ARE ALWAYS CONTINUOUS

Proposition 4.4.8. *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 4.4.9. 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 4.4.10. 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. □

Terminology 4.4.2. Call a continuous relation $\bullet \to X^\tau$ a **state** of X^τ, and a continuous relation $X^\tau \to \bullet$ a **test** of X^τ.

Proposition 4.4.3. States $R : \bullet \to X^\tau$ correspond with subsets of X.

Proof. The preimage $R^\dagger(U)$ of a (non-$\varnothing$) open $U \in \tau$ is $\star$ if $R(\star) \cap U$ is nonempty, and $\varnothing$ otherwise. Both $\star$ and $\varnothing$ are open in $\{\star\}^\bullet$. $R(\star)$ is free to specify any non-$\varnothing$ subset of X. The empty relation handles $\varnothing$ as an open of X^τ. □

Proposition 4.4.4. Tests $R : X^\tau \to \bullet$ correspond with open sets $U \in \tau$.

Proof. The preimage $R^\dagger(\star)$ of $\star$ must be an open set of X^τ by definition **??**. $R^\dagger(\star)$ is free to specify any open set of X^τ. □

Reminder 4.4.5 (Empty relation)**.** The **empty relation** $X \to Y$ relates nothing. It is defined:

$$\varnothing \subset X \times Y$$

Reminder 4.4.7 (Full relation)**.** The **full relation** $X \to Y$ relates everything to everything. It is all of $X \times Y$.

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 4.4.10.

Proposition 4.4.12. *Proof.* The preimage of any open set under the identity relation is itself, which is open by assumption. □

Reminder 4.4.11 (Identity relation)**.** The **identity relation** $X \to X$ relates anything to itself. It is defined:

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

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

Proposition 4.4.14. 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. □

Reminder 4.4.13 (Union, intersection, and ordering of relations)**.** Recall that relations $X \to Y$ can be viewed as subsets of $X \times Y$. So it makes sense to speak of the union and intersection of relations, and of partially ordering them by inclusion.

Corollary 4.4.15. 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 4.4.17 (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 4.4.18 (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 4.4.19 (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 4.4.20. Continuous partial functions form a topological basis for the space $(X \times Y)^{(\tau \multimap \sigma)}$, where the opens are continuous relations $X^\tau \to Y^\sigma$.

Reminder 4.4.16 (Topological Basis)**.** $\mathfrak{b} \subseteq \tau$ is a basis of the topology τ if every $U \in \tau$ is expressible as a union of elements of $\mathfrak{b}$. Every topology has a basis (itself). Minimal bases are not necessarily unique.

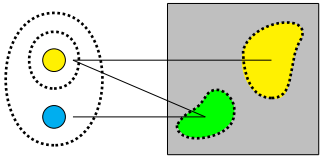


Figure 4.1: Regions of ■ in the image of the yellow point alone will be coloured yellow, and regions in the image of both yellow and cyan will be coloured green:

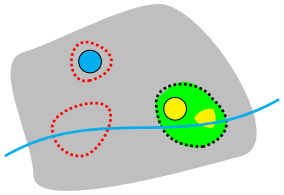


Figure 4.2: Regions in the image of the cyan point alone cannot be open sets by continuity, so they are either points or lines. Points and lines in cyan must be surrounded by an open region in either yellow or green, or else we violate continuity (open sets in red).

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 4.4.19. By Lemma 4.4.18, every $R \cap F$ term is a partial function, and by Corollary 4.4.15, continuous. □

$\mathcal{S} \to \mathcal{S}$: We can use Proposition 4.4.20 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 4.5.

$\mathcal{S} \to \blacksquare$: Now we use the colour convention of the points in $\mathcal{S}$ to "paint" continuous relations on the unit square "canvas", as in Figures 4.1 and 4.2. 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.

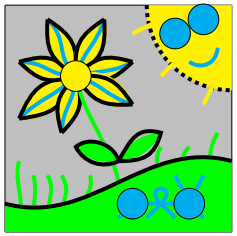


Figure 4.3: A continuous relation $\mathcal{S} \to \blacksquare$: "Flower and critter in a sunny field".

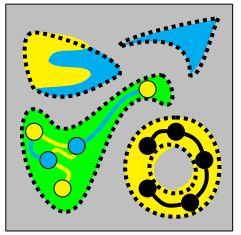


Figure 4.4: A continuous relation $\blacksquare \to \mathcal{S}$: "still math?". Black lines and dots indicate gaps.

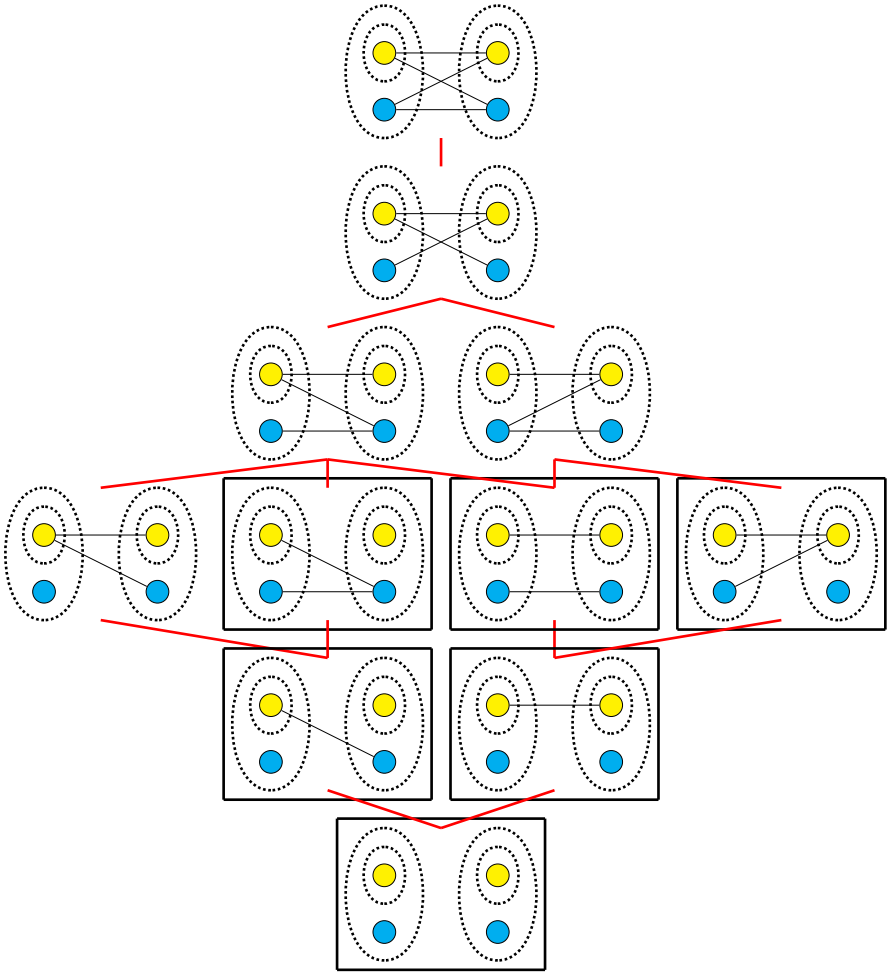


Figure 4.5: Hasse diagram of all continuous relations from the Sierpiński space to itself. Each relation is depicted left to right, and inclusion order is bottom-to-top. Relations that form the topological basis are boxed.

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.

Figure 4.6: continuous functions $[0,1] \to \blacksquare$ follow the naïve notion of continuity: a line one can draw on paper without lifting the pen off the page.

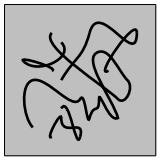


Figure 4.7: So a continuous partial function is `"(countably) many (open-ended) lines, each of which one can draw on paper without lifting the pen off the page."`

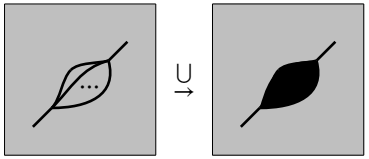


Figure 4.8: We can control the thickness of the brushstroke, by taking the union of (uncountably) many lines.

Figure 4.9: Assign the visible spectrum of light to $[0,1]$. Colour open sets according to perceptual addition of light, computing brightness by normalising the measure of the open set.

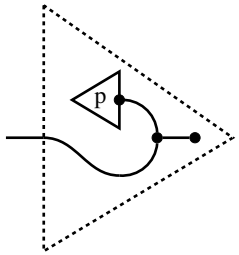
4.5 *Populating space with shapes using sticky spiders*

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

Example 4.5.2 (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 S with itself.

Proposition 4.5.3. 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.

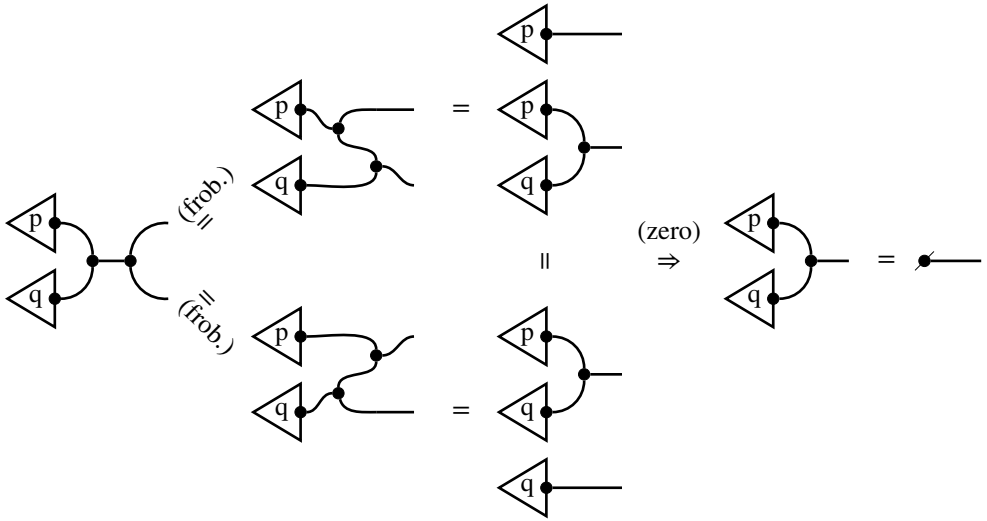


Figure 4.10: Like it or not, a continuous relation $[0, 1] \to \blacksquare$: "The Starry Night", by Vincent van Gogh.

Reminder 4.5.1 (copy-compare spiders of **Rel**)**.** For a set X, the *copy* map $X \to X \times X$ is defined:

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

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

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

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

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

and *everything*:

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

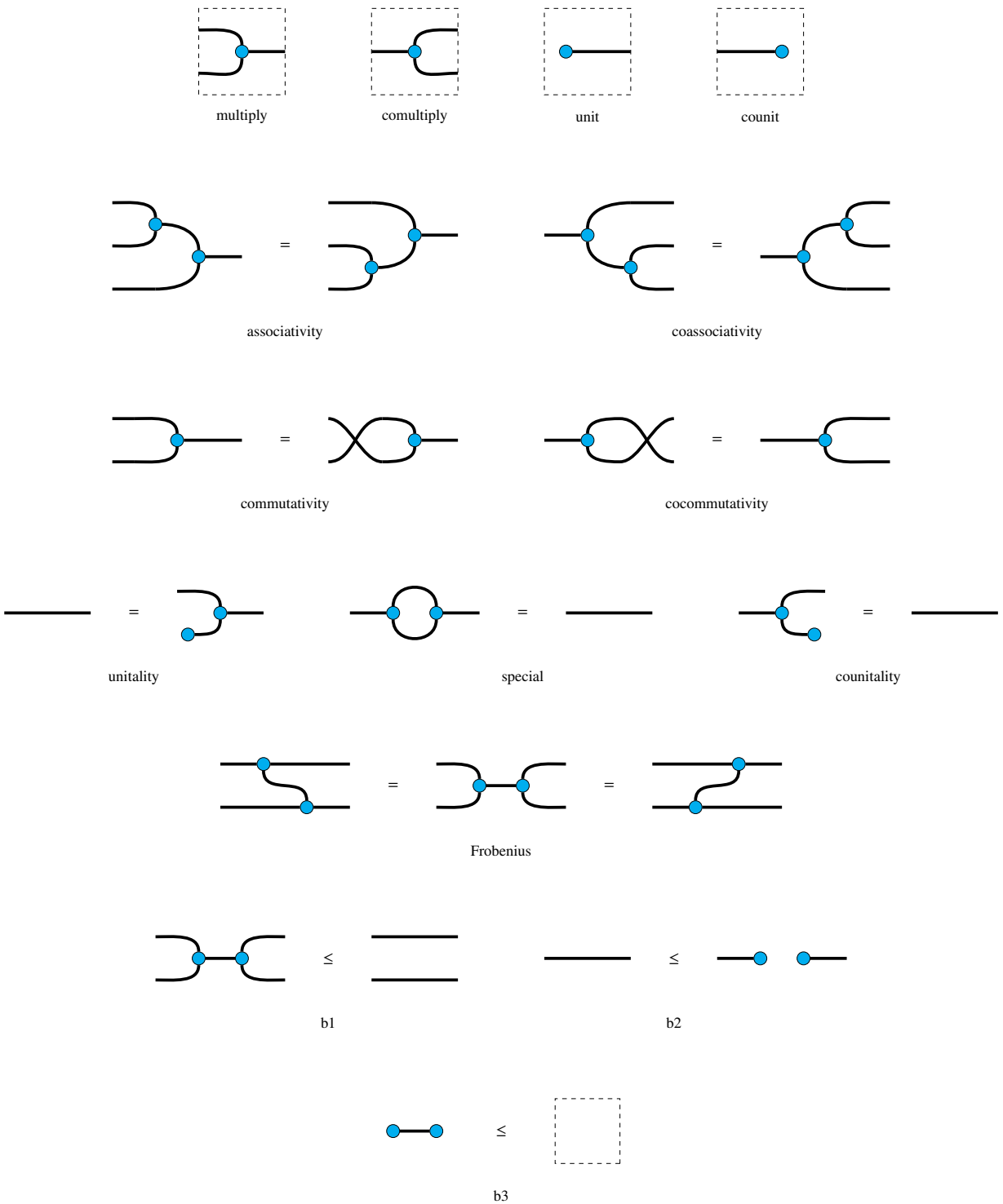
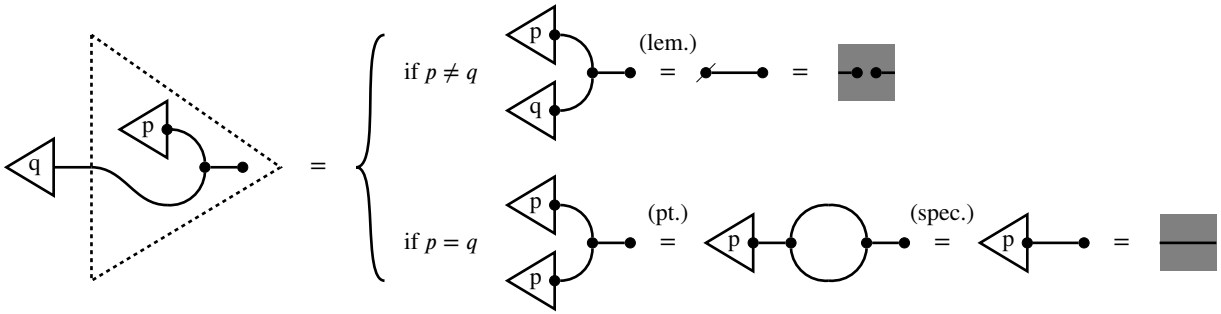


Figure 4.11: The generators (in dashed boxes) and relations that make a spider. When the spider satisfies in addition the three inequalities b1-3, we call it a **relation-spider**.

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.

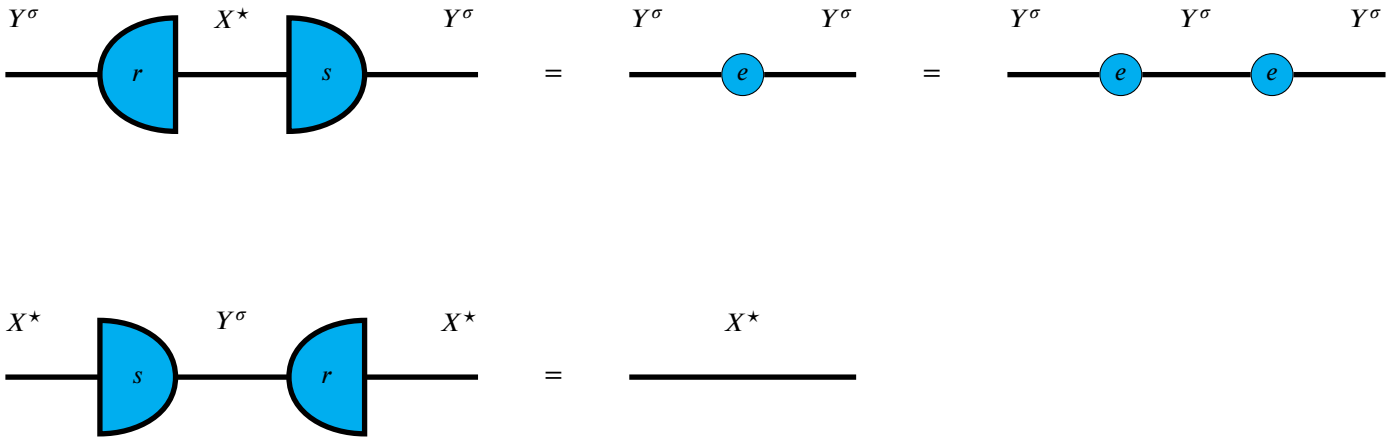
Reminder 4.5.4 (Split idempotents)**.** An **idempotent** in a category is a map $e : A \to A$ such that

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

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

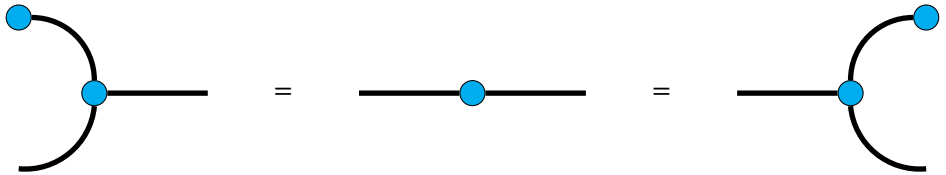
$$A \overset{e}{\to} A \equiv A \overset{r}{\to} B \overset{s}{\to} A$$
$$B \overset{s}{\to} A \overset{r}{\to} B = B \overset{id}{\to} B$$

WE CAN 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.

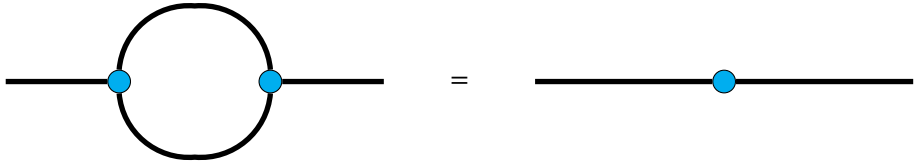


Definition 4.5.5 (Sticky spiders)**.** A **sticky spider** (or just an e-spider, if we know that e is a split idempotent), is a spider *except* every identity wire on any side of an equation in Figure 4.11

is replaced by the idempotent e.



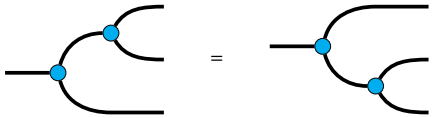
e-(co)unitality



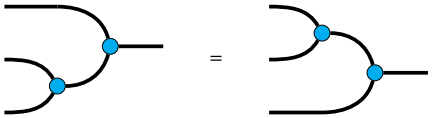
e-special

The desired graphical behaviour of a sticky spider is that one can still coalesce all connected spider-bodies together, but the 1-1 spider "sticks around" rather than disappearing as the identity. This is achieved by the following rules that cohere the idempotent e with the (co)unit and (co)multiplications; they are the same as the usual rules for a special commutative frobenius algebra with two exceptions. First, where an identity wire appears in an equation, we replace it with an idempotent. Second, the monoid and comonoid components 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.

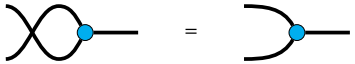
Coassociative



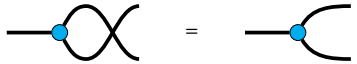
Associative



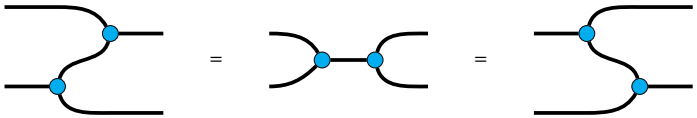
Commutative



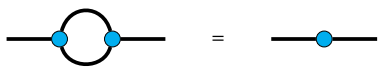
Cocommutative



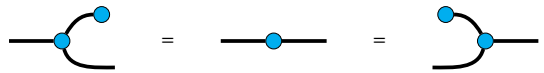
Frobenius



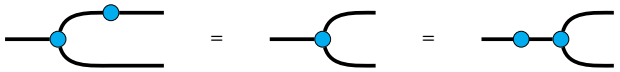
Sticky-special



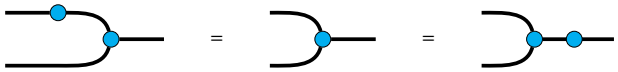
Sticky-(co)unital



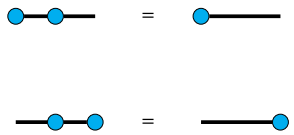
Absorb-comult



Absorb-mult

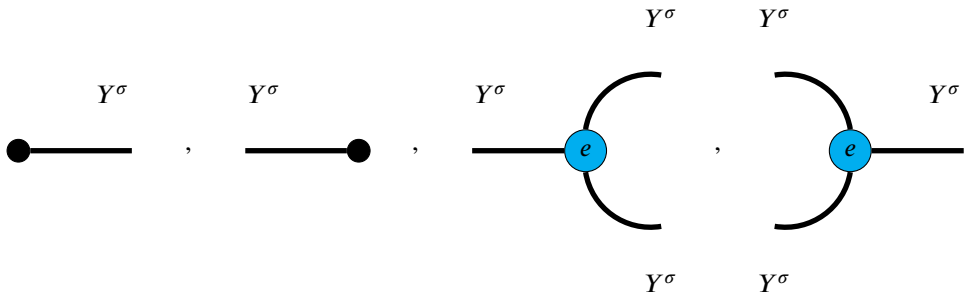


Absorb-(co)unit



Construction 4.5.6 (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:

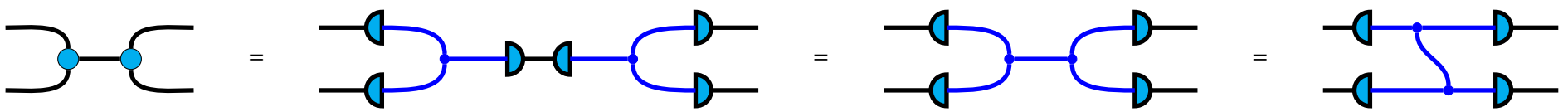
Proposition 4.5.7 (Every idempotent that splits through a discrete topology gives a sticky spider)**.**



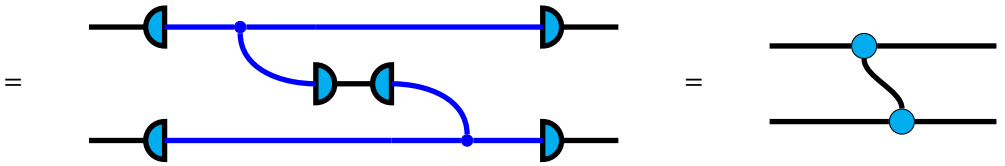
is a sticky spider

Proof. We can check that our construction satisfies the frobenius rules as follows. We only present one equality; the rest follow the same idea.

(defn.) (idem.) (frob.)



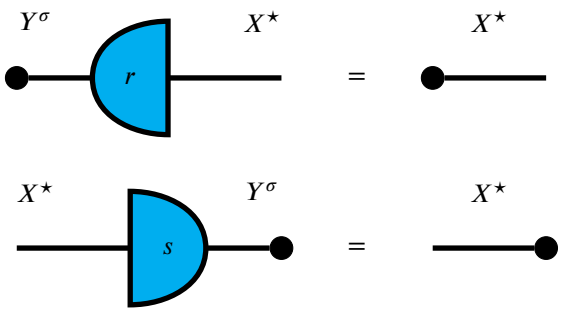
(split) (defn.)



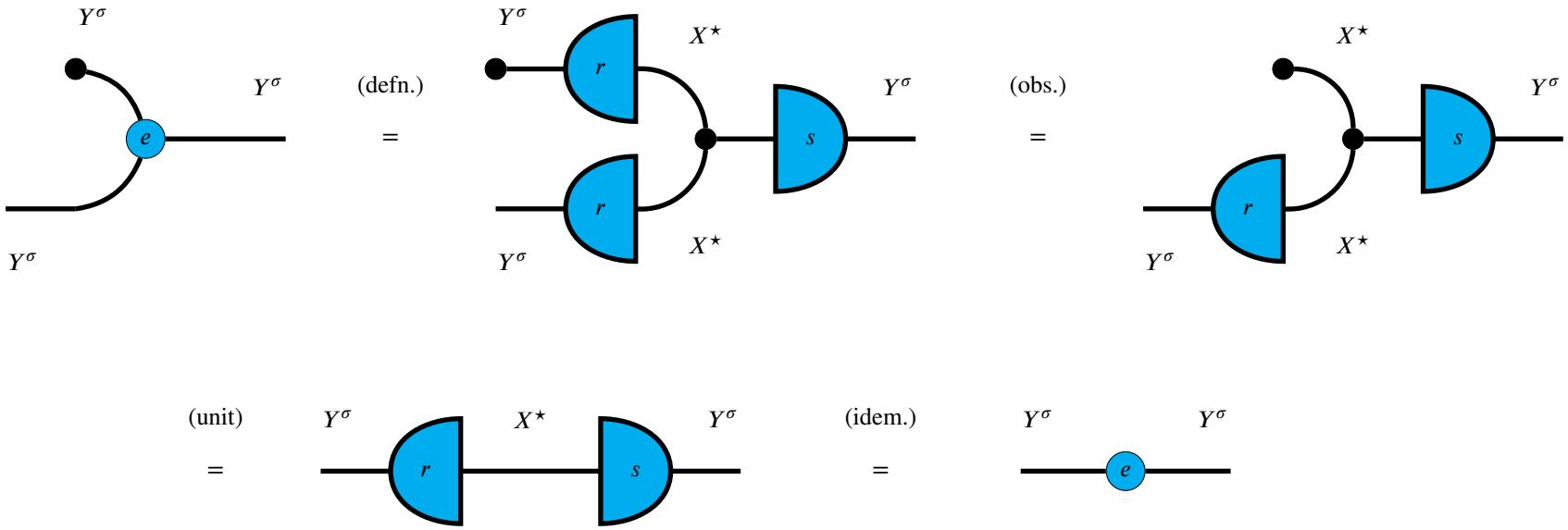
To verify the sticky spider rules, we first observe that since

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

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:

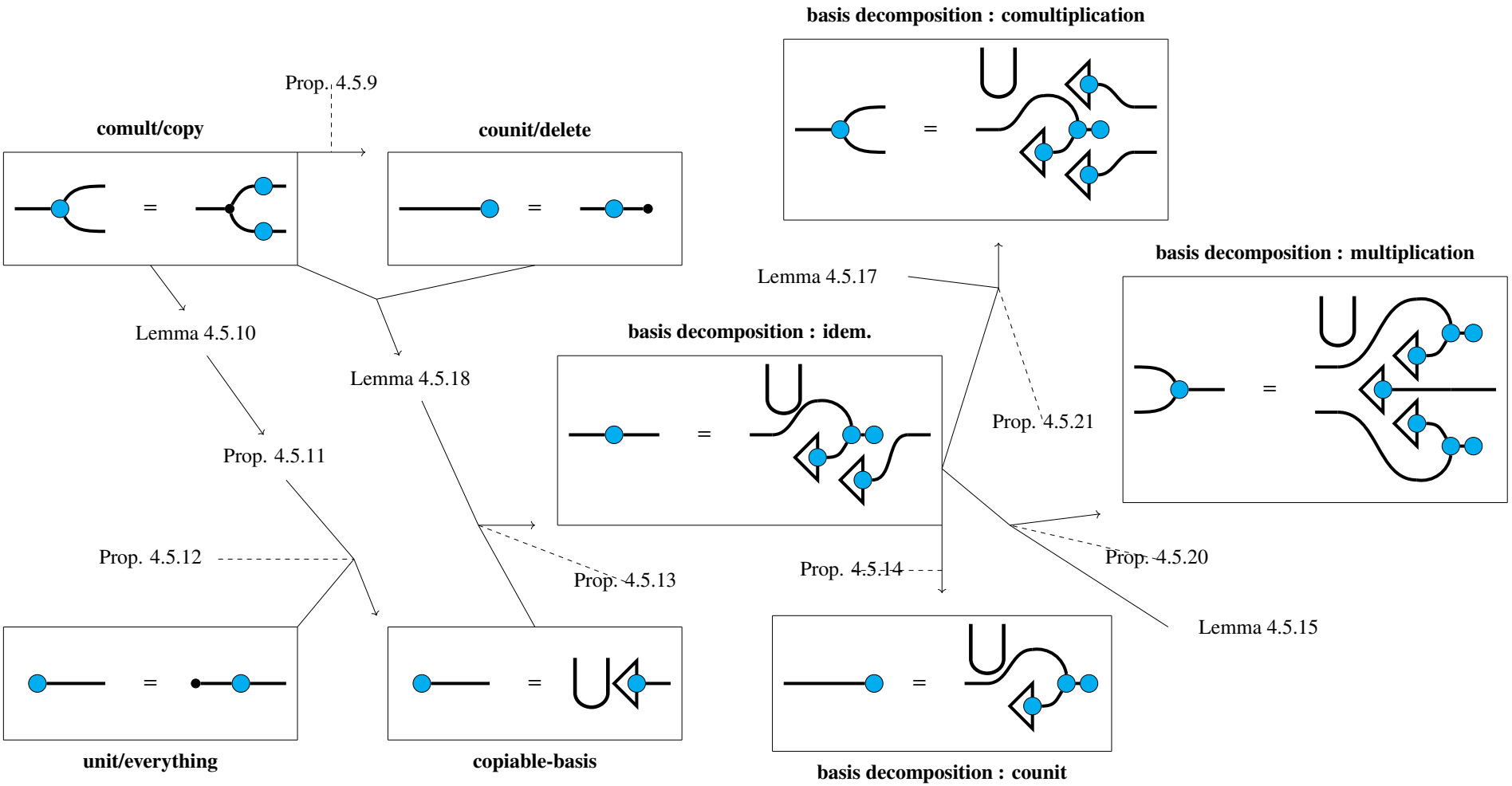


The proofs of e-counitality, and e-speciality follow similarly. □

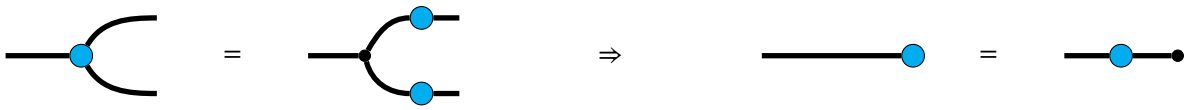
WE CAN PROVE A PARTIAL CONVERSE OF PROPOSITION 4.5.7: we can identify two diagrammatic equations that tell us precisely when a sticky spider has an idempotent that splits though some discrete topology.

Theorem 4.5.8. 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 relations are also satisfied.

The proof is rather involved, so we provide a map below of the various lemmas and propositions that will yield the claim.

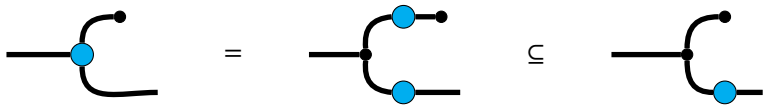

basis decomposition : comultiplication
Prop. 4.5.9
comult/copy
counit/delete
Lemma 4.5.17
basis decomposition : multiplication
Lemma 4.5.10
basis decomposition : idem.
Lemma 4.5.18
Prop. 4.5.21
Prop. 4.5.11
Prop. 4.5.12
Prop. 4.5.13
Prop. 4.5.14
Prop. 4.5.20
Lemma 4.5.15
unit/everything
copiable-basis
basis decomposition : counit

Proposition 4.5.9 (comult/copy implies counit/delete)**.**

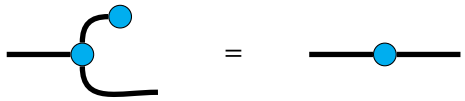


Proof.

(comult/copy) (del)



(del) (e-unit) (copy-del)



So:

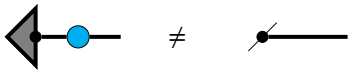
So:

□

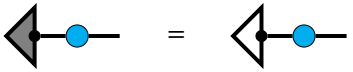
Lemma 4.5.10 (All-or-Nothing)**.** Consider the set $e(\{x\})$ obtained by applying the idempotent e to a singleton $\{x\}$, and take an arbitrary element $y \in e(x)$ of this set. Then $e(\{y\}) = \varnothing$ or $e(\{x\}) = e(\{y\})$. Diagrammatically:

Proof.

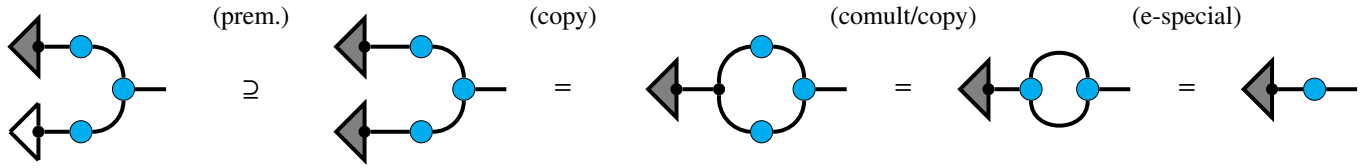
Suppose



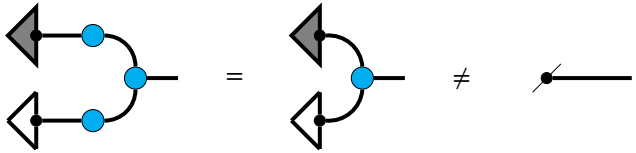
For the claim, we seek:



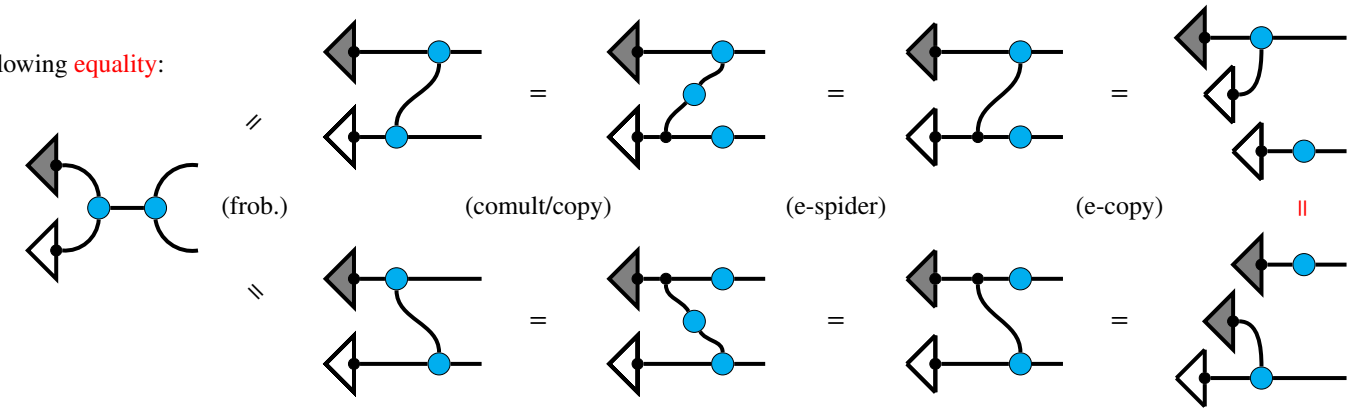
We have the following inclusion:



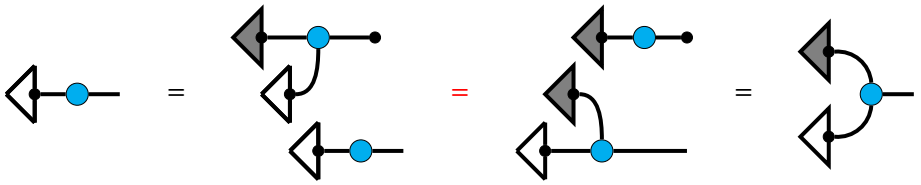
Therefore:



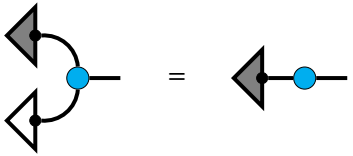
So we have the following equality:



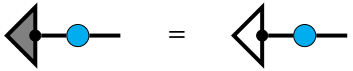
Which implies:



and symmetrically,

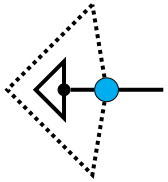


So we have the claim:

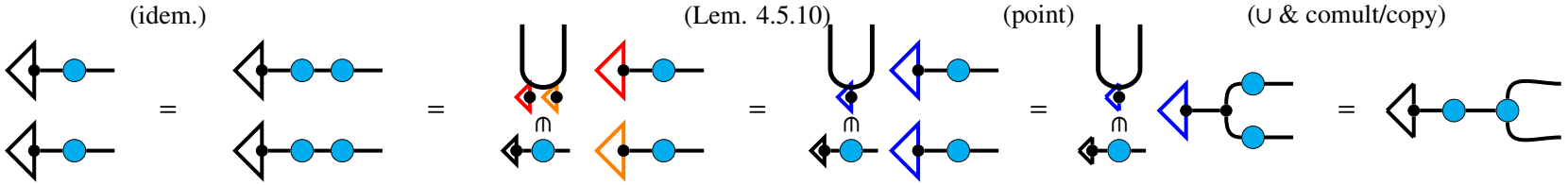


□

Proposition 4.5.11 (*e* of any point is *e*-copiable)**.**

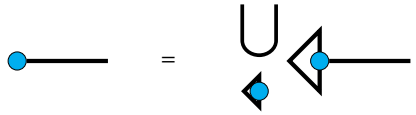


Proof.



□

Proposition 4.5.12 (The unit is the union of all *e*-copiables)**.**

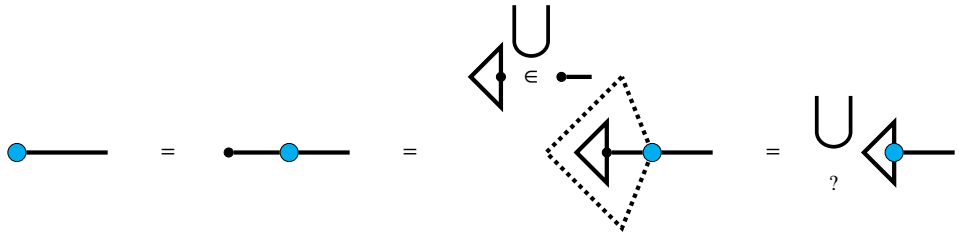


Proof.

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

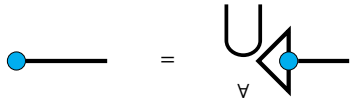
(Lem. 4.5.18) (evr.) (unit/evr.)

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



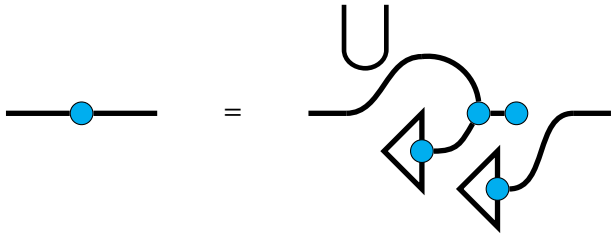
(unit/evr.) (Prop. 4.5.11)

So the containment must be an equality.

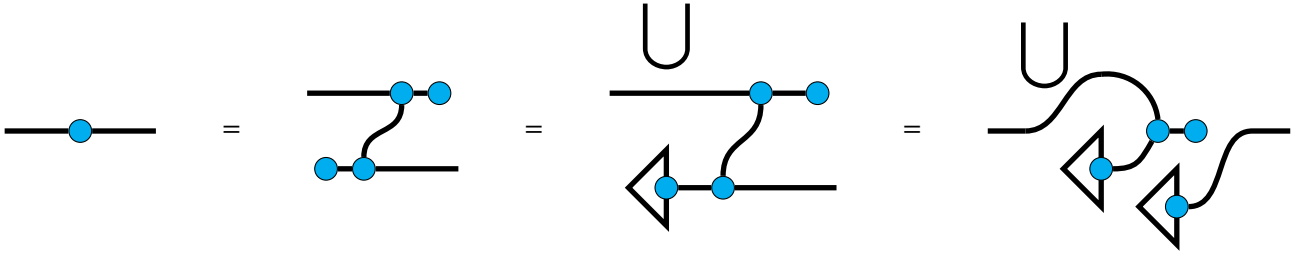


□

Proposition 4.5.13 (*e*-copiable decomposition of *e*)**.**



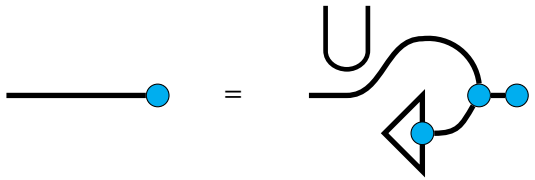
Proof.



(Prop. 4.5.12) (*e*-copiable)

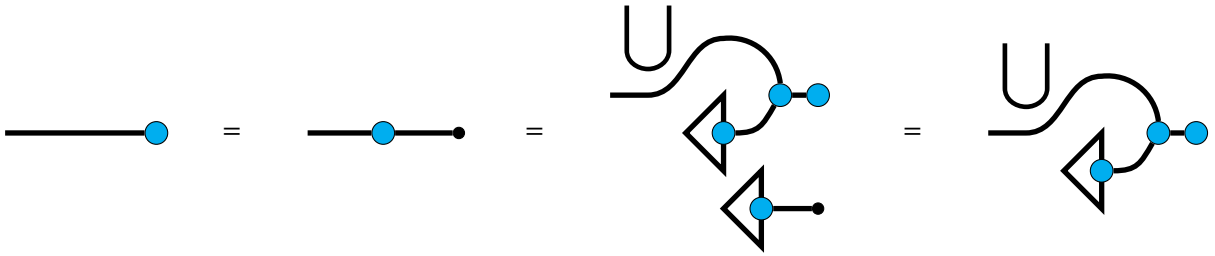
□

Proposition 4.5.14 (*e*-copiable decomposition of counit)**.**



Proof.

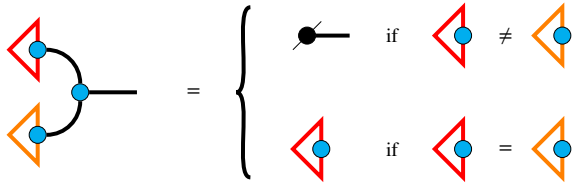
(Prop. 4.5.13)



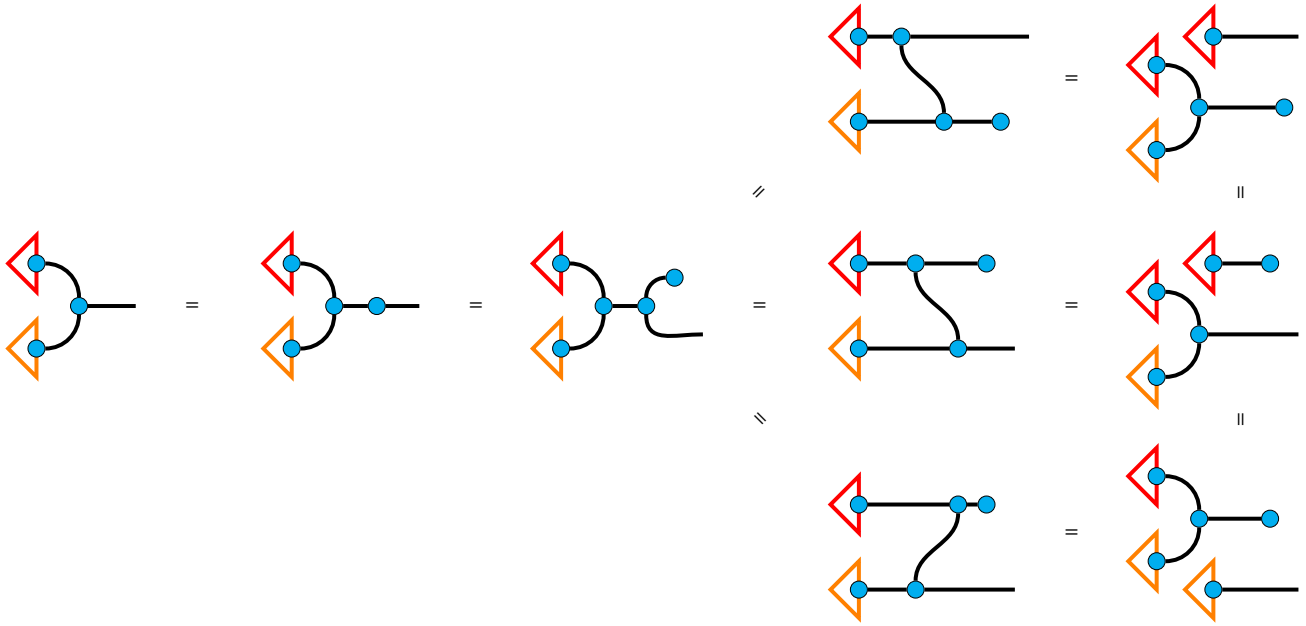
□

THE *e*-COPIABLE STATES REALLY DO BEHAVE LIKE AN ORTHONORMAL BASIS, AS THE FOLLOWING LEMMAS SHOW.

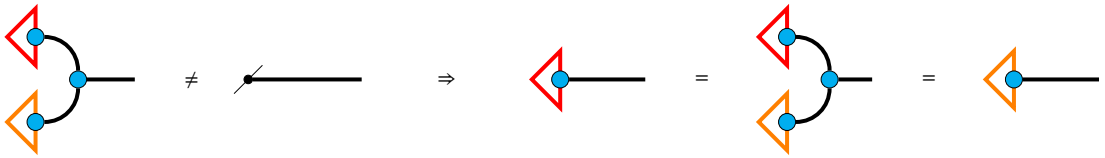
Lemma 4.5.15 (*e*-copiables are orthogonal under multiplication)**.**



Proof.

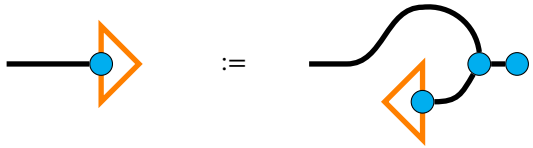


So:

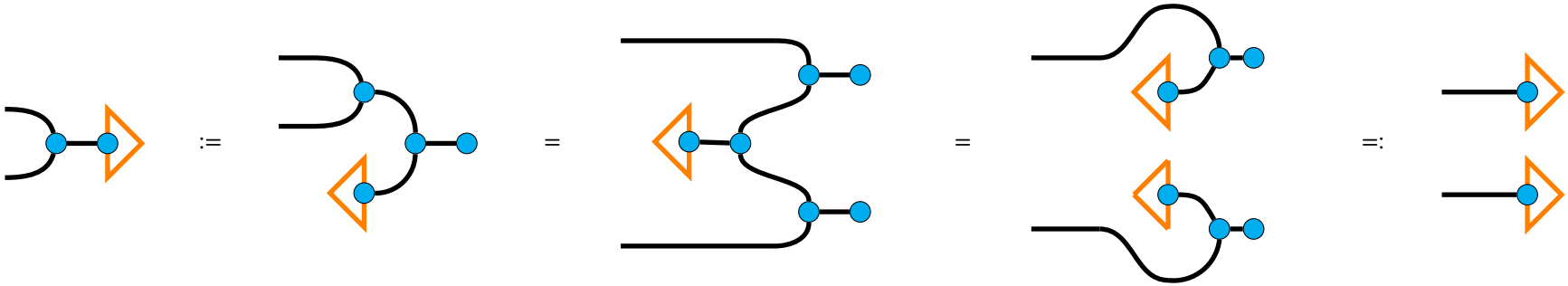


□

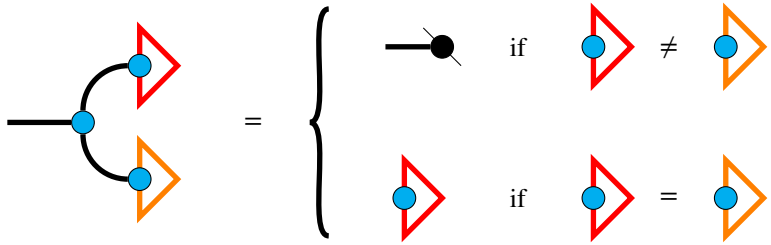
Convention 4.5.16 (Shorthand for the open set associated with an *e*-copiable)**.** We introduce the following diagrammatic shorthand.



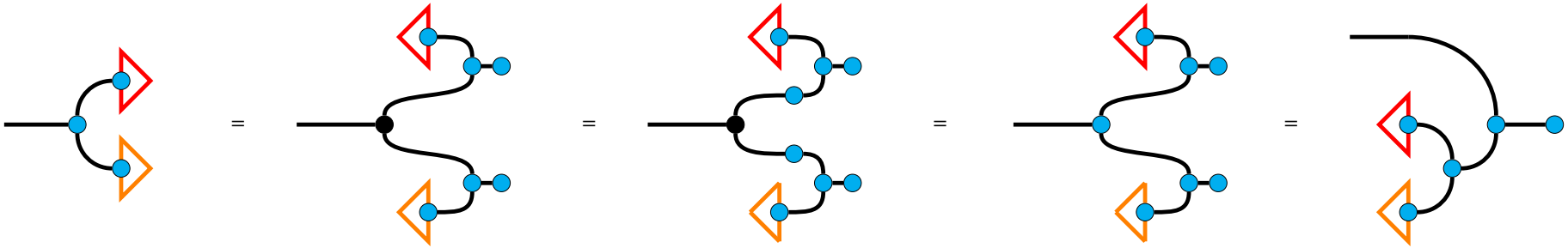
Including the coloured dot is justified, because these open sets are co-copiable with respect to the multiplication of the sticky spider.



Lemma 4.5.17 (Co-match)**.**

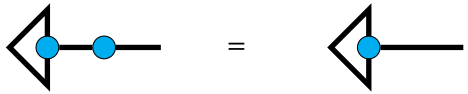


Proof.



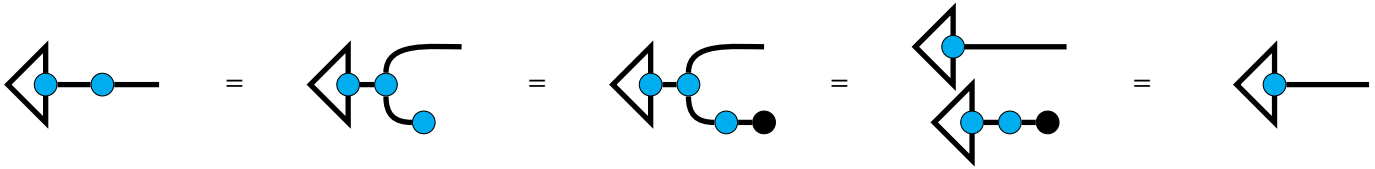
The claim then follows by applying Lemma 4.5.15 to the final diagram. □

Lemma 4.5.18 (e-copiables are e-fixpoints)**.**



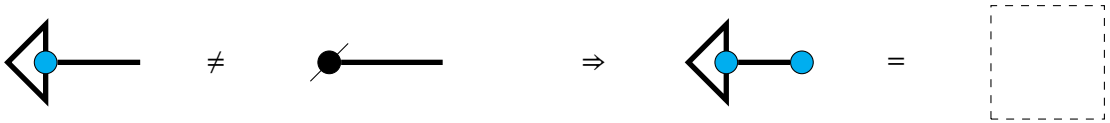
Proof.

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



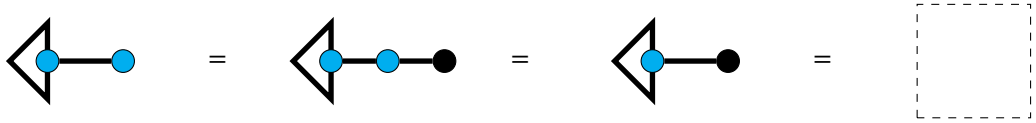
Observe that the final equation of the proof also holds when the initial e-copiable is the empty set. □

Lemma 4.5.19 (*e*-copiables are normal)**.**



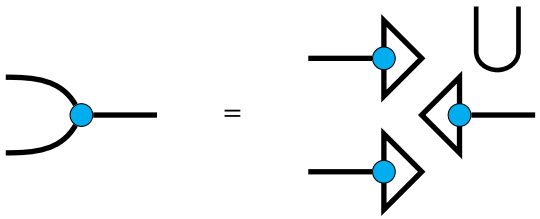
Proof.

(coun/del) (Lem. 4.5.18) (Prem.)

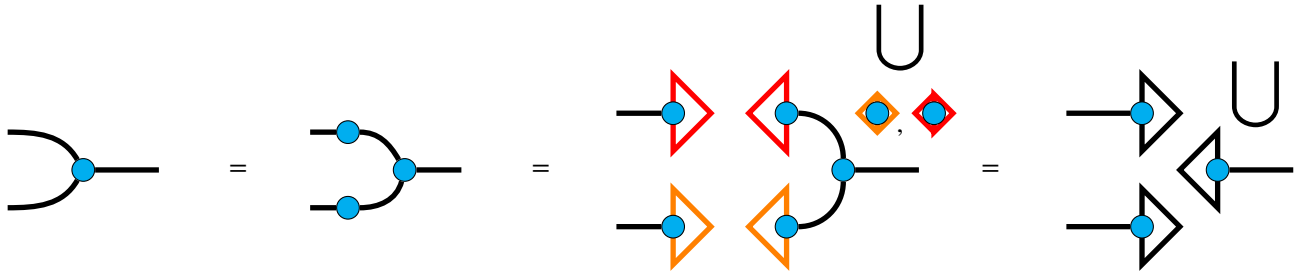


□

Proposition 4.5.20 (*e*-copiable decomposition of multiplication)**.**



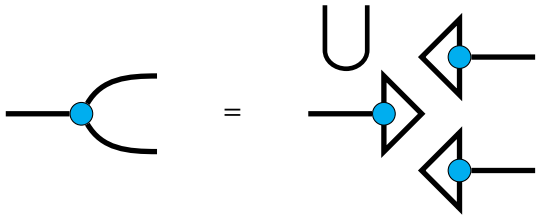
Proof.



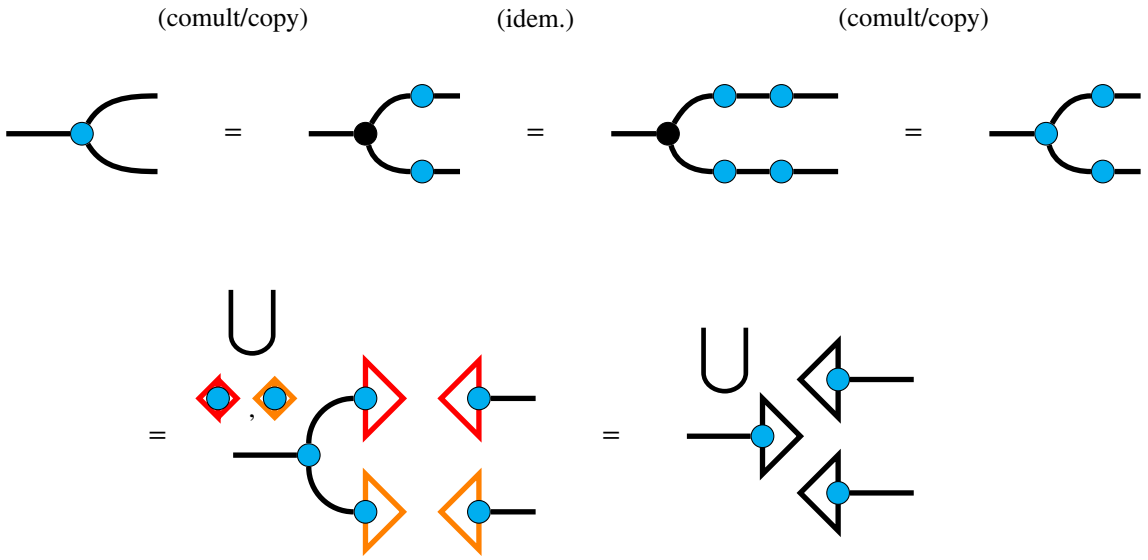
(e-spider) (Prop. 4.5.13) (Lem. 4.5.15)

□

Proposition 4.5.21 (*e*-copiable decomposition of comultiplication)**.**



Proof.

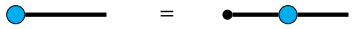


□

NOW WE CAN PROVE THEOREM 4.5.8.

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.

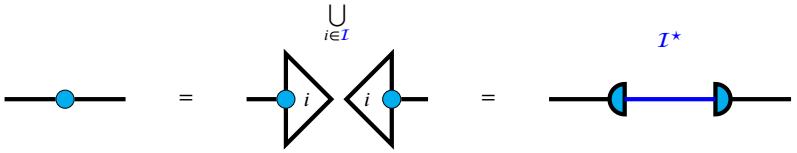
Unit/everything



Comult/copy

The crucial observation is that the e-copiable decomposition of the idempotent given by Proposition 4.5.13 is equivalent to a split idempotent though the set of e-copiables equipped with discrete topology.

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



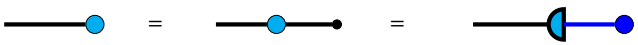
(Prop. X)

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

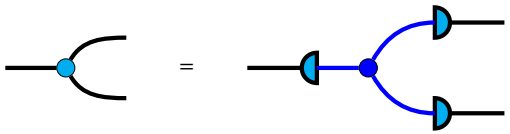
By copiable basis Proposition 4.5.12 and the decompositions Propositions 4.5.14, 4.5.20, 4.5.21, we obtain the only-if direction.

(unit/evr.) (Prop. 4.5.12)

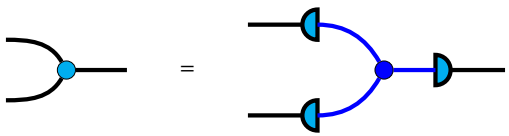
(Prop. 4.5.9) (Prop. 4.5.14)



(Prop. 4.5.21)



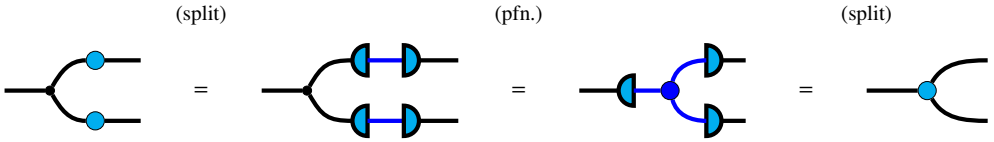
(Prop. 4.5.20)



The if direction is an easy check. For the unit/everything relation, we have:

(split) (Prop. 4.5.7) (split)

For the counit/delete relation, we observe that for any split idempotent, the retract must be a partial function. To see this, suppose the split idempotent $e = r; s$ is on (X, τ) and the discrete topology is $Y^\star$. 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 $I := r(x) \subseteq Y$ contains more than one 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 must recover the identity on $Y^\star$, the section 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 $(a, b), (b, a) \in s; r$, which by $a \neq b$ contradicts that $s; r$ is the identity relation 1_Y.



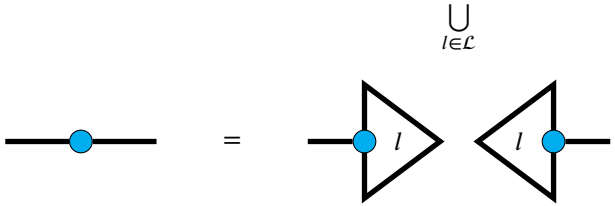
□

*4.6 Topological concepts in flatland via **ContRel***

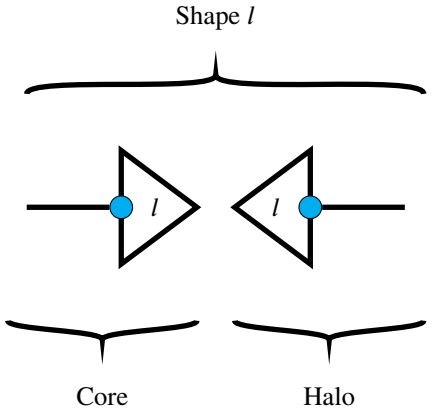
The goal of this section is to demonstrate the use of sticky spiders as formal semantics for the kinds of schematic doodles or cartoons we would like to draw. Throughout we consider sticky spiders on $\mathbb{R}^2$. In Section 4.6.1, we introduce how sticky spiders may be viewed as labelled collections 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 4.6.2 and 4.6.3 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 4.6.4 and 4.6.5 we diagrammatically characterise an analogue of homotopy in the relational setting. Finally, in Sections 4.6.6 and 4.6.7 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.

4.6.1 Shapes and places

Definition 4.6.1 (Labels, shapes, cores, halos)**.** Recall by Proposition 4.5.13 that we can express the idempotent as a union of continuous relations formed of a state and test, for some indexing set of *labels* $\mathcal{L}$.



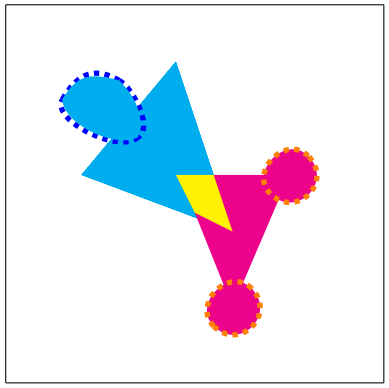
A *shape* is a component of this union corresponding to some arbitary $l \in L$. So we refer to a sticky spider as a labelled collection of shapes. The state of a shape is the *halo* of the shape. The halos are precisely the copiables of the sticky spider. The test of a shape is the *core*. The cores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



Proposition 4.6.2 (Core exclusion: Distinct cores cannot overlap)**.** *Proof.* A direct consequence of Lemma 4.5.17. □

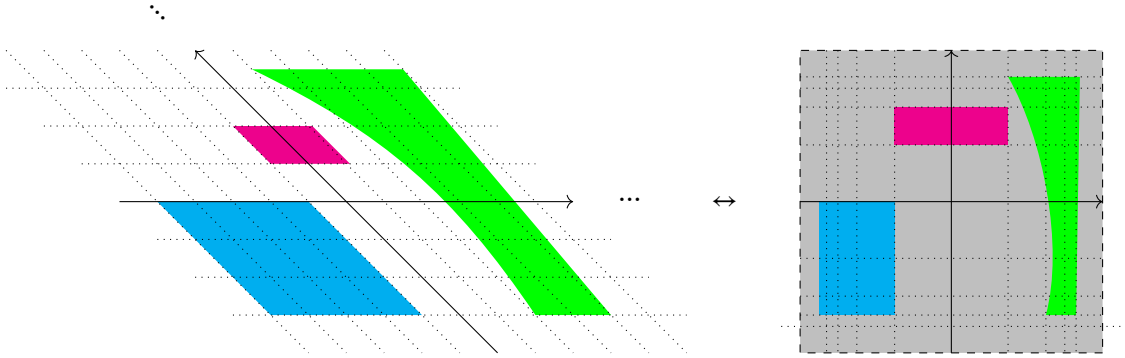
Proposition 4.6.3 (Core-halo exclusion: Each core only overlaps with its corresponding halo)**.** *Proof.* Seeking contradiction, if a core overlapped with multiple halos, Lemma 4.5.18 would be violated. □

Proposition 4.6.4 (Halo non-exclusion: halos may overlap)**.** *Proof.* By example:

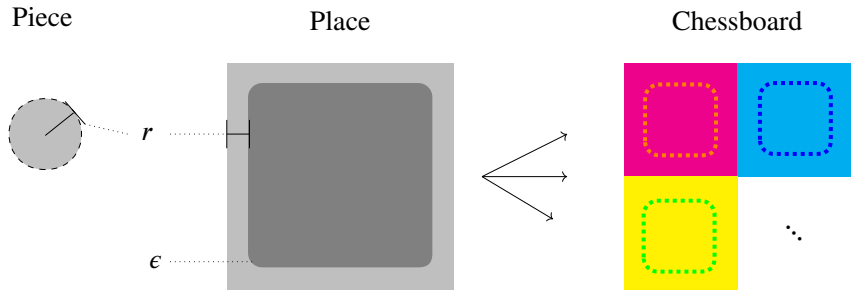


The two shapes are colour coded cyan and magenta. The halos are two triangles which overlap at a yellow region, and partially overlap with their blobby cores. The cores are outlined in dotted blue and orange respectively. Observe that cores and halos do not have to be simply connected; in this example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

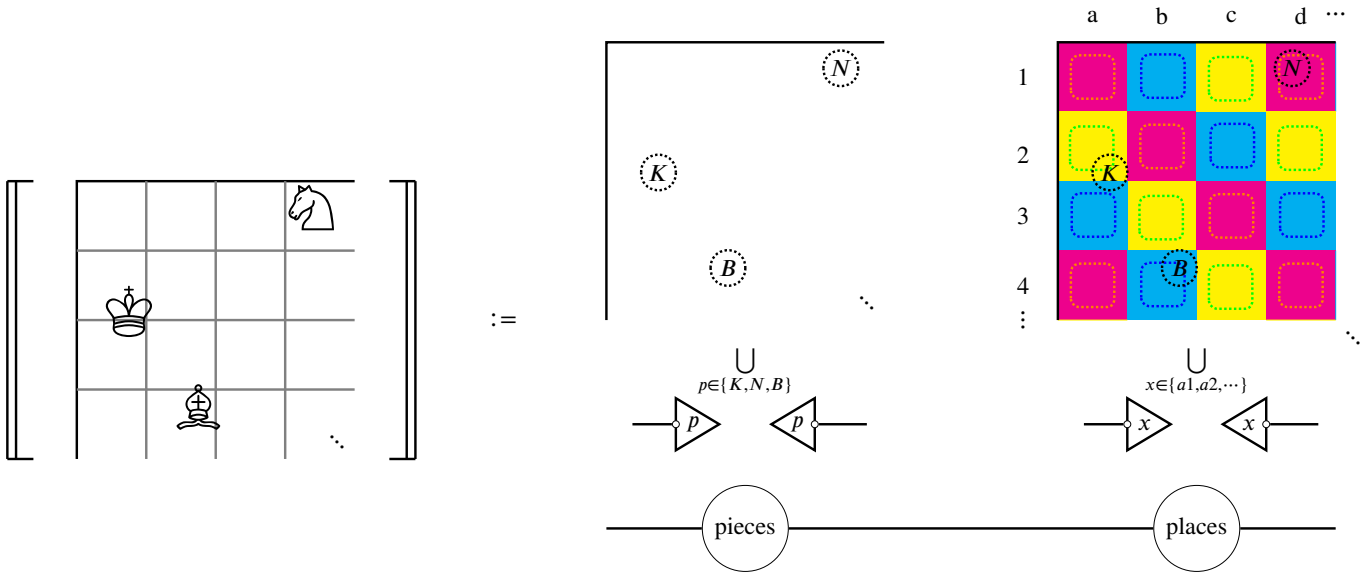
Remark 4.6.5. When we draw on a finite canvas representing all of euclidean space, properly there should be a fishbowl effect that relatively magnifies shapes close to the origin and shrinks those at the periphery, but that is only an artefact of representing all of euclidean space on a finite canvas. 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 4.6.6 (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.



4.6.2 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.

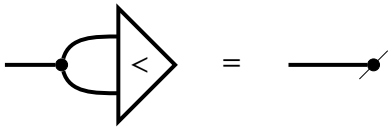
THE REALS There are many spaces homeomorphic to the real line. How do we know when we have one of them? The following theorem provides an answer:

Theorem 4.6.7 (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 $\mathbb{R}$.

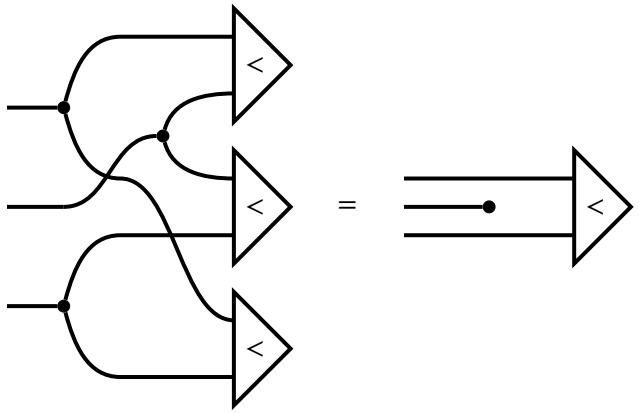
We can define all of these pieces using diagrammatic equations.

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:

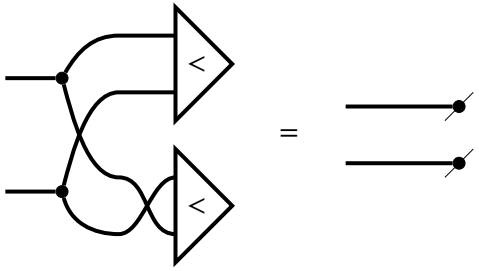
Antireflexive ($\forall x : x \not< x$)



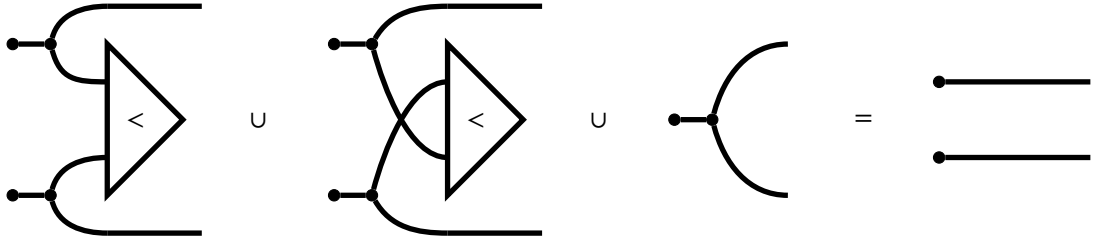
Transitive ($\forall xyz : x < y \,\&\, y < z \Rightarrow x < z$)



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



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

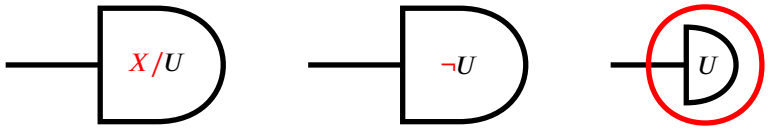


ENDPOINTS We can introduce endpoints for open intervals directly by asking for the space X to have points that are less than or greater than all other points. Another method, which we

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 4.6.8 (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.

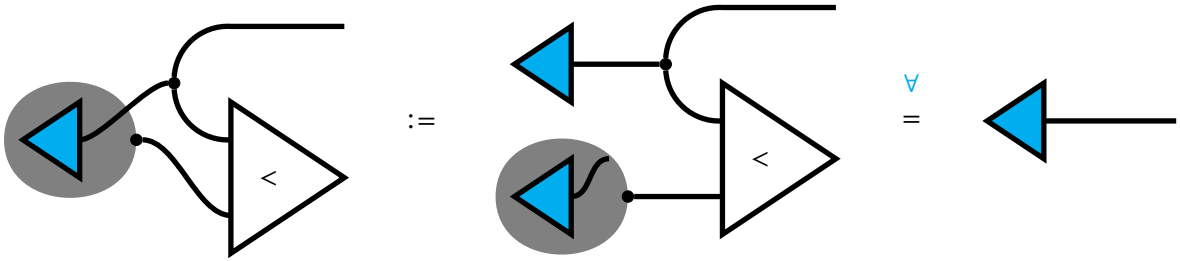


Unfortunately, the complementation operation $X/-$ is not in general a continuous relation, hence in the lattermost expression above we resort to using bubbles as a syntactic sugar. Formally, these bubbles are *endocombinators*, the semantics and notation for which we borrow and modify from [].

Definition 4.6.9 (Partial endocombinator)**.** In a category C, a *partial endocombinator* on a homset $(C)(A, B)$ is a function $(C)(A, B) \to (C)(A, B)$

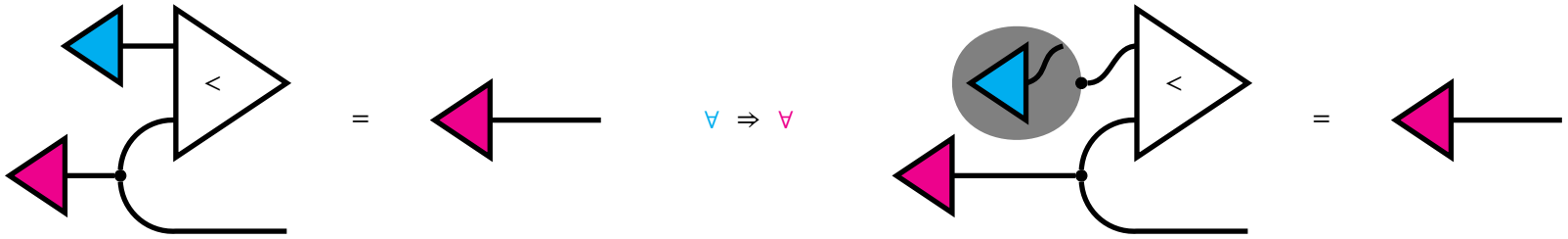
Using this technology, we can define:

Upper Bound



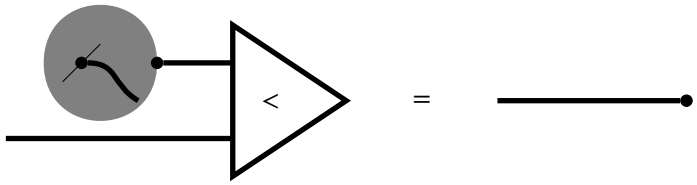
And we can add in further equations governing the upper bound endocombinator to turn it into a supremum, where the lower endpoint is obtained as the supremum of the empty set, and the upper endpoint is the supremum of the whole set.

Supremum

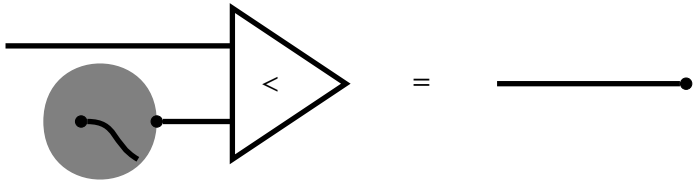


Now we can define endpoints purely graphically:

Lower endpoint



Upper endpoint

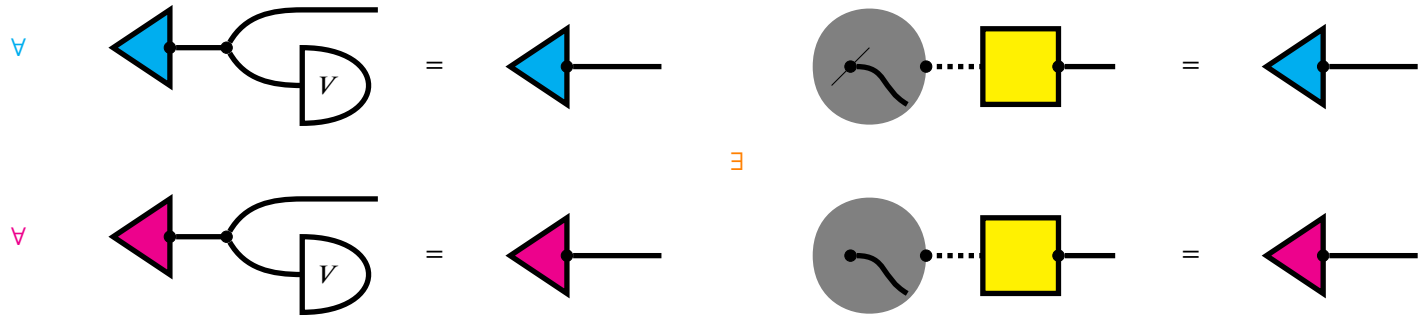


Going forward, we will denote the unit interval using a thick dotted wire.

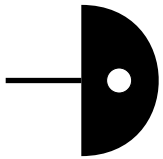
Simply connected spaces

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.

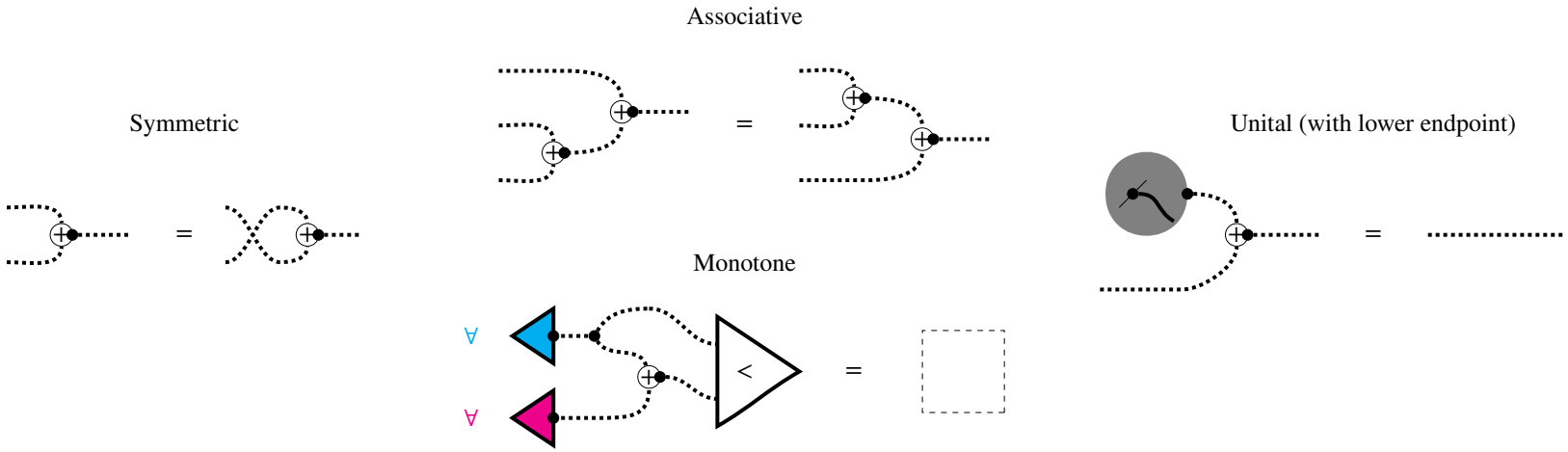


4.6.3 Displacing shapes

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

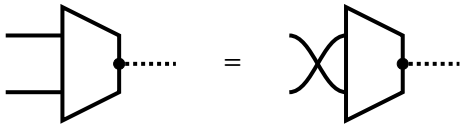
More precisely, we only need an additive monoid structure on the unit interval. We do not care about obtaining precise values from our metric, and we will not need to subtract distances from each other. All we need to know is that the lower endpoint stands in for "zero distance" – as the unit of the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



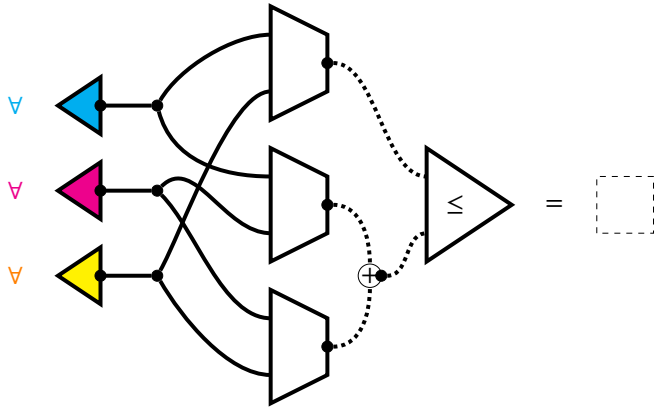
METRICS

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

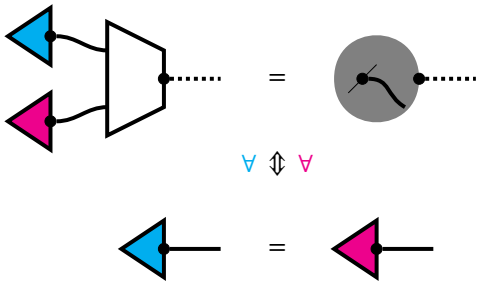
$d(x, y) = d(y, x)$



$d(x, z) \leq d(x, y) + d(y, z)$



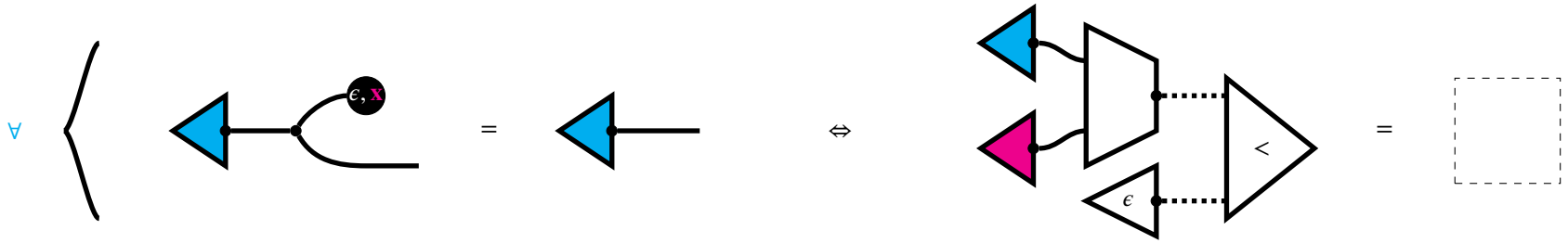
$d(x, y) = 0 \iff x = y$



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.

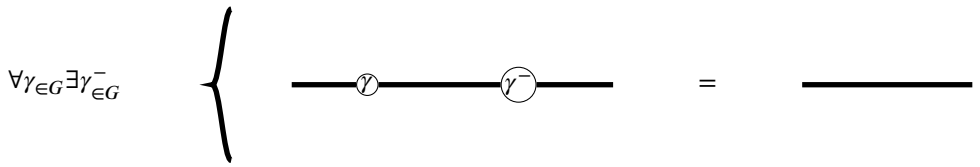
Open ball of radius ϵ at a point **x**



Topological Group

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

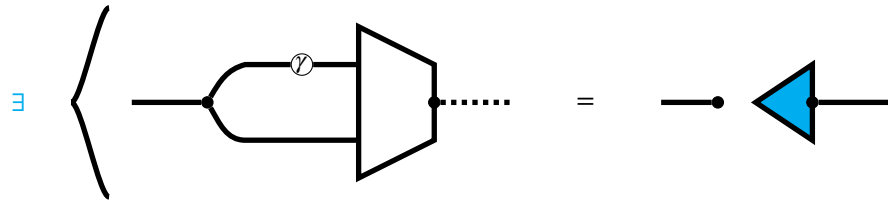
A topological group G



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:

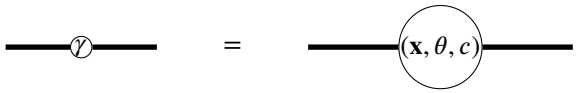
γ is an *isometry*



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 $\mathbf{R}^2$



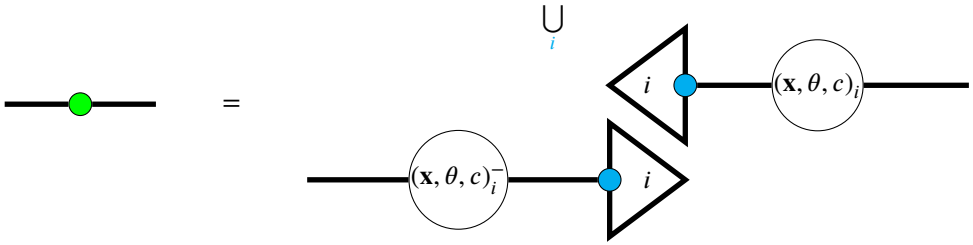
$\mathbf{x} \in \mathbf{R}^2$

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

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

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.

4.6.4 Moving shapes

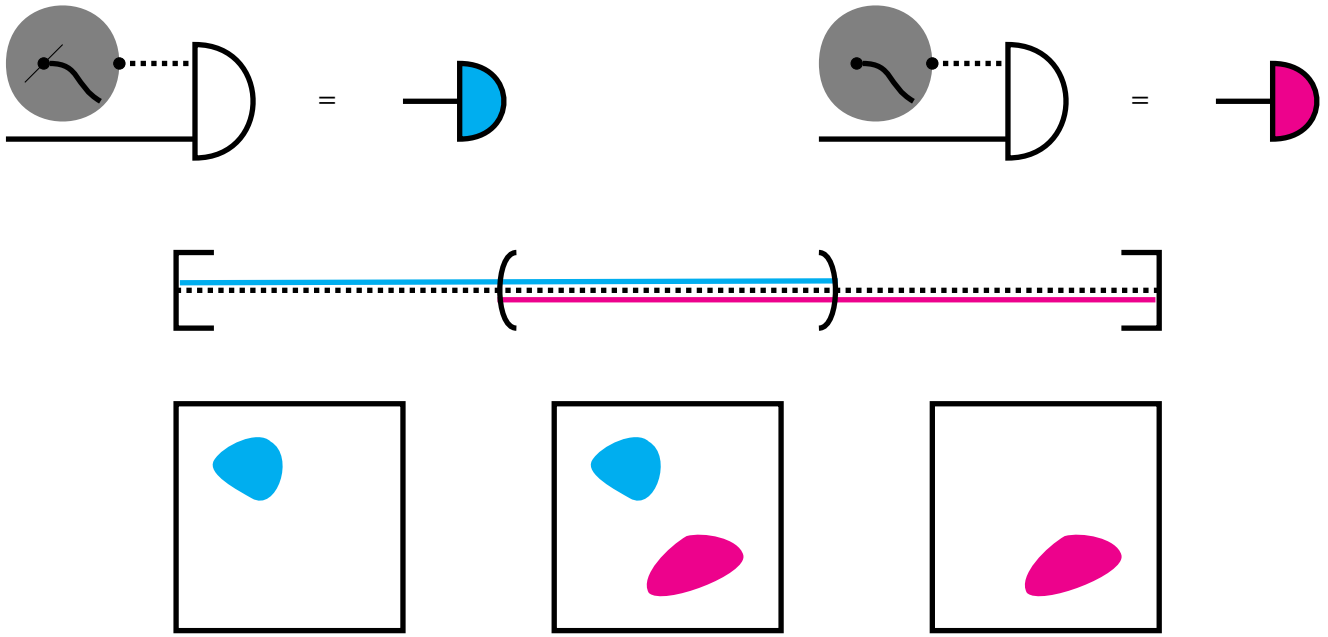
If we want continuous transformations in the plane from the configuration of shapes in one spider to end at the configuration of shapes in another, we ought to define an analogue of *homotopy*: the continuous deformation of one map to another. However, we will have to massage the definition a little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

HOMOTOPY IN **CONTREL**

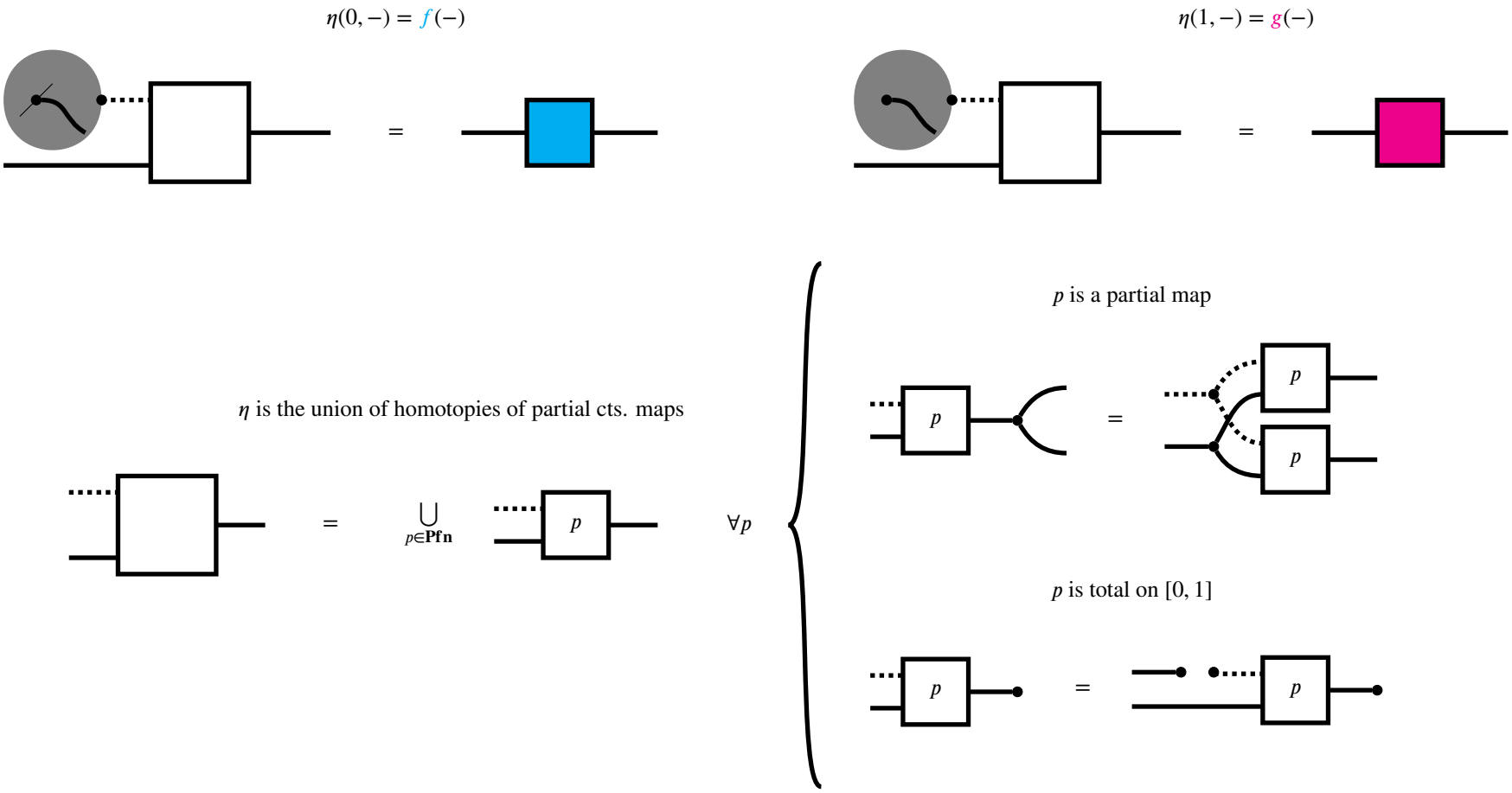
Usually, when we are restricted to speaking of topological spaces and continuous functions, a homotopy is defined:

Definition 4.6.10 (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 $\eta(1,-) = g(1,-)$.

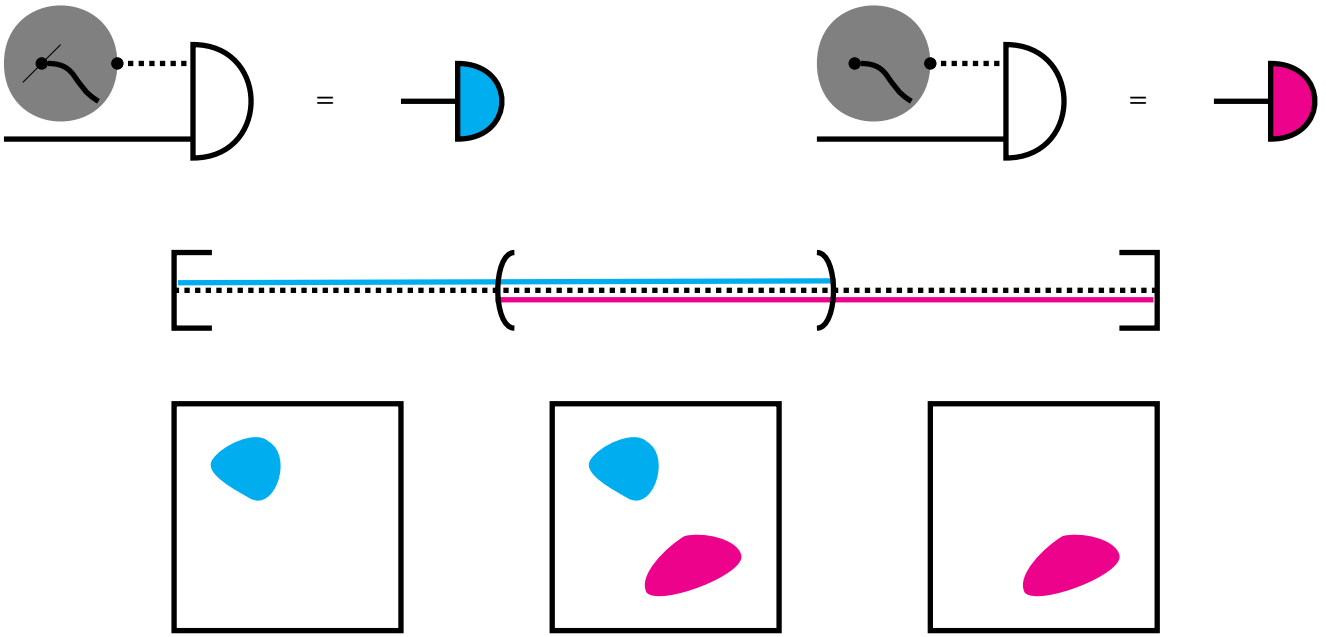
In other words, a homotopy is like a short film where at the beginning there is an f, which continuously deforms to end the film being a g. Directly replacing "function" with "relation" in the above definition does not quite do what we want, because we would be able to define the following "homotopy"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



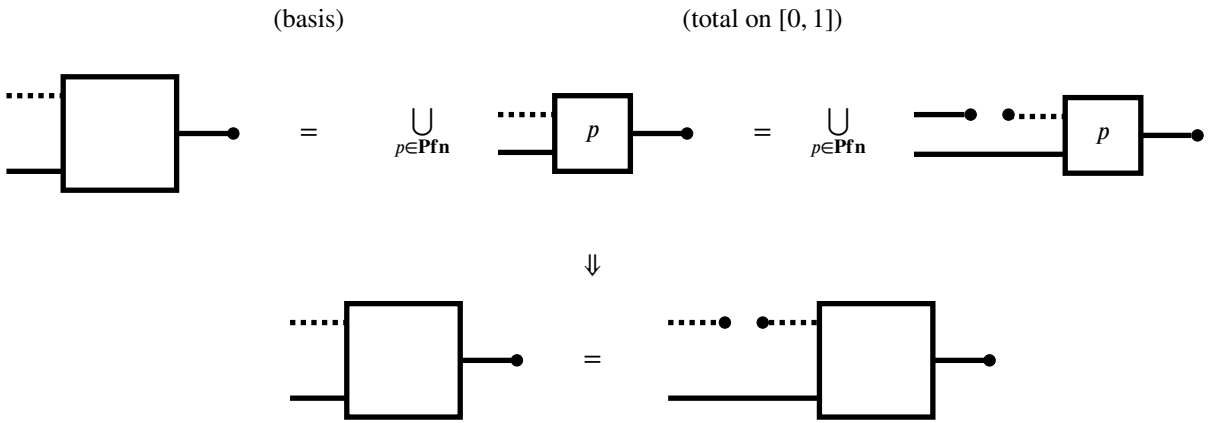
What is happening in the above film is that we have our starting open set, which stays constant for a while. Then suddenly the ending open set appears, the starting open disappears, and we are left with our ending; while *technically* there was no discontinuous jump, this isn't the notion of sliding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



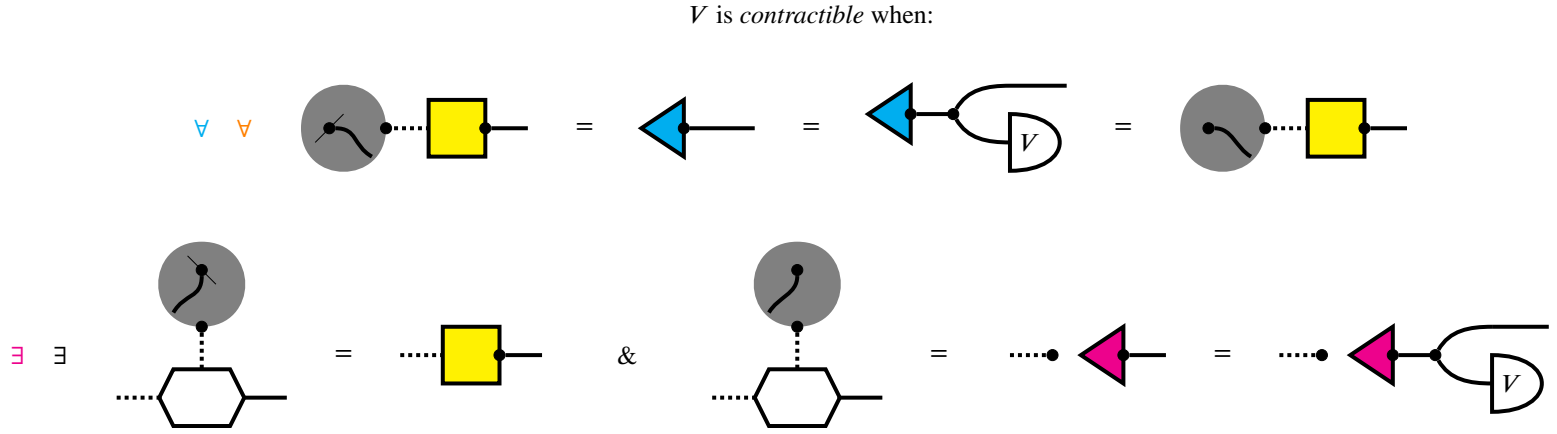
Observe that the second condition asking for decomposition in terms of partial comes for free by Proposition 4.4.20; the constraint of the definition is provided by the first condition, which is a stronger condition than just asking that the original continuous relation be total on I:



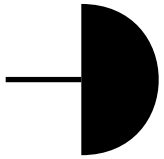
This definition is "natural" in light of Proposition 4.4.20, that the partial continuous functions $A \to B$ form a basis for **ContRel**(A, B): we are just asking that homotopies between partial continuous functions – which can be viewed as regular homotopies with domain restricted to the subspace topology induced by an open set – form a basis for homotopies between continuous relations.



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



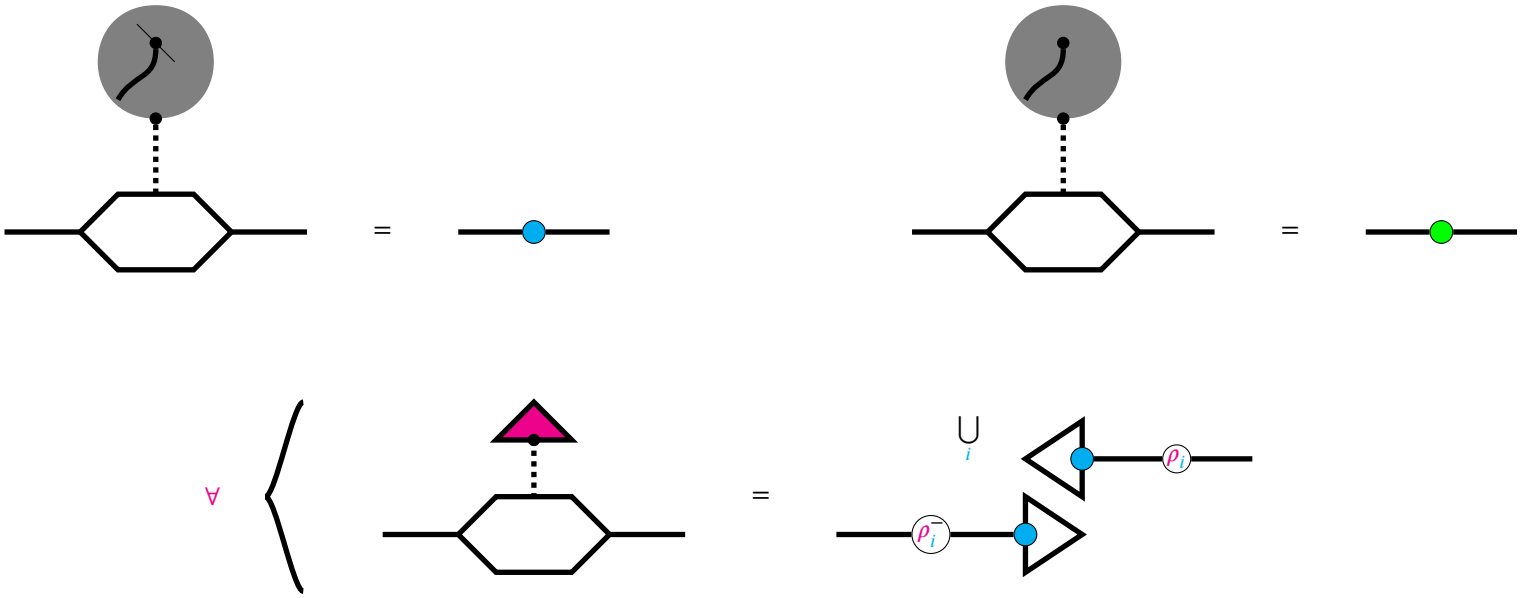
Contractible open sets are worth their own notation too; a solid black effect, this time with no hole.



4.6.5 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.

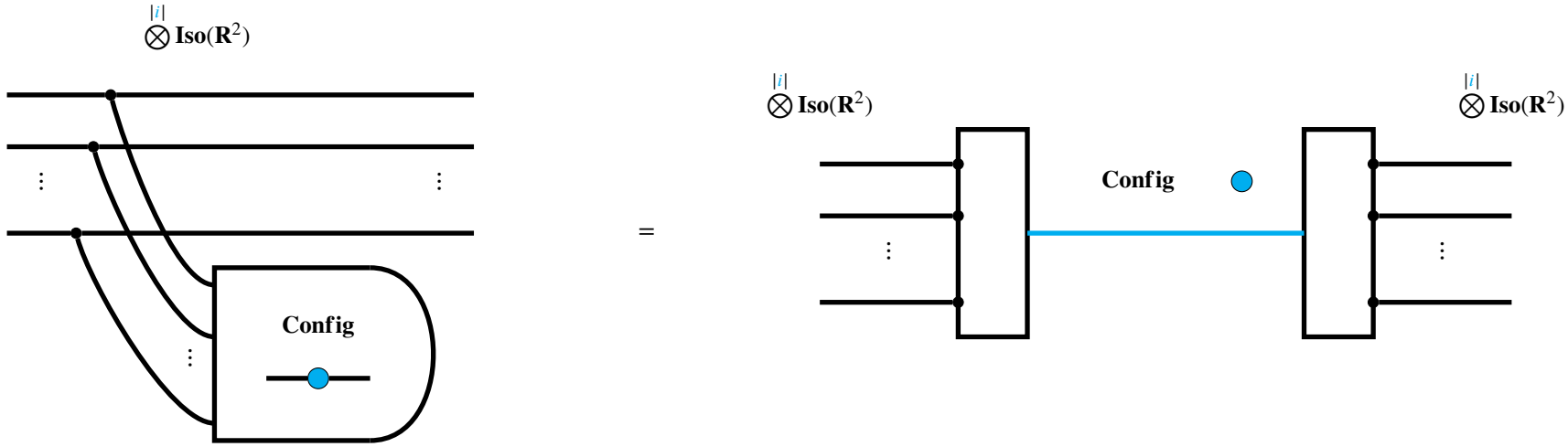
Rigid motion



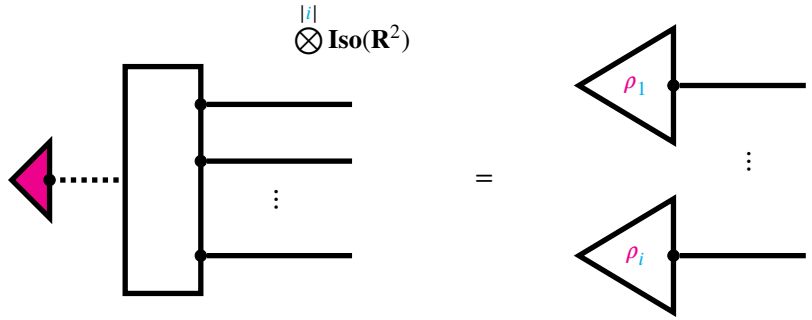
CONFIGURATION SPACES

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

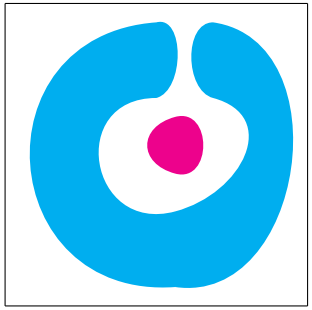


Observe that the data of rigid motion on a sticky spider as we have defined above can be captured as a continuous map from the unit interval to rigid transformations: one for each shape in the spider. This is precisely a continuous path in configuration space.

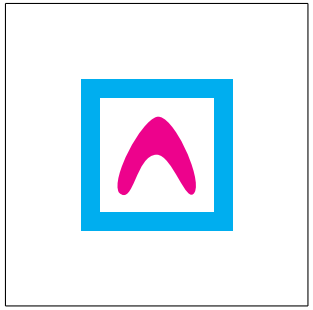


What are the connected components of configuration space? Evidently, there are pairs of spiders that are both valid displacements, but not mutually reachable by rigid motion. For example, shapes might *enclose* or *trap* other shapes, or shapes might be *interlocked*. Depicted below are some pairs of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

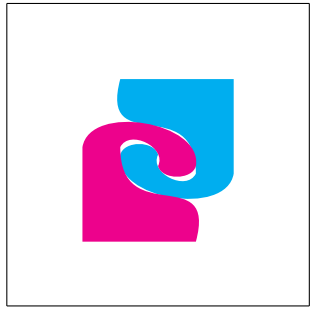
Trapped



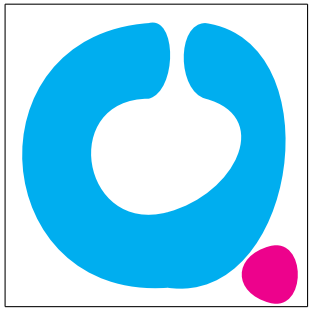
Enclosed



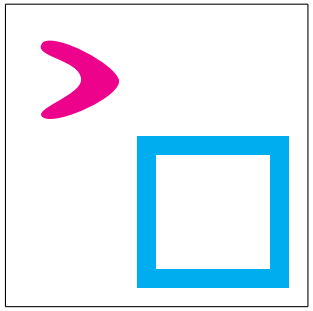
Interlocked



Not trapped



Not enclosed



Not interlocked

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

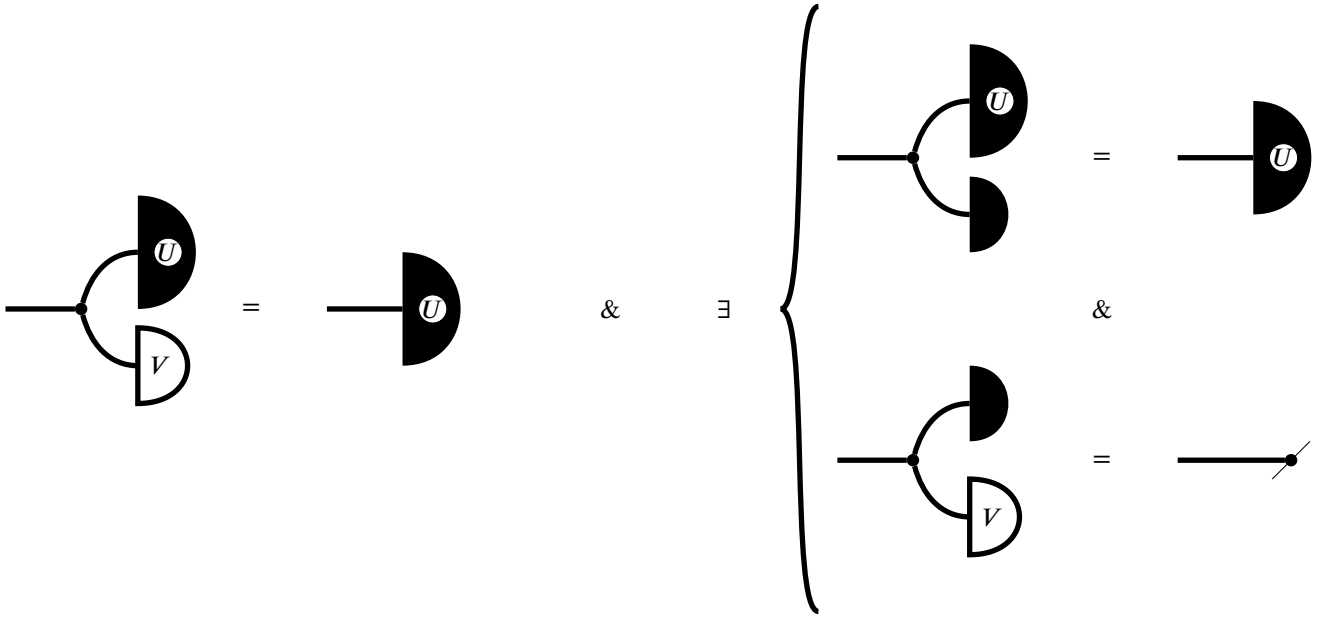
4.6.6 Modelling linguistic topological concepts

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:

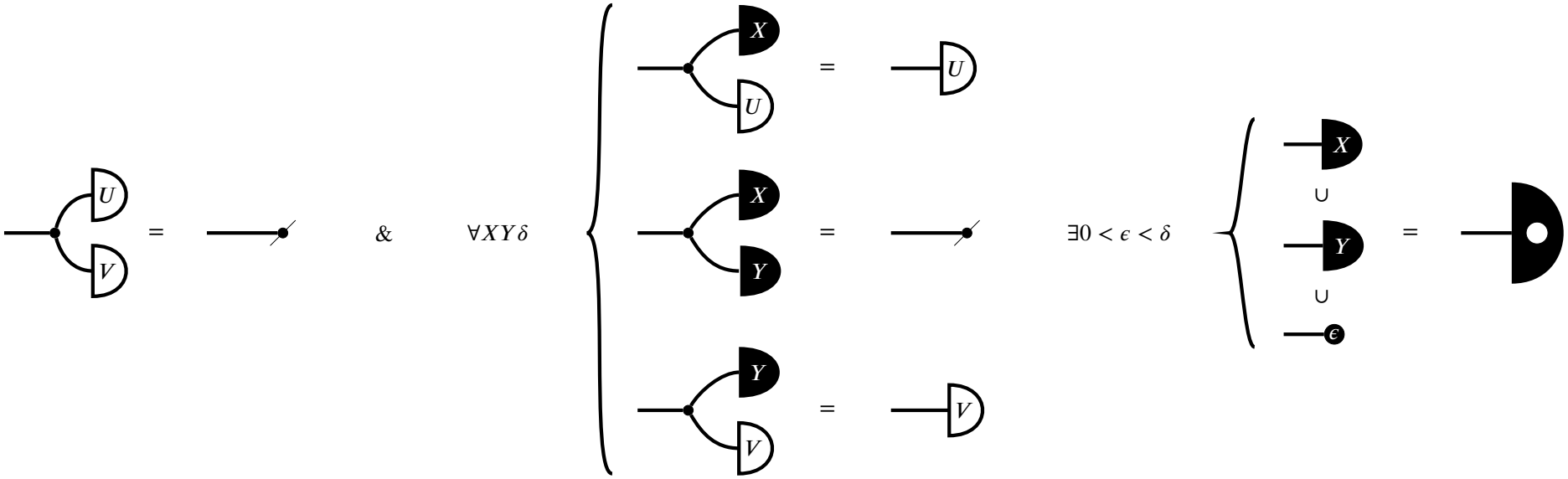
U is *a part of V*



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:

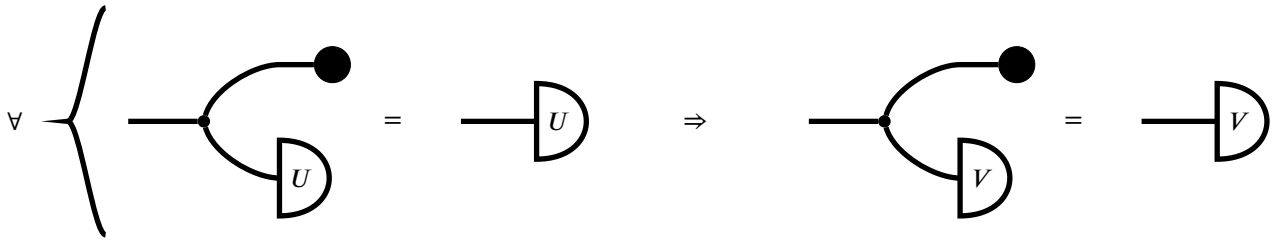
U and V are *touching*



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.

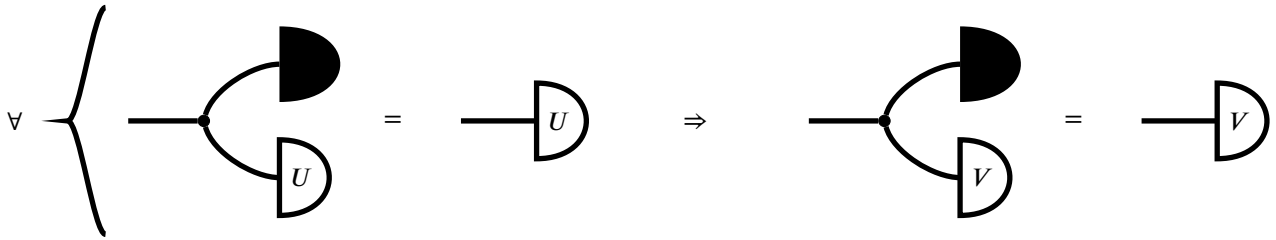
V is *within* U, or U *surrounds* V



Containers and enclosure

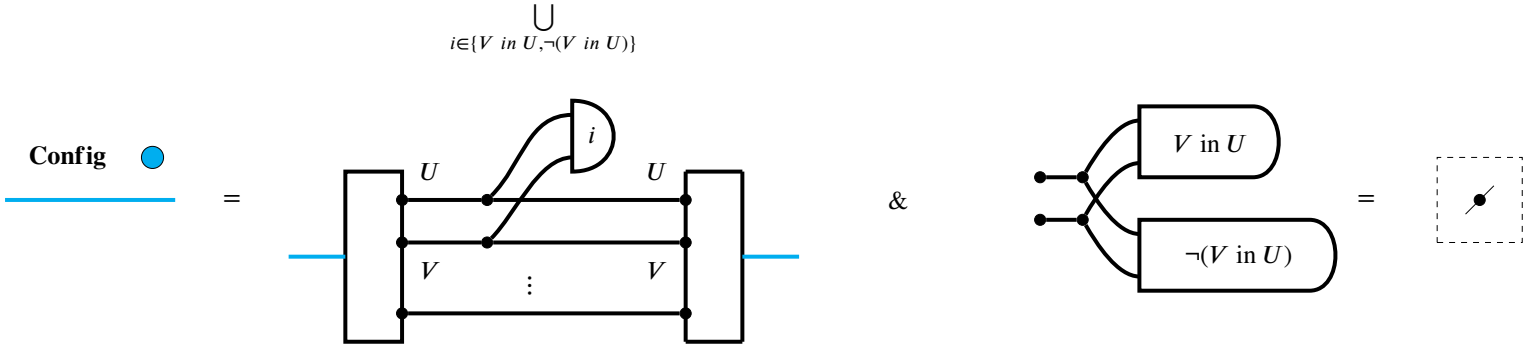
There is a strong version of within-ness, which we will call enclosure. As in when we are in elevators and the door is shut, nothing gets in or out of the container. Intuitively, there is a hole in the container surrounded on all sides, and the contained shape lives within the hole. To give a real-world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

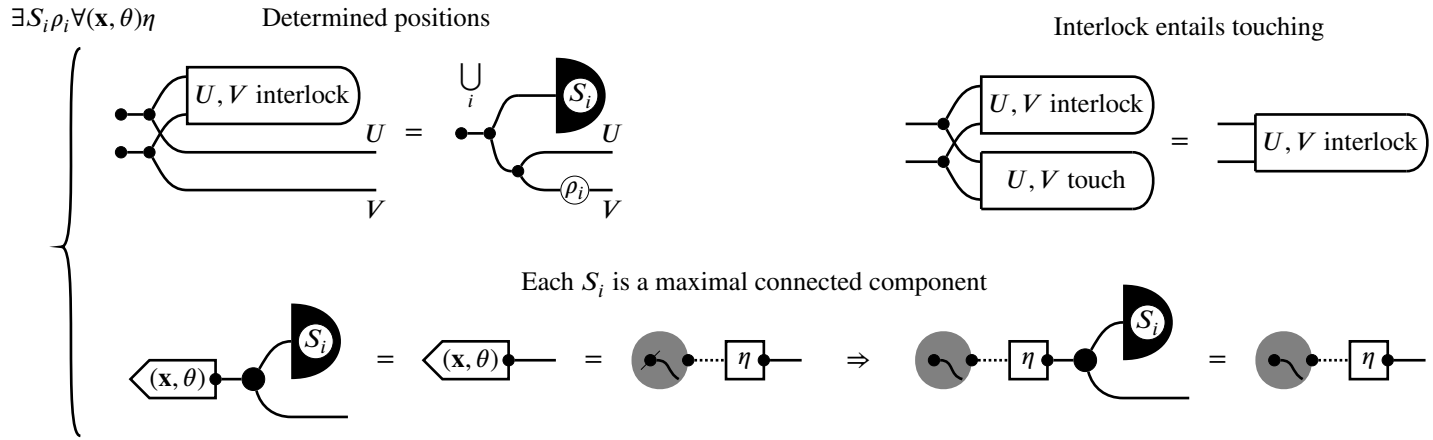
There is an intermediate notion between within-ness and enclosure; for instance, standing in the stonehenge you are surrounded by the pillars, but you can always walk away, whereas if the pillars are very close, such as the bars of a jail cell, a human would not be able to leave the trap while still 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

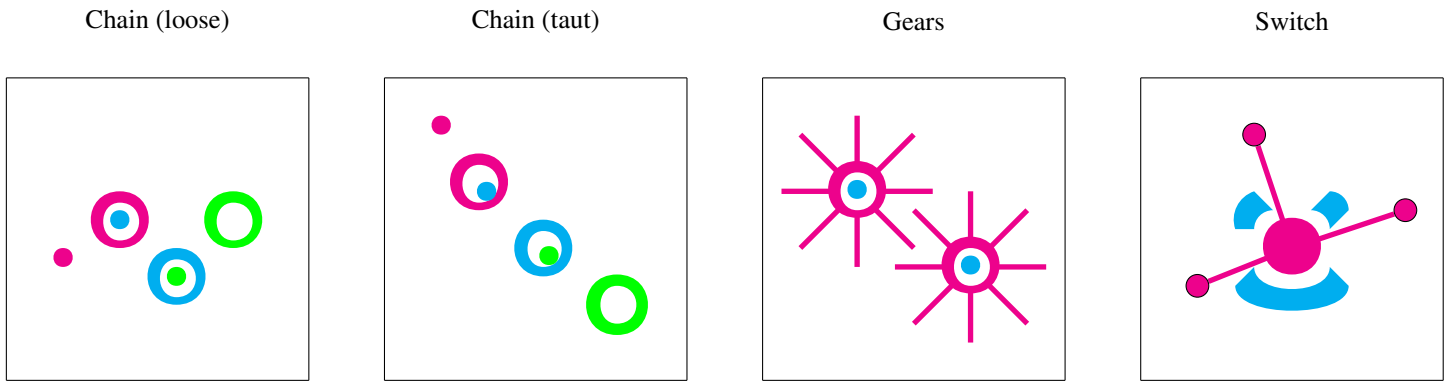
Two shapes might be tightly interlocked without being inside one another. Some potentially familiar examples are plastic models of molecular structure that we encounter in school, metal

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.

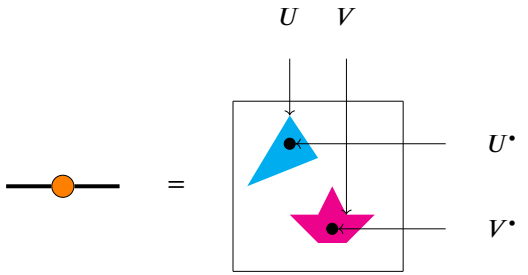


There are as many definitions to go through here as there are potential mechanical models, and among other things, there are mechanically realised clocks [], computers [], and analogues of electric circuits []. So instead, we will allow ourselves to additionally specify open sets as concepts in configuration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

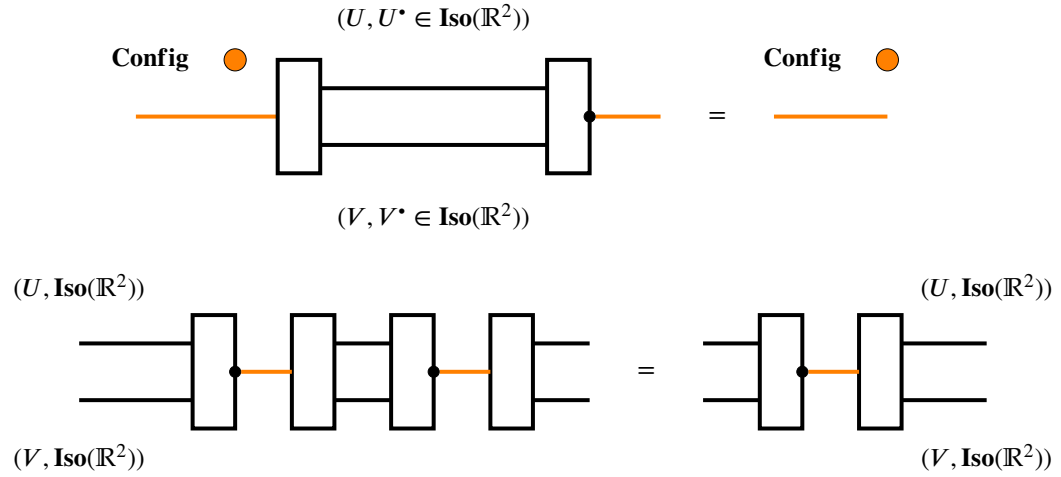
we assure the reader seeking rigour that blueprints exist for all the mechanisms humans have built. Of course in reality mechanical motions are reversible among rigid objects, and directional behaviour is provided by a source of energy, such as gravitational potential, or wound springs. But we may in 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.

4.6.7 States, actions, manner

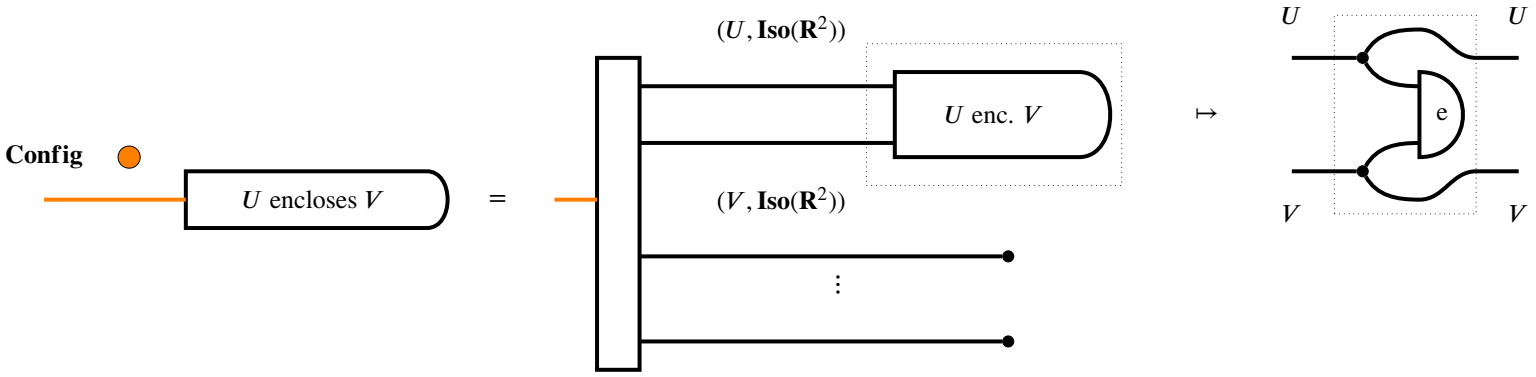
Configuration space explains why we label noun wires: each wire in expanded configuration space must be labelled with the shape within the sticky spider it corresponds to so that the section and retract know how to reconstruct the shapes, since each shape may have a different spatial extent.



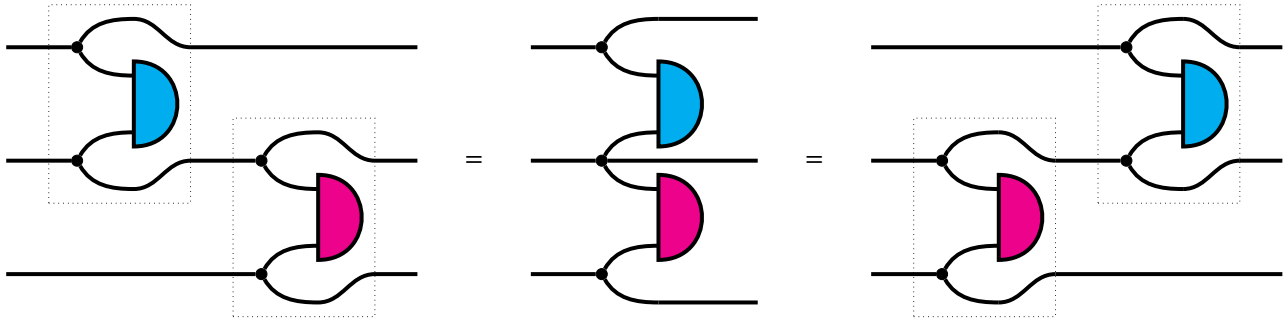
Concretely, for each shape in a sticky spider,
in order to reconstruct the shape,
the data of configuration space
only has to remember a basepoint
(which the isometries act upon)
paired with a label naming the shape
(so that the extent of the shape is reconstructible)



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.



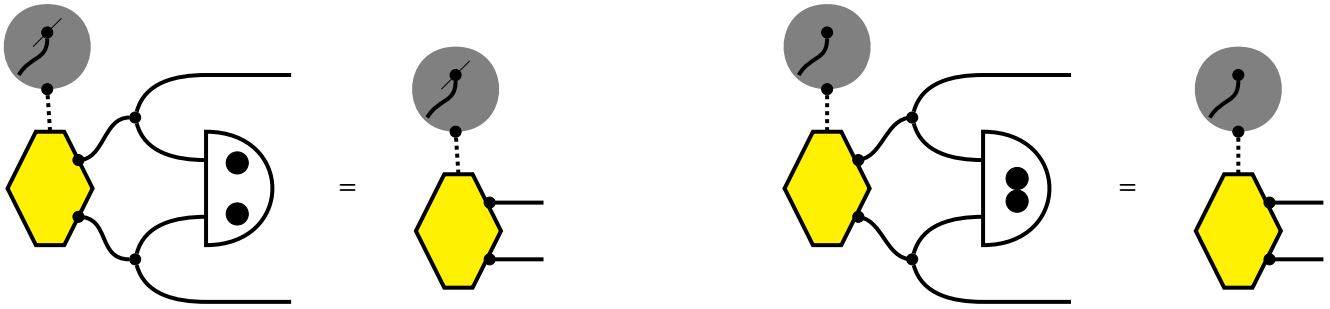
Going forward, we will just label the wires with the names of each shape when necessary. We notice that one feature of this procedure to get gates from open sets is that all gates commute, due to the commutativity of copy.



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.

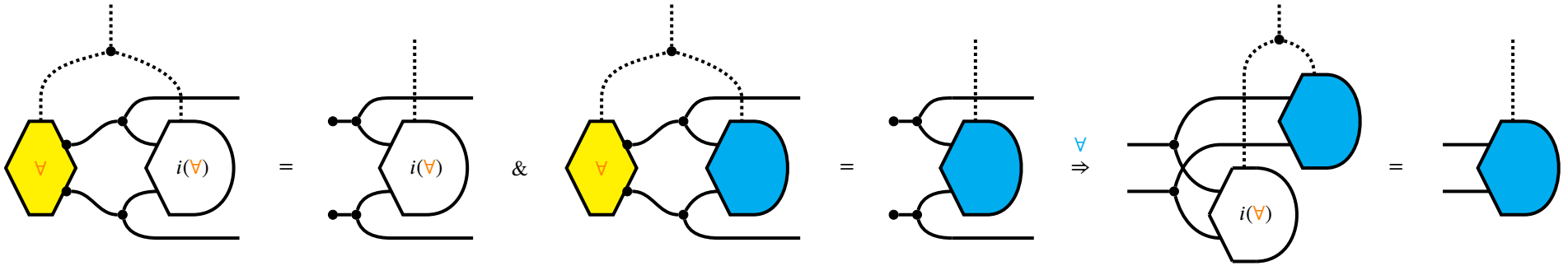
A particular collision trajectory



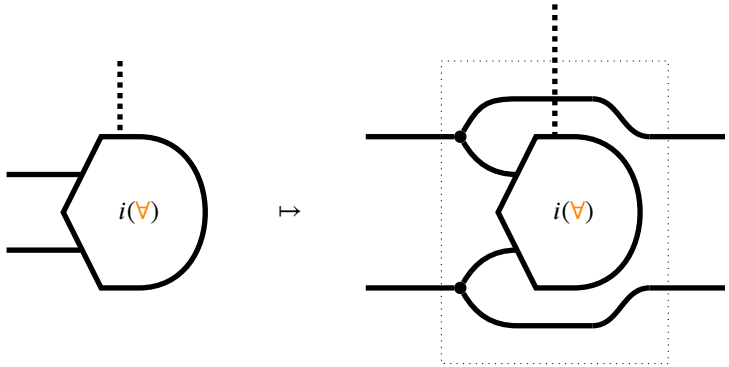
At $t = 0$, the shapes do not touch

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 ∀. Now we seek to define the interior $i(\forall)$ as the concept of collision; the expressible collection of all particular collisions. But this is not just an open set on the potential configuration of shapes, it is a collection of open sets parameterised by homotopy.

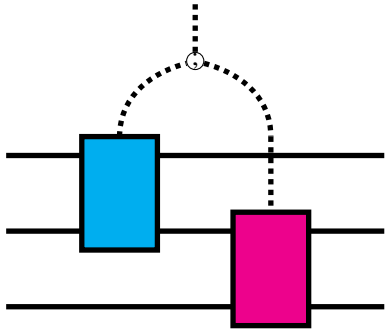


Once we have the open set $i(\forall)$ that corresponds to all expressible collisions, we have a homotopy-parameterised gate. Following a similar procedure, we can construct gates of motion that satisfy whatever pre- and post-conditions we like.

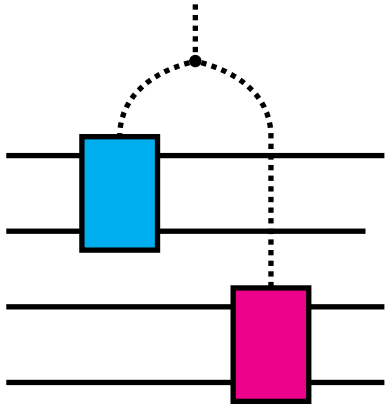


We can compose multiple rigid motions sequentially by a continuous function ; that splits a single unit interval into two: $; := x \mapsto \begin{cases} (2x, 0) \text{ if } x \in [0, \frac{1}{2}) \\ (1, 2x - 1) \text{ if} x \in [\frac{1}{2}, 1] \end{cases}$. The effect of the map is to splice two vignettes of the same length together by doubling their speed, then placing them one after the other. We can achieve the same thing without resorting to units of measurement, because recall by Theorem 4.6.7 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

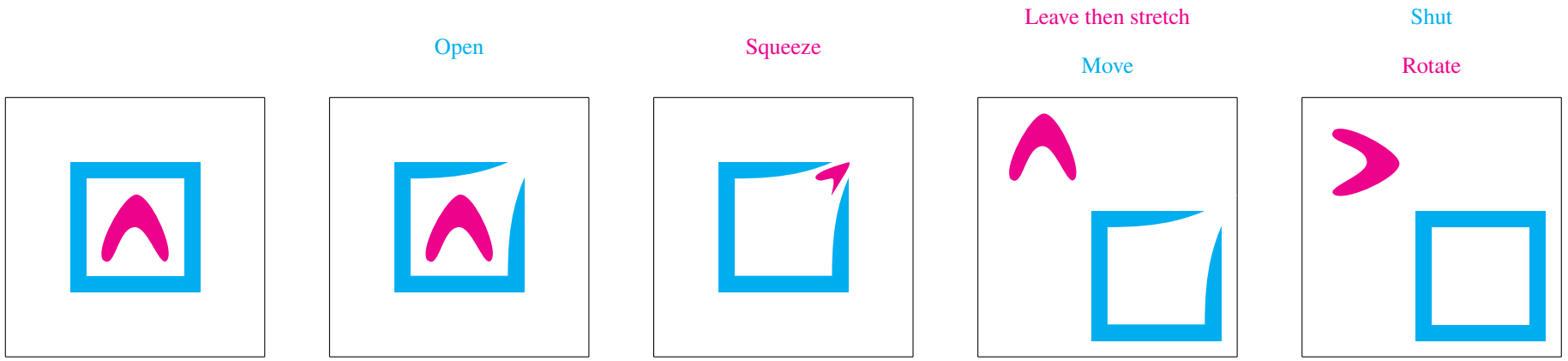


Parallel composition of motions



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

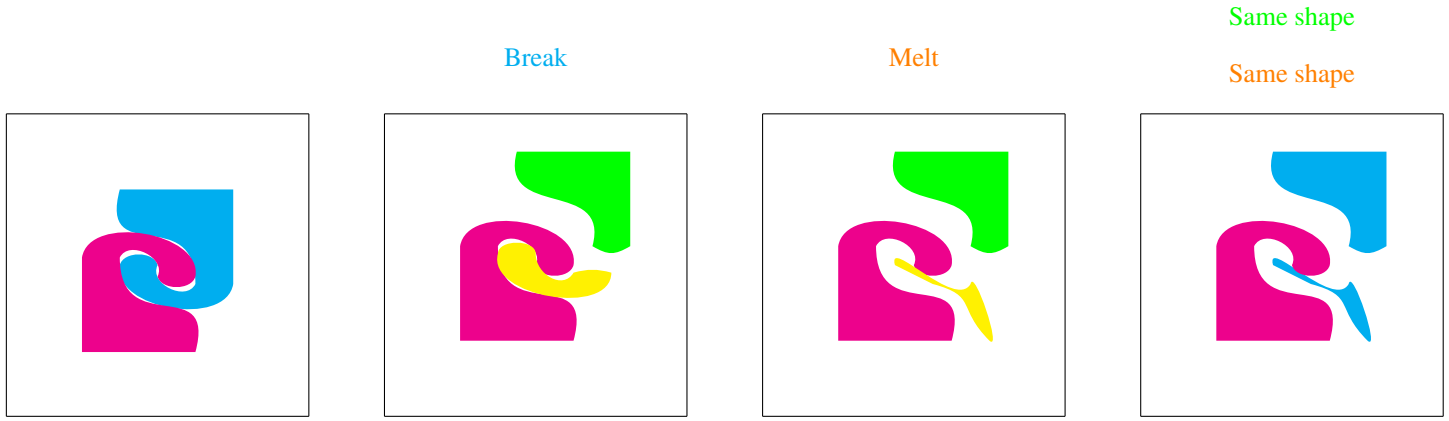
arbitrary homotopies allow deformations, so our verb gates allow shapes to shrink and open and bend in the process of a homotopy, as long as at the end they arrive at a rigid displacement of their original form.



Enclosed

Not 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.



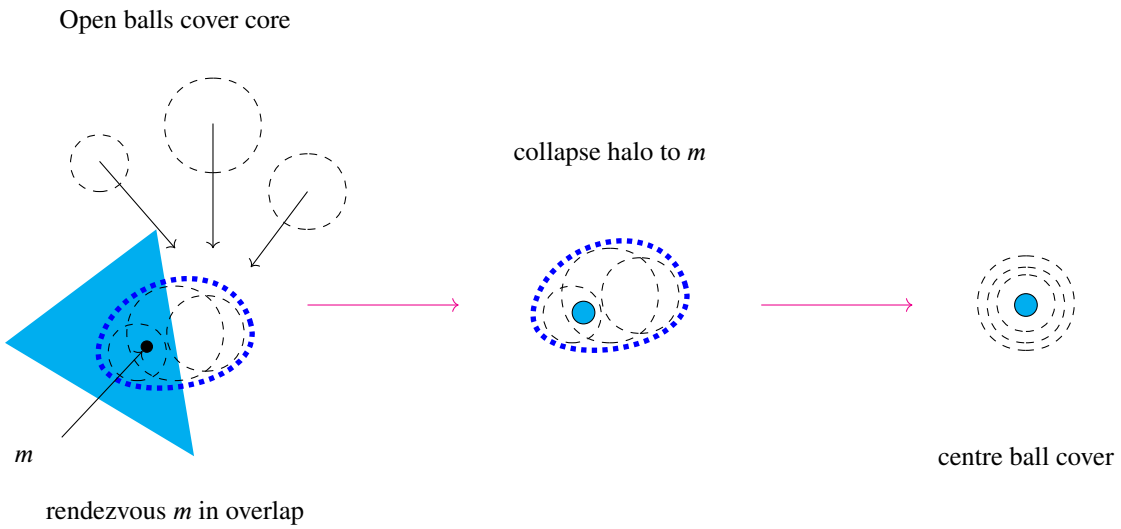
Interlocked

Not interlocked

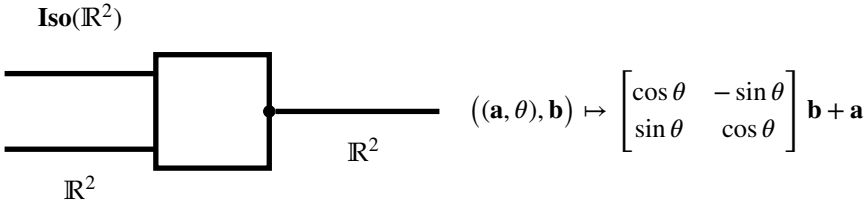
We provide the following construction as a general recipe to construct homotopies between spiders.

Construction 4.6.11 (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

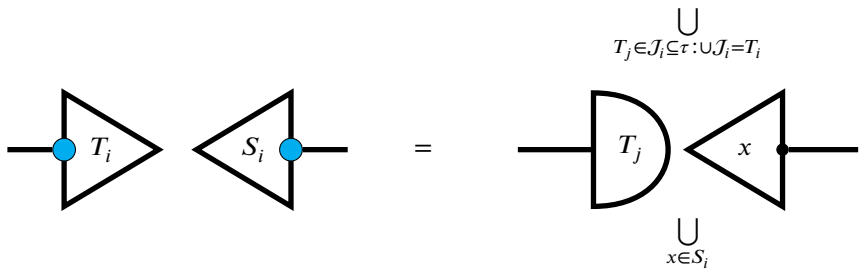
components, we gather all of the balls at a meeting point m (to be determined shortly.) Intuitively we can illustrate this process as follows:



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:



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.

4.7 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.

4.7.1 The category *ContRel*

Proposition 4.7.1 (**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**. □

4.7.2 Symmetric Monoidal structure

Proposition 4.7.2. (**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 $(R \times S)^\dagger = R^\dagger \times S^\dagger$:

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

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

THE NATURAL ISOMORPHISMS ARE INHERITED FROM **REL**. We will be explicit with the unitor, but for the rest, we will only check that the usual isomorphisms from **Rel** are continuous in **ContRel**. To avoid bracket-glut, we will vertically stack some tensored expressions.

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

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

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

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

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

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

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

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

MONOIDAL CLOSURE: Here is the evaluator.

placeholder

4.7.3 Rig category structure

Definition 4.7.3 (Biproducts and zero objects)**.** A *biproduct* is simultaneously a categorical product and coproduct. A *zero object* is both an initial and a terminal object. **Rel** has biproducts (the coproduct of sets equipped with reversible injections) and a zero object (the empty set).

Proposition 4.7.4. 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 4.7.5. **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 4.7.6. 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.

*4.7.4 **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 4.7.7 (Any relation R between discrete topologies is continuous)**.** *Proof.* All subsets in a discrete topologies are open. □

Definition 4.7.8 (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 4.7.7.

Evidently identities and associativity of composition are preserved.

Definition 4.7.9 (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 4.7.10 ($RL = 1_{\mathbf{Rel}}$)**.** 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 4.7.11 (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 4.7.12 (Coarsening is a continuous relation)**.** Let X^σ be coarser than X^τ. The identity relation on underlying points $X^\tau \xrightarrow{1_X} X^\sigma$ is then a continuous relation.

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

Proposition 4.7.13 ($L \dashv R$)**.** *Proof.* We verify the triangular identities governing the unit and counit of the adjunction, which we first provide. By Lemma 4.7.10, we take the natural transformation $1_{\mathbf{Rel}} \Rightarrow RL$ we take to be the identity morphism:

$$\eta_X := 1_X$$

The counit natural transformation $LR \Rightarrow 1_{\mathbf{ContRel}}$ we define to be a coarsening, the existence of which in **ContRel** is granted by Lemma 4.7.12.

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

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

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

THE USUAL FORGETFUL FUNCTOR FROM **CONTREL** TO **LOC** HAS NO LEFT ADJOINT

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

Reminder 4.7.14 (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 4.7.15. Here are informal intuitions to ease the definition. The lattice of open sets of a given topology ordered by inclusion forms a frame – observe the analogy "arbitrary unions" : "all joins" :: "finite intersections" : "finite meets". Closure under arbitrary joins guarantees a maximal element corresponding to the open set that is the whole space. So frames are a setting to speak of topological structure alone, without referring to a set of underlying points, hence, pointless topology. Observe that in the definition of continuous functions, open sets in the *codomain* must correspond (uniquely) to open sets in the *domain* – so every continuous function induces a frame homomorphism going in the opposite direction that the function does between spaces, hence, to obtain the category **Loc** such that directions align, we reverse the arrows of **Frm**. Observe that continuous relations induce frame homomorphisms in the same way. These observations give us insight into how to construct the free and forgetful functors.

Definition 4.7.16 (U : **ContRel** → **Loc**)**.** On objects, U sends a topology X^τ to the frame of opens in τ, which we denote $\hat{\tau}$.

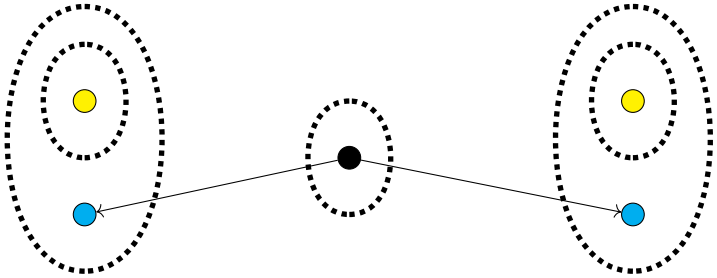
On morphisms $R : X^\tau \to Y^\sigma$, the corresponding partial frame morphism $\hat{\tau} \leftarrow \hat{\sigma}$ (notice the direction reversal for **Loc**), we define to be $\{(U_{\in\sigma}, R^\dagger(U)_{\in\tau}) \mid U \in \sigma\}$. We ascertain that this is (1) a function that is (2) a frame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

Proposition 4.7.17 (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 4.4.14, 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.

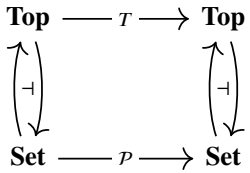
*4.7.5 Why not Span(**Top**)?*

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.



*4.7.6 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: $A \overset{f}{\to} \mathcal{P}B \overset{\mathcal{P}g}{\to} \mathcal{P}\mathcal{P}C \overset{\mu_{\mathcal{P}C}}{\to} \mathcal{P}C$. So it is reasonable to investigate whether there is a monad T on **Top** such that K_T is equivalent to **ContRel**. We observe that the usual free-forgetful adjunction between **Set** and **Top** sends the former to a full subcategory (of continuous functions between discrete topologies) of the latter, so a reasonable coherence condition we might ask for the putative monad T to satisfy is that it is related to $\mathcal{P}$ via the free-forgetful adjunction. This amounts to asking for the following commutative diagram (in addition to the usual ones stipulating that T and $\mathcal{P}$ are monadic):



This condition would be nice to have because it witnesses $K_{\mathcal{P}}$ as precisely K_T restricted to the discrete topologies, so that T really behaves as a conservative generalisation of the notion of relations to accommodate topologies. As a consequence of this condition, we may observe that discrete topologies $X^\star$ must be sent to discrete topologies on their powerset $\mathcal{P}X^\star$. In particular, this means the singleton topology is sent to the the discrete topology on a two-element set; $T\bullet = \mathbf{2}$. This sinks us. We know from Proposition 4.4.4 that the continuous relations $X^\tau \to \bullet$ are precisely the open sets of τ, which correspond to continuous functions into Sierpiński space $X^\tau \to \mathbb{S}$, and $\mathbb{S} \neq \mathbf{2}$.

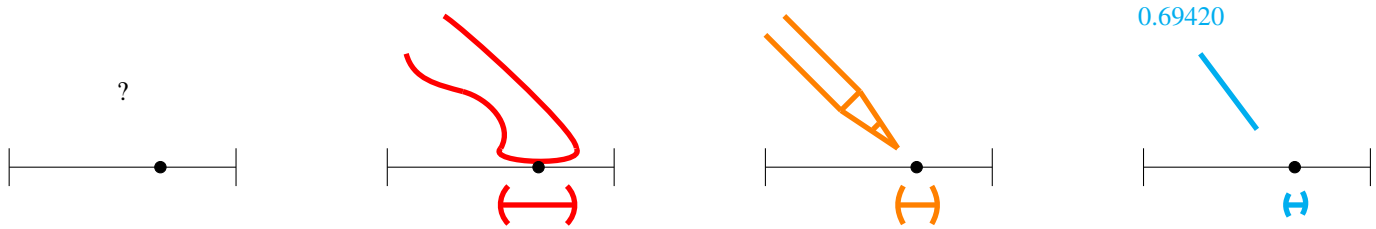
4.7.7 Where is the topology coming from?

It is category-theoretically natural to ask whether **ContRel** is "giving topology to relations" or "powering up topologies with relations", but we have explored those techniques and it doesn't seem to be that. It is possible that the failure of these regular avenues may explain why I had such difficulty finding any trace of **ContRel** in the literature. However, we do have a free-forgetful adjunction between **ContRel** and **Rel**, and if we focus on this, it *is* possible to crack the nut of where topology is coming from with enough machinery; here is one such sketch. Observe that the forgetful functor looks like it could be a kind of fibration, where the elements of the fibre over any set A in **Rel** correspond to all possible topologies on A. Moreover, these topologies may be partially ordered by coarseness-fineness to form a frame (though a considering it a preorder will suffice.) The fibre over a relation $R : A \to B$ is all pairs of topologies τ, σ such that R is continuous between A^τ and B^σ. The crucial observation is that if R is continuous between τ and σ, then R will be continuous for any finer topology in the domain, $\tau \leq \tau'$, and any coarser topology in the codomain $\sigma' \leq \sigma$; that is, the fibre over R displays a boolean-valued profunctor between preorders. So **ContRel** can be viewed as the display category induced by a functor **Rel** $\to \mathcal{C}$, where $\mathcal{C}$ is a category with preorders for objects and boolean-enriched profunctors as morphisms, and the functor encodes topological data by sending sets in **Rel** to preorders of all possible topologies, and relations to profunctors. I have deliberately left this a sketch because it doesn't seem worth it to view something so simple in such a complex way.

4.7.8 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 we talk about concepts, and the 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 we might tell someone else about where this point is. We could point at it with a pudgy appendage, or the tip of a pencil, or give some finite decimal approximation.



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.

5

Sketches of the shape of language

5.1 Lassos for generalised anaphora

A PROBLEM ARISES WHEN ANYTHING CAN BE A NOUN WIRE. By linguistic introspection, we realise we must account for *Entification* and *Processising* – the process of turning non-nouns into noun-entities and back again. If we think about English, we find that just about any word can be turned into a noun and back again (e.g. `run` by gerund to `running`, `quick` by a suffix to `quickness`, and even entire sentences `Bob drinks Duvel` can become a noun `the fact that Bob drinks Duvel`).

This consideration carries some linguistic interest as well. In the usual treatment of anaphora resolution, pronouns refer to nouns, for instance: `Bob drinks a beer. It is cold.`, where `it` refers to the beer. But there are situations where pronouns can point to textual data that are not nouns. For instance: `Jono was paid minimum wage. He didn't mind it.`, where `it` would like to refer to something like `the fact that Jono was paid minimum wage`. While there are extensions of discourse reference theory to accommodate event structures [], the issue at hand is that pronouns in the appropriate context seem to be able to refer to *any meaningful part of text*. For example, `The tiles were grey. It was a particularly depressing shade.`, where `it` seems to refer just to the entified adjective `the greyness (of the tiles)`. Or, `Alice's cake was tastier than Bob's, but it wasn't enough so for the judges to decide unanimously.`, where *it* seems to refer the entified tastiness differential of `tastier`: `the difference in tastiness between Alice and Bob's cakes`.

Since we have so far built up a theory around noun-wires as first-class citizens, these observations present nontrivial mathematical constraints for interpretations of text circuits. Now we try to interpret these constraints in mathematical terms, staying within the graphical confines we have established in **TopRel** as much as possible. Let us denote the noun-wire type by Ξ. First we observe that any finite collection of noun wires $\bigotimes^n \Xi$ has to be *encodable* in a single noun wire Ξ, because we can always interpose with `and`. We take this to mean that there will exist morphisms such that:

placeholder

Second, for any word-gate w of grammatical type $\mathfrak{g}$, we ought to have noun-states and an evaluator process that witness entification and processising:

placeholder

Second-and-a-half, any morphism (or "meaningful part of text") $T \in \bigotimes^n \Xi\,,\ \bigotimes^m \Xi$ for any $n, m \in N$ – has to be encodable as a state of Ξ. This is expressed as the following graphical condition:

placeholder

Condition two-and-a-half follows if the former conditions are met, provided that all text circuits are made up of a fixed stock of grammatical-gate-types:

placeholder

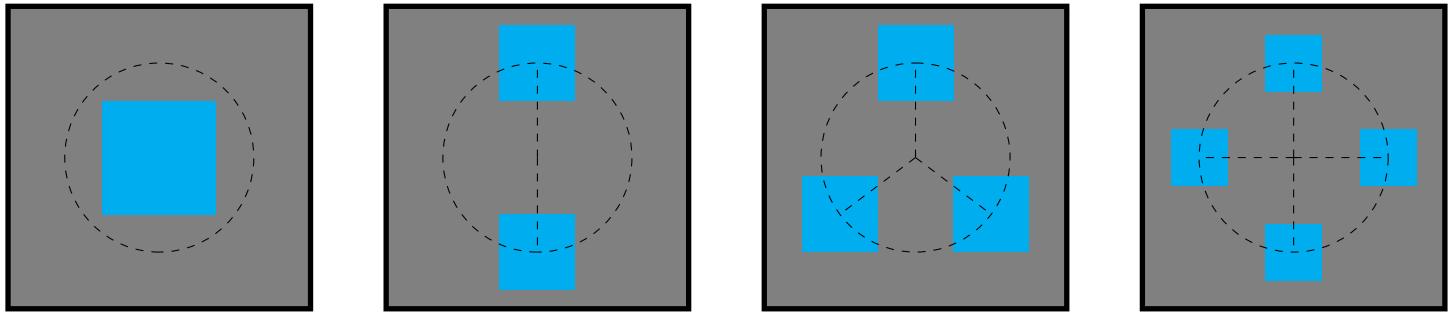
If we have all the above, then we can grab any part of a circuit and turn it into a noun. We can notate this using a *lasso*.

Recall that Lassos – a graphical gadget that can encode arbitrary morphisms into a single wire – can be interpreted in a monoidal computer. Recall that monoidal computers require a universal object Ξ. Here we show how in **TopRel**, by taking $\Xi := \square$ the open unit square, we have a monoidal computer in **Rel** restricted to countable sets and the relations between them.

We will make use of sticky spiders. We have to show that; ▦ has a sticky-spider corresponding to every countable set; how there is a suitable notion of sticky-spider morphism to establish a correspondence with relations; what the continuous relations are on ▦ that mimick various compositions of relations.

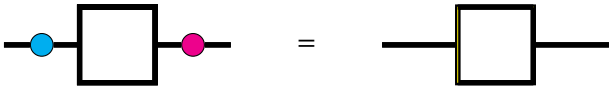
Proposition 5.1.1 ($(0,1)\times(0,1)$ splits through any countable set X)**.** For any countable set X, the open unit square ▦ has a sticky spider that splits through $X^\star$.

Proof. The proof is by construction. We'll assume the sticky-spiders to be mereologies, so that cores and halos agree. Then we only have to highlight the copiable open sets. Take some circle and place axis-aligned open squares evenly along them, one for each element of X. The centres of the open squares lie on the circumference of the circle, and we may shrink each square as needed to fit all of them.

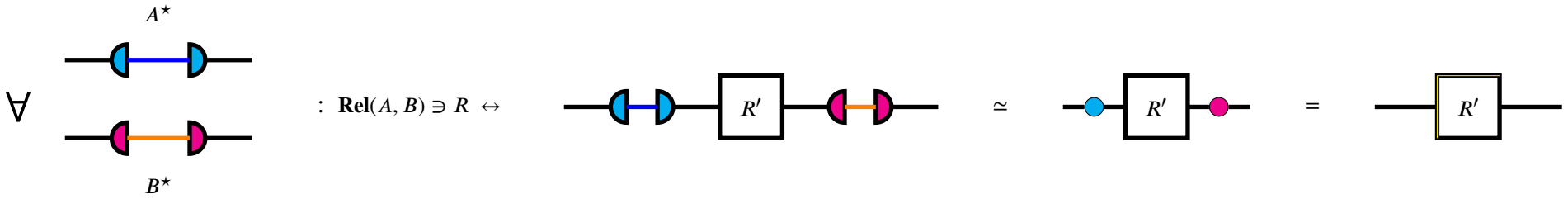


□

Definition 5.1.2 (Morphism of sticky spiders)**.** A morphism between sticky spiders is any morphism that satisfies the following equation.

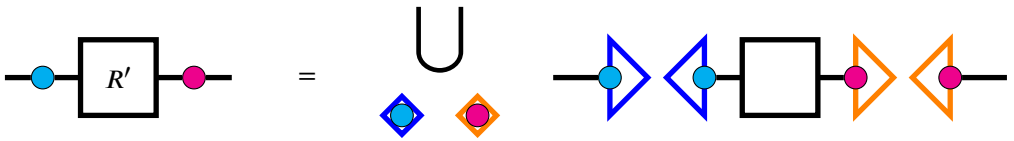


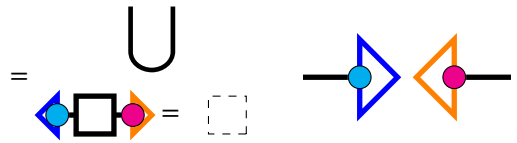
Proposition 5.1.3 (Morphisms of sticky spiders encode relations)**.** For arbitrary split idempotents through $A^\star$ and $B^\star$, the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \to B$.



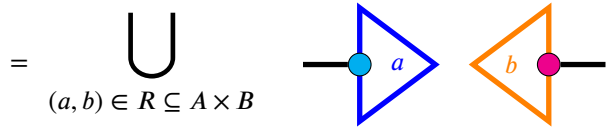
Proof.

($\Leftarrow$) : Every morphism of sticky spiders corresponds to a relation between sets.

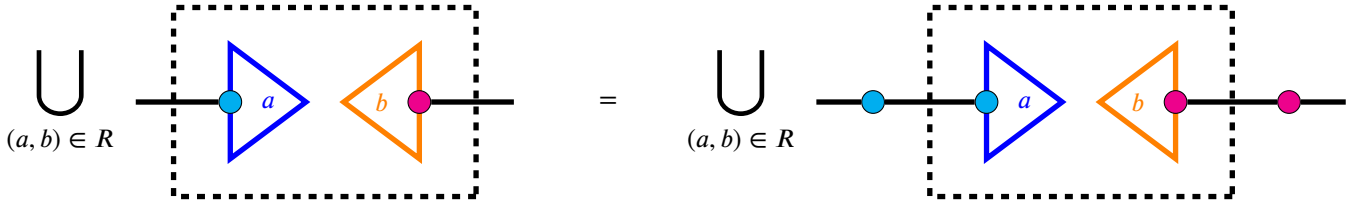




Since (co)copiables are distinct, we may uniquely reindex as:

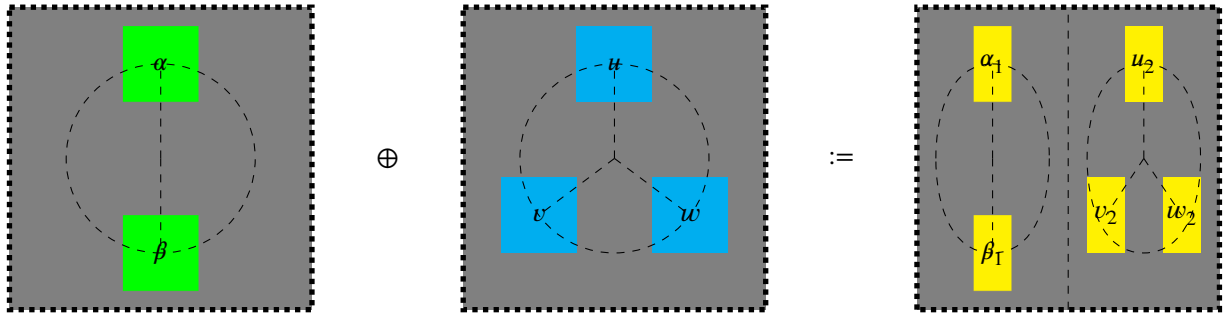


($\Rightarrow$) : By idempotence of (co)copiables, every relation $R \subseteq A \times B$ corresponds to a morphism of sticky spiders.



□

Construction 5.1.4 (Representing sets in their various guises within ▦)**.** We can represent the direct sum of two ▦-representations of sets as follows.

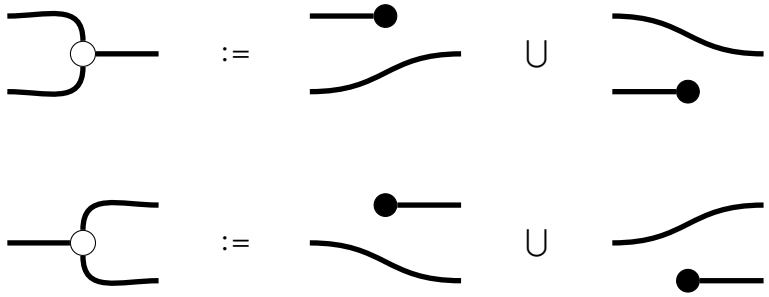


The important bit of technology is the homeomorphism that losslessly squishes the whole unit square into one half of the unit square. The decompressions are partial continuous functions, with domain restricted to the appropriate half of the unit square.

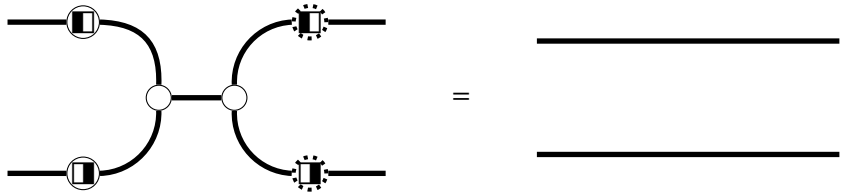
$(x, y) \mapsto (\frac{x}{2}, y)$ $\quad (x, y) \mapsto (\frac{x+1}{2}, y)$ $\quad (x, y)|_{x<\frac{1}{2}} \mapsto (2x, y)$ $\quad (x, y)|_{x>\frac{1}{2}} \mapsto (2x-1, y)$

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

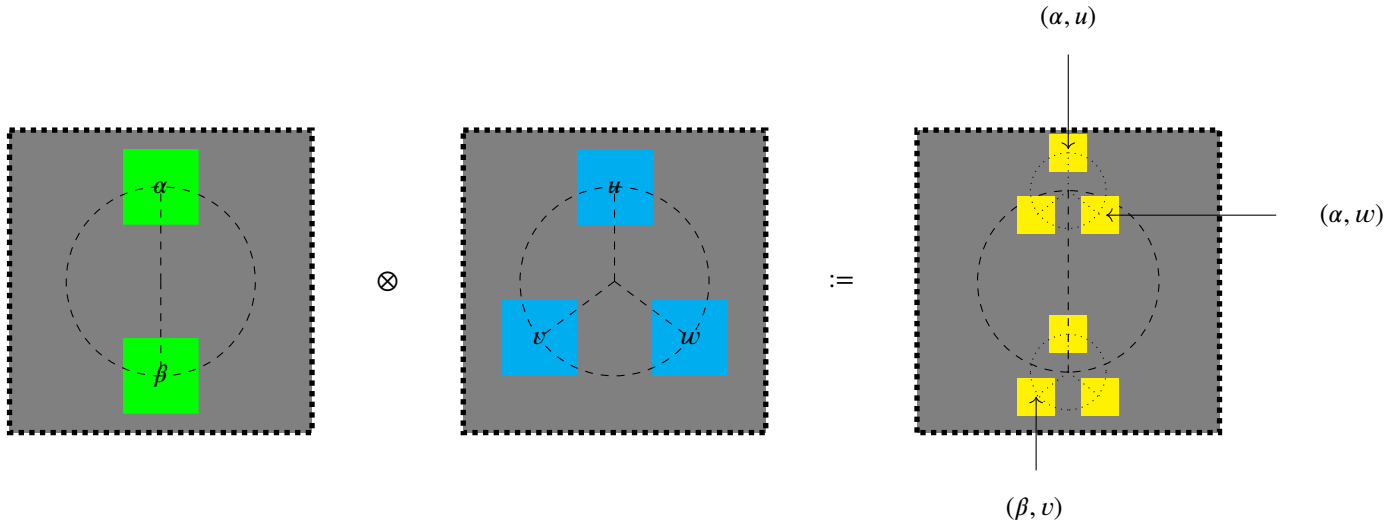
Now, to put the two halves together and to take them apart, we introduce the following two relations. In tandem with the squishing and stretching we have defined, these will behave just as the projections and injections for the direct-sum biproduct in **Rel**.



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

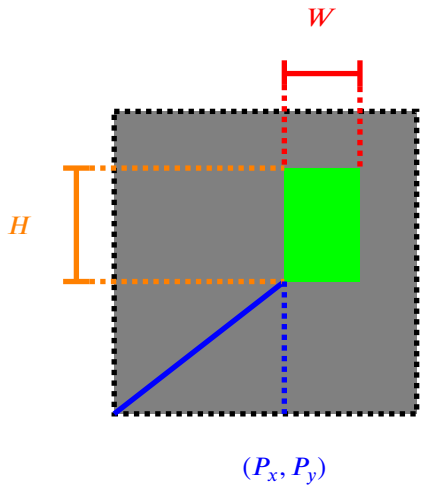


We encode the tensor product $A \otimes B$ of representations by placing copies of B in each of the open boxes of A.

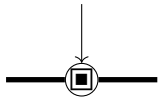


The important bit of technology here is a family of homeomorphisms of ▦ parameterised by axis-aligned open boxes. We depict the parameters outside the body of the homeomorphism for

clarity. The squish is on the left, the stretch on the right.

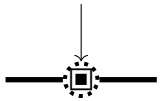


(W, H, P_x, P_y)



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

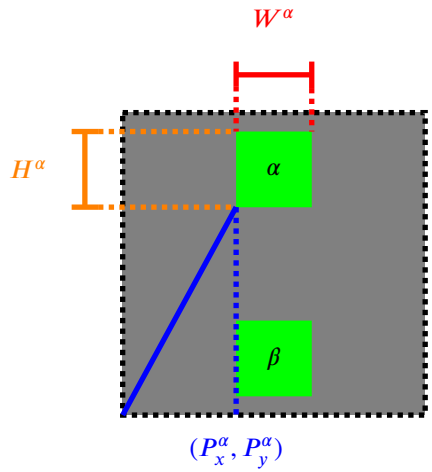
(W, H, P_x, P_y)



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

Now, for every representation of a set in ▦ by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a squish-stretch homeomor-

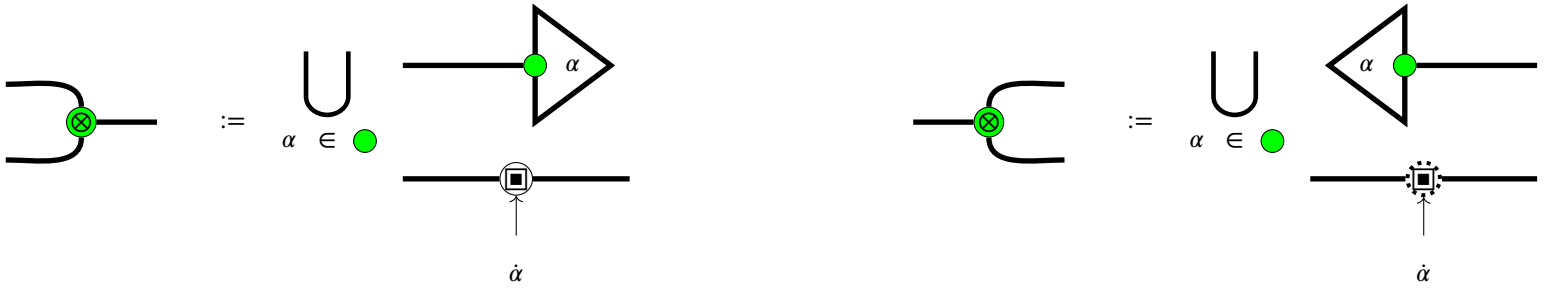
phism via the parameters of the open box, which we notate with a dot above the name of the element.



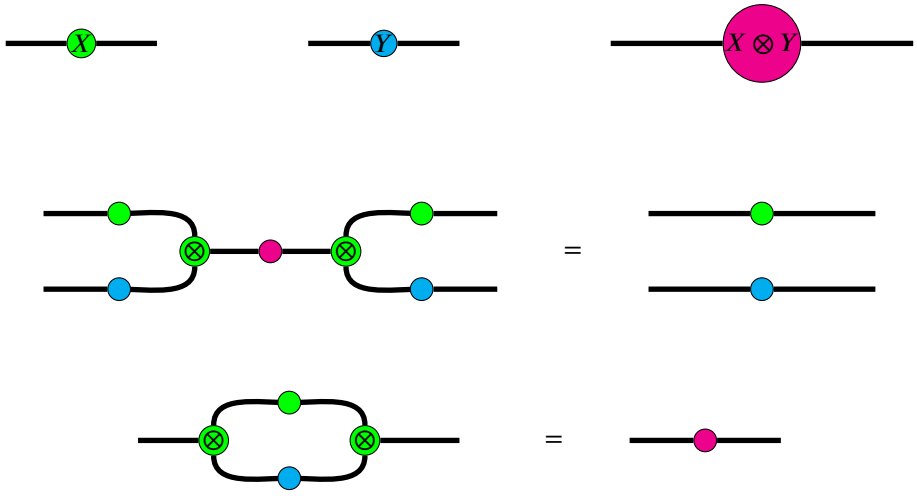
$$\dot{\alpha} = (W^\alpha, H^\alpha, P_x^\alpha, P_y^\alpha)$$

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

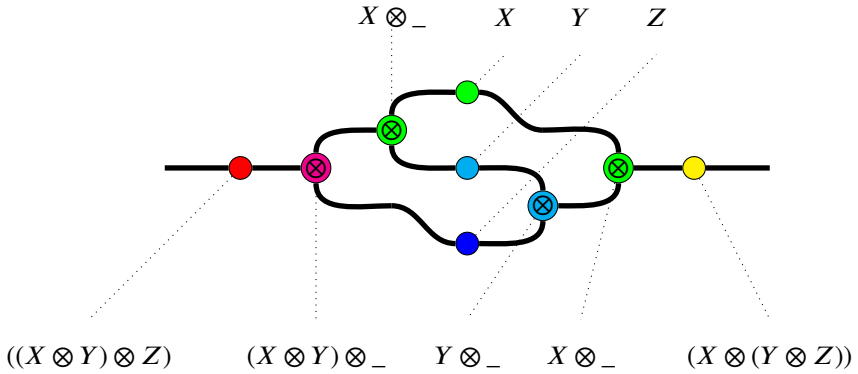
Now we can define the "tensor X on the left" relation $\_ \to X \otimes \_$ and its corresponding cotensor.



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

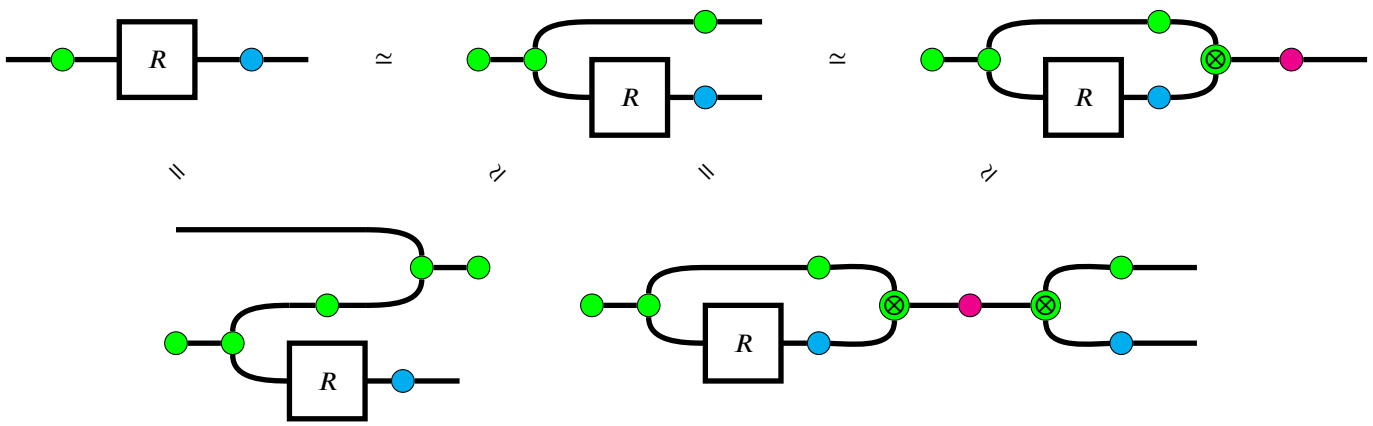


And by construction, the (co)tensors and (co)pluses interact as we expect, and they come with all the natural isomorphisms between representations we expect. For example, below we exhibit an explicit associator natural isomorphism between representations.

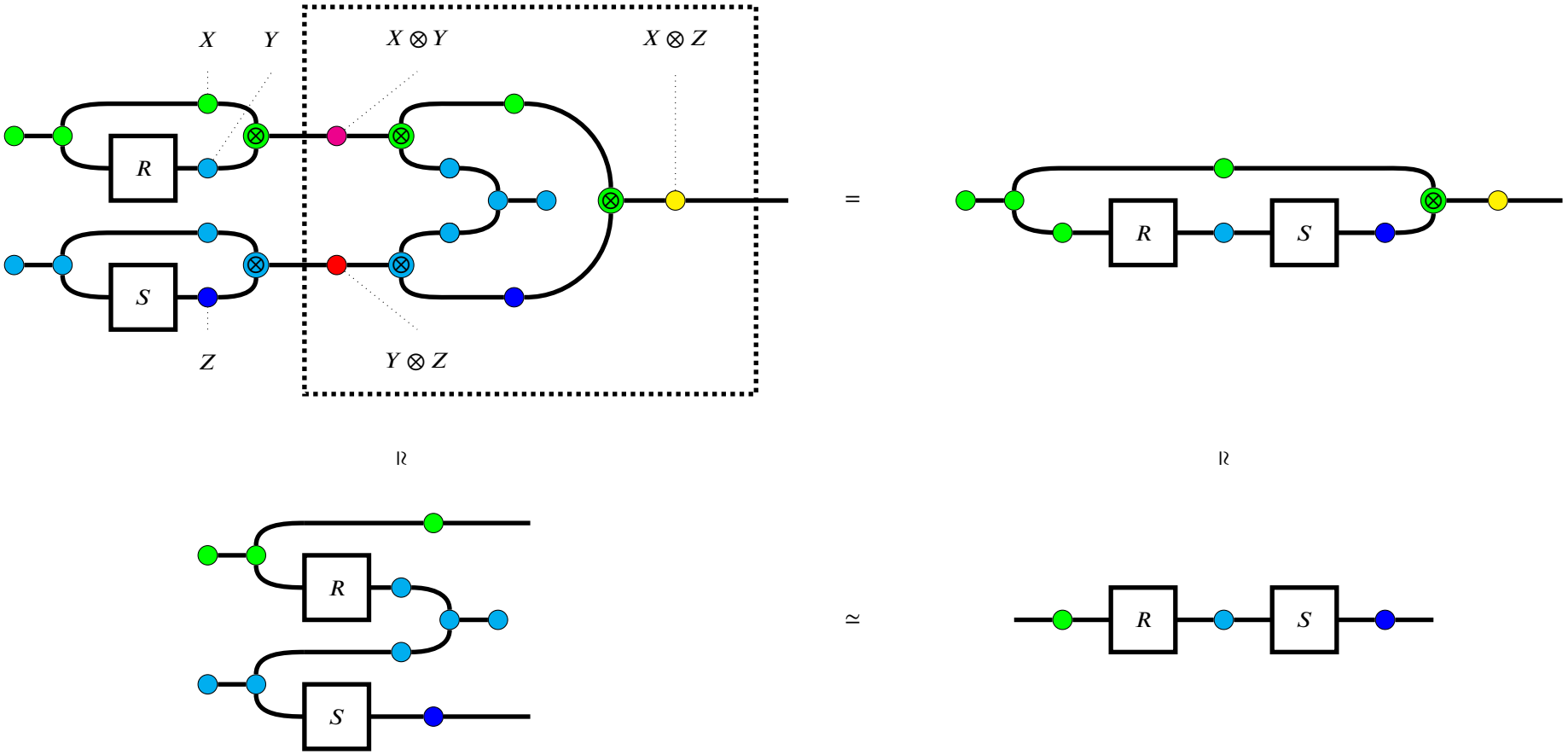


Construction 5.1.5 (Representing relations between sets and their composition within ▦)**.** With all the above, we can establish a special kind of process-state duality; relations as processes

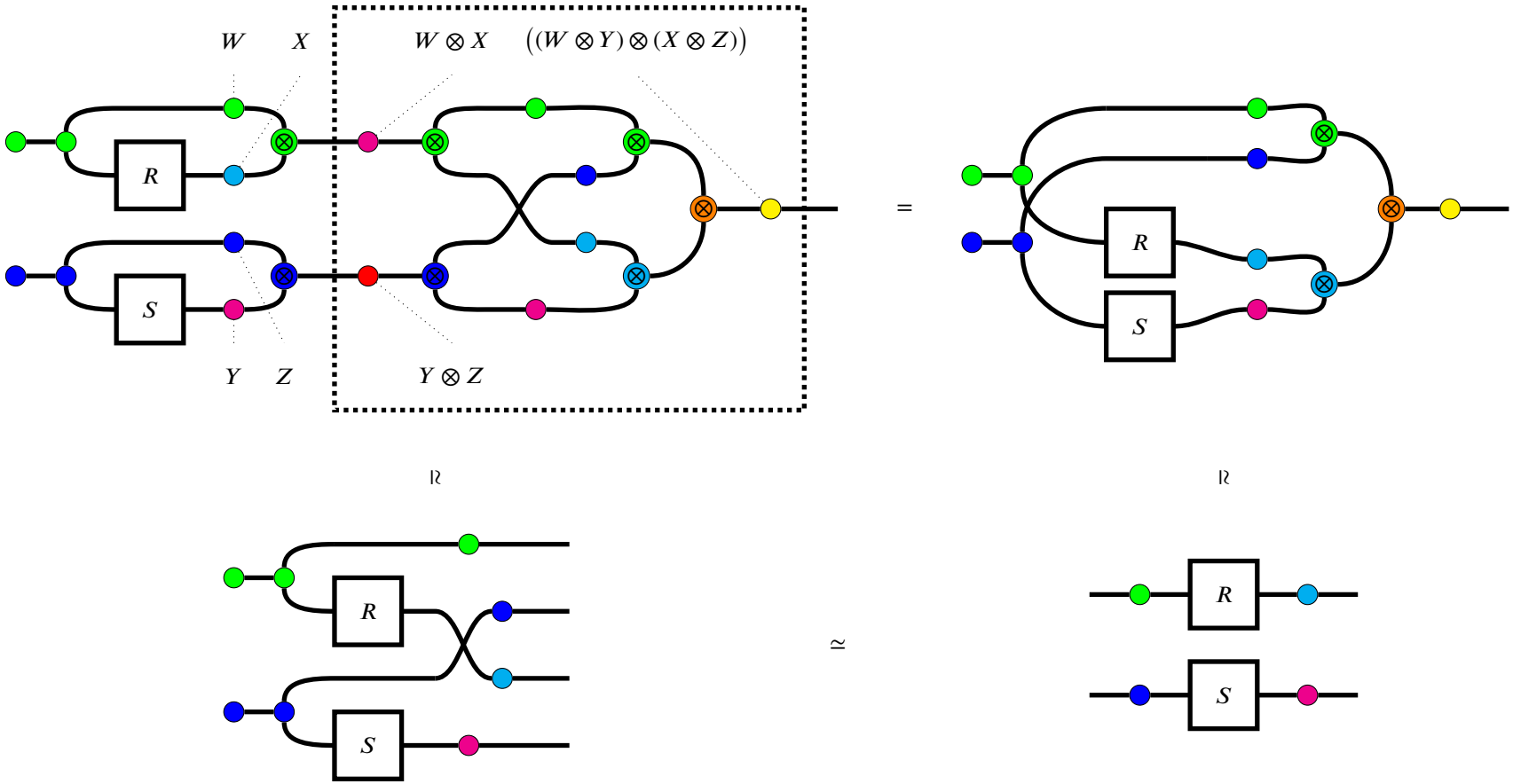
are isomorphic to states of ▦, up to the representation scheme we have chosen.



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



And we also know how to take the parallel composition of relations by tensors.



5.2 (Im)possibility results for learning text circuits from data

So far, we have been working process-theoretically, using equations between processes to specify their behaviour. It is natural to ask whether it is possible to realise each process in a process theory as a neural net, using the equations as training criteria so that the neural nets jointly model a process theory. This approach is worth pursuing to combine the benefits and ease of data-driven learning with the modularity and explainability benefits of process theories. Moreover, the onus is on us to demonstrate that text circuits can be learnt in this way, or else we would be no better off in terms of a practical theory of language for the age of big data.

In this sketch, we briefly introduce neural nets diagrammatically, along with the *Universal Approximation Theorem*, which, along with variants for different architectures, states that for any dimension m and any $\epsilon > 0$, there exists a neural net that approximates any continuous function $\mathbb{R}^m \to \mathbb{R}$ on a compact subset of the domain $\mathbb{R}^m$ within a discrepancy of ϵ. Then we introduce the notion of approximability for PROPs, and we observe that not all PROPs are approximable in terms of smooth functions of the form given by the universal approximation theorem. So we restrict our attention to PROPs for basic text circuits, which we demonstrate are suitable for certain learning tasks. We prove that basic text circuit PROPs of bounded depth and width – a notion we will define – are approximable; in other words, that text circuits work in principle alongside data-driven techniques. We close with a discussion of limitations and extensions. We give a corollary that finitely generated subcategories of **FinSet** are realisable as ensembles of deterministic neural nets, and we show how introducing probabilistic states extends the situation to **FinRel**. We formalise an observed tension between the space-resource demands of deterministic representations and unbounded compositionality by a no-go conjecture.

5.2.1 Approximating Text Circuits with deterministic neural nets

There is a lot to be gained from a process-theoretic view of interacting ensembles of neural nets. For a simple example, consider that an autoencoder is precisely a pair of neural nets trained cooperatively encode a large input space into a small latent space and decode the original input from the latent space. Diagrammatically, this amounts to asking for the equations of a split idempotent to be treated as training conditions for a pair of processes.

autoencoder

If that's what we can do with a pair of equations, what can we do with an arbitrary PROP? We first need to decide what qualifies as a valid interpretation the generators of a PROP in terms of neural nets. Not just any functor will do, because we want to rule out trivial solutions that map all processes to constant functions. We also need to put in some work to interpret what equality of processes should mean in the setting of neural nets.

Definition 5.2.1 (Approximating a (coloured) PROP)**.** An $(\epsilon^{=}, \epsilon^{\neq})$-approximation of a finitely presented coloured PROP $\mathfrak{P}$ is a strict symmetric monoidal functor $\mathcal{T}$ that interprets $\mathfrak{P}$ in the (cartesian) symmetric monoidal subcategory of **Top** generated by Euclidean spaces with the usual metric as wires equipped with cartesian copy and delete, along with neural nets as processes. As a nontriviality condition, $\mathcal{T}$ must send each wire colour in $\mathfrak{P}$ to a Euclidean space of finite positive dimension. Equality relations presented in $\mathfrak{P}$ are interpreted as $\epsilon^{=}$-closeness by $\mathcal{T}$, i.e. if $\mathfrak{P}$ stipulates that $f = g$ for $f, g : A \to B$, then we have the following inequality in the metric of $\mathcal{T}B$:

$$\forall \mathbf{x}_{\in \mathcal{T}A} : d_{\mathcal{T}B}\big(\mathcal{T}f(\mathbf{x}), \mathcal{T}g(\mathbf{x})\big) \leq \epsilon^{=}$$

Any PROP that equates generators directly is redundant, and we can without loss of generality restrict consideration to PROPs where each generator is implicitly assumed to be distinct. We interpret inequality as $\epsilon^{\neq}$ farness, i.e., for all pairs of generators f, g of the same type $A \to B$, we ask that $\mathcal{T}$ satisfies:

$$\exists \mathbf{y}_{\in \mathcal{T}A} : d_{\mathcal{T}B}\big(\mathcal{T}f(\mathbf{y}), \mathcal{T}g(\mathbf{y})\big) \geq \epsilon^{\neq}$$

Since the category is cartesian monoidal, states are points in euclidean space, and the above definition specialise to treating points as "equal" if they are $\epsilon^{=}$-close and "inequal" if they are $\epsilon^{\neq}$-far. We choose to treat the determination of equality and inequality as separate semidecidable procedures because "equality" as we have defined it is not necessarily transitive, but we

can recover a form of bounded transitivity by making $\epsilon^{=}$ very small compared to $\epsilon^{\neq}$, so that equality is testable within a tolerance of $\epsilon^{\neq}$, granting $\frac{\epsilon^{\neq}}{\epsilon^{=}}$-fold transitivity. We can always recover decidable "equality" at the expense of transitivity by setting $\epsilon^{=} = \epsilon^{\neq}$. With that out of the way, we observe that since the target category of deterministic neural nets is cartesian monoidal, not all PROPs are approximable.

Example 5.2.2 (Not all PROPs are approximable)**.** We take the snake equations as an example. The PROP generating the snake equation is as follows:

$$snakeprop$$

Since we are dealing with a cartesian monoidal category, the cup can only be interpreted as a pair of points, and the cap can only be a pair of deletes []. The only Euclidean space in which the identity is equal to a constant map is the singleton zero-dimensional space.

$$trivialityproof$$

So nondeterminism is a necessary but possibly insufficient condition for the realisation of general PROPs. Not all is lost; if we restrict our consideration to well-behaved PROPs, such as those of simple text circuits, then we can get somewhere eventually.

Definition 5.2.3 (Basic Text Circuit PROP)**.** A *basic text circuit PROP* has two colours of wires, N for "noun" and A for "answer". The generators fall under four main families. *Nouns* have type $1 \to N$. *Gates* have type $\bigotimes^k N \to \bigotimes^k N$ for some positive k. *Queries* have type $\bigotimes^k N \to A$ for some positive k. *Answers* have type $1 \to A$.

$$states, gates, queries, answers$$

The relations of a text circuit fall under three families. *Axioms* are equations between pairs of nonempty composites of gates; the only kind we disallow is an equality between two generators.

$$axiom$$

Instances are equations between a composite of nouns and gates and a single query on the left – a *datum* – and an answer on the right – a *label*.

$$instance$$

In addition, we ask for a special generator with a non-finite family of rules to enforce a coherence condition; we decide that distinct nouns should maintain their identity no matter what relations they participate in. The generator is *Name*, of type $N \to N$, and its relations are such that applying *Name* to any noun-wire that traces back to a noun will return that noun.

$$name$$

Example 5.2.4 (How basic text circuits PROPs may be used in practice)**.**

A limitation that arises from the interaction of the definition of approximability and the universal approximation theorem is that any compact subset of euclidean space can be covered by finitely many ϵ-balls. This means that the universal approximation theorem is unable to provide us guarantees if we want potentially unboundedly large text circuits, but we might reasonably expect compositional behaviour up to some notion of text circuits with bounded width and depth, which we state as follows.

Definition 5.2.5 (Bounded width and depth)**.** We say that a basic text circuit PROP $\mathfrak{T}$ is *terminating* if all of its axiom relations can be equipped with directions so that applying directed equality rewrites to any diagram (necessarily finite) yields a finite set of equal diagrams. As an easy example, a basic text circuit PROP with a single idempotent gate is terminating when we equip the idempotence relation with a direction that reduces the number of gates; we don't want to deal with cases where equalities explode to infinity.

$$idempotentreduce$$

Observe that if $\mathfrak{T}$ is terminating, then applying axioms to the datum of any instance relation will also yield a finite set of equal diagrams, and each instance diagram may be rewritten by isotopies so that parallel gates are displaced so that each gate occupies a distinct level, sandwiched by a layer of nouns and query, which we also count as layers. We say that $\mathfrak{T}$ has *bounded depth* $d \in \mathbb{N}$ if the maximum depth in layers obtained in this way from any datum in $\mathfrak{T}$ is bounded above by d. The case of width is simpler, because axioms cannot change the number of wires in a datum, which is the same as the number of nouns; we say that $\mathfrak{T}$ has *bounded width* $w \in \mathbb{N}$ if the maximum number of nouns that occur in any datum is bounded above by w.

Even this is not enough. Another issue arises from the interaction of approximability with the nature of cartesian monoidal categories.

Theorem 5.2.6 (Fox's Theorem)**.**

A consequence of Fox's theorem is that in any cartesian monoidal category, we can equip every wire with canonical cocommutative comonoids (copy and delete), and every morphism in a cartesian monoidal category is a cohomomorphism with respect to copy and delete; recall from Section 2.2.4 that this is the diagrammatic definition of deterministic maps. This consequence means that for some text circuit PROPs, approximable-equalities may hold in *any* of their interpretations as deterministic neural nets that are not equalities licensed by the original PROP.

Example 5.2.7.

The deeper essence of this issue is that in a cartesian monoidal category, every wire can at most carry data about its diagrammatic causal past, since equalities of the following sort always hold.

placeholder

This conflicts with the nature of updating representations with text, where later information may force revisions of earlier representations. For a crude example, consider this transcription of a meme about a morning routine:

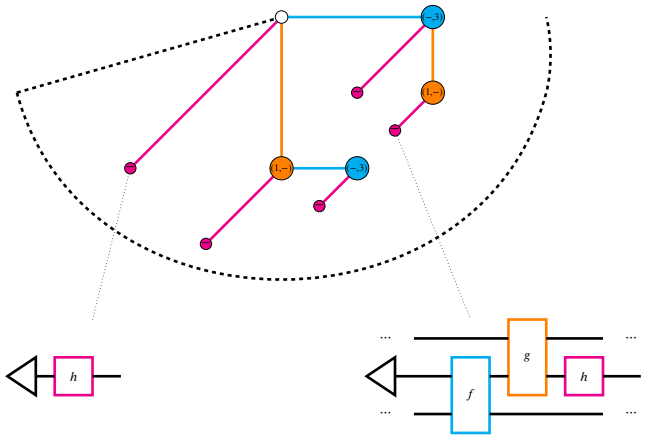
`Wake up.  Take a shit.  Eat.  Get out of bed.  Have breakfast.`

Now I will kill the joke by overthinking it. What happens in our mind's eye if we are constructing a little vignette of events? In this example, the first three clauses take a pedestrian interpretation – `Taking a shit` normally happens in toilets, and the inferred object of `Eat` is breakfast. Clauses four and five require belief revisions. The internal representation of the sequence of events up to those points must be retroactively changed in a manner that is not representable as gates updating entities locally:

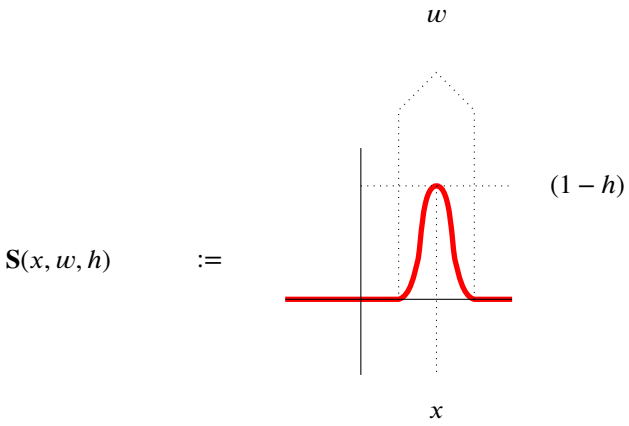
cartoon

So in the deterministic setting, after every local update, we require a *global* coordination gate that gives all representations access to each other's data and a chance for them to revise themselves. To summarise our restrictions so far, we are only dealing with text circuit PROPs, of bounded depth to remain within the guarantees of the universal approximation theorem, and of fixed width as a diagrammatic consequence of having a single global coordinator gate.

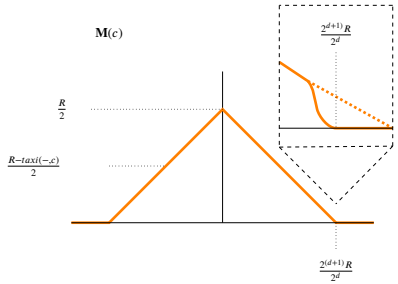
Theorem 5.2.8 (Basic Text Circuit PROPs of bounded depth and fixed width are approximable using a global coordinator)**.** *Proof.* We sketch a construction. The essential idea is to have each noun represent all of the gates it has passed through. The representation is (the monoid analogue of) a Cayley graph of a free group in $\mathbb{R}^{|\mathbf{G}|}$, where if $w \in \mathbb{N}$ is the fixed width, $\mathbf{G}$ is the set of gates where every wire but one is assigned: $\{g(i, j, \cdots, -, \cdots, k) \mid g \in \text{Gates}\ ,\ 1 \leq i, j, \cdots k \leq w\}$. In the illustration below, we can for instance record that a noun wire has passed through a unary gate h as the only argument, or that it has passed through three gates, first a binary f gate as the first argument, sharing the gate with the third wire, then a binary g gate as the second argument, sharing the gate with the first wire, and then a unary h gate. Each distinct generator has its own dimension in Euclidean space.



By asking for each node to be an ϵ-ball, we can test for equality by a spike function.



We don't want equal steps in each direction in the graph, or else all pairs of steps would commute. To reflect the order in which steps occur, we want subsequent steps to be smaller than earlier ones. So for each colour of node in the graph, we add the value of a move function in the desired dimension, parameterised by the initial position of the noun, so that each subsequent step is half the distance of the previous one. To make life easier, we have each move function use the taxicab metric. By bounded depth, there is some $d \in \mathbb{N}$ after which we don't need to carry representations, so we modify the ends of the move function to hit zero early. Setting $R >> 2^d \epsilon$, each move function acts to translate all graph nodes within an R-ball to their destination graph nodes after uniformly applying some generator. We ask that each wire is initialised at least $2R$-far apart, so each wire has their own R-ball neighbourhood in which to encode data as a graph.



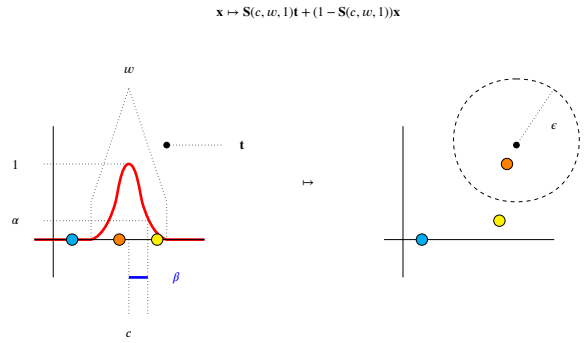
Now we can jointly specify representations of nouns and gates. For example, suppose f is a binary gate, and that $w = 3$. Then there are six shift functions

$$\mathbf{M}_f(-,1), \mathbf{M}_f(-,2), \mathbf{M}_f(-,3), \mathbf{M}_f(1,-), \mathbf{M}_f(2,-), \mathbf{M}_f(3,-)$$

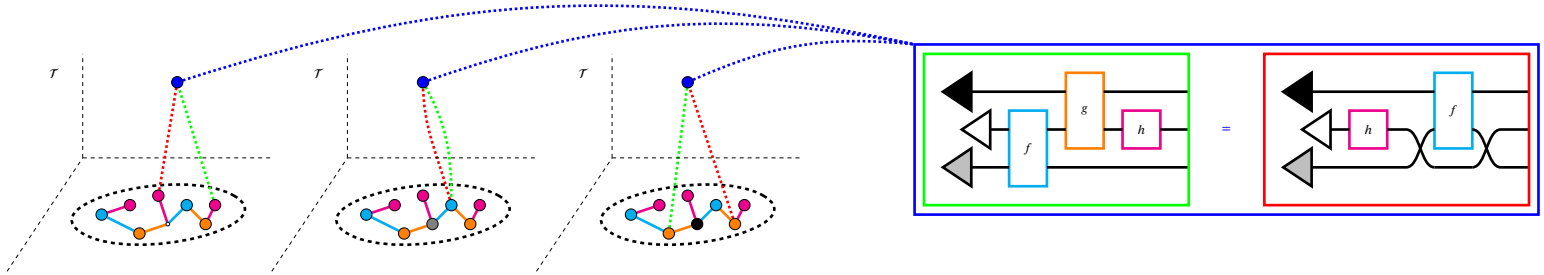
Set an ansatz dimension $\mathcal{A}$ aside, which distinguishes noun wires by indicator-spike functions $\mathbf{1}_1, \mathbf{1}_2, \mathbf{1}_3$ – a continuous analog of one-hot encoding – then let f be:

$$f(x,y) := \begin{cases} x \mapsto x + \sum_{i^x, i^y = 1}^{w} \left(\mathbf{1}_{i^x}(x) \mathbf{1}_{i^y}(y) \mathbf{M}_f(-, i^y) \right) \\ y \mapsto y + \sum_{i^x, i^y = 1}^{w} \left(\mathbf{1}_{i^x}(x) \mathbf{1}_{i^y}(y) \mathbf{M}_f(i^x, -) \right) \end{cases}$$

In prose, the coefficients obtained by the indicator functions of each summand behave as a case guard, identifying the wire-numbers of x, y by sending all other cases to zero. This generalises to representations of gates with higher arity. At this point, we have obtained representations for a free theory of just the gates of bounded-depth and fixed-width text circuits – we have not yet incorporated the potential equalities of the theory. First we deal with the axiom-type equalities. Here is where we make use of a global coordinator gate; where width is w and each noun is represented in some space $\mathbb{R}^J$, the global coordinator is typed $\prod^w \mathbb{R}^J \to \prod^w \mathbb{R}^J$. The global coordinator looks at the local data recorded by each wire, and when it spots an equality in the theory, it pinches free graph nodes together. The pinch function is a modification of the spike and shift functions, moving only points within a tolerance β into a target ϵ-ball.



The global coordinator waits to see if the product state of all the wires satisfies an equality, and when it does, it pinches the agreeing free representations on each wire to a new node along a new dimension – we ask for one dimension for each equivalence class of expressions quotiented by equality in the theory, and here the bounded-depth and fixed-width conditions mean that we only have to add finitely more dimensions in this way.



The target location of each pinch can be chosen to be compatible with the shift functions that generate the graphs – i.e. each equality-class is treated as if it were its own generator with its own shift function. The global coordinator is the sum of all such pinches, and the way to use it is to sandwich it between each gate.

placeholder

Finally, the query morphisms in the instance-type rules of the text circuit PROP are continuous functions that assign each ϵ-ball node a value in a Euclidean answer-space, which can mimick a finite discrete number of cases by spike-indicator functions. All of this is, by construction, an (ϵ, R)-approximation of a text circuit PROP. □

5.2.2 *Text circuits with unbounded depth and width*

For a better understanding of what is possible, we might approach "from above", seeking first a mathematical setting in which we have unrestricted compositionality, and then gradually applying sensible constraints. One such mathematical setting is **Rel**, which we show can faithfully model any finitely presented coloured PROP.

Definition 5.2.9 (Faithful models and expressive completeness)**.** Given a coloured prop $\mathfrak{P}$, a symmetric monoidal category $\mathcal{M}$, and a symmetric monoidal functor $T : \mathfrak{P} \to \mathcal{M}$, we say that T is a *faithful model of* $\mathfrak{P}$ *in* $\mathcal{M}$ if, for all diagrams $f, g \in \mathfrak{P}$, $f =_{\mathfrak{P}} g \iff T(f) =_{\mathcal{M}} T(g)$. Given a collection of props $\{\mathfrak{P}\}$, we say that $\mathcal{M}$ is *expressively complete for* $\{\mathfrak{P}_i\}$ when each $\mathfrak{P}_i$ has a faithful model in $\mathcal{M}$.

Theorem 5.2.10 ($\mathbf{Rel}^{\times}$ is expressively complete for all finitely presented coloured PROPs)**.** *Proof.* Our strategy has two main ideas. First is to take the Lindenbaum-Tarski algebra of diagrams for an arbitrary $\mathfrak{P}$, quotiented by the equivalence relation $=_{\mathfrak{P}}$; this will give faithfulness. Second is to treat the sought functor T for $\mathfrak{P}$ as a category of elements construction, adapted to the symmetric monoidal setting. Let $\mathfrak{P}^{\star}$ denote the finitely presented coloured PROP $\mathfrak{P}$ augmented with new generators that do not obey any relations: a state $\circ\!-$ and effect $-\!\circ$ for each colour C in $\mathfrak{P}$. Let $\mathfrak{P}^{\star}_{\mathbf{LT}}$ denote the set of diagrams of $\mathfrak{P}^{\star}$ quotiented by the equivalence relation obtained by equality $=_{\mathfrak{P}}$ in $\mathfrak{P}$. Let $T(0)$ be the singleton. For each colour $C \in \mathfrak{P}$, let $T(C)$ be the subset of $\mathfrak{P}^{\star}_{\mathbf{LT}}$ that selects all states on C. For each nonempty list of colours $[C_i]$, let $T([C_i])$ be the subset of $\mathfrak{P}^{\star}_{\mathbf{LT}}$ that selects all states on $C_1 \otimes_{\mathfrak{P}} C_2 \otimes_{\mathfrak{P}} \cdots C_i$. For $f : C \to D$ in $\mathfrak{P}$, let $T(f) := \{(\ulcorner\langle c|\urcorner, \ulcorner\langle c| ; f\urcorner) \mid c_{\in T(C)}\}$, where corner notation denotes an equivalence class of states under $=_{\mathfrak{P}}$; as a particular consequence, for a state $\langle C|$ on C, $T(\langle c|) = \ulcorner\langle c|\urcorner \subseteq T(C)$. Note that applying f after any state $\ulcorner\langle c|\urcorner \in T(C)$ yields a state $\ulcorner\langle c| ; f\urcorner \in T(D)$, and that these states include actual states native to $\mathfrak{P}$ and formal states from $\mathfrak{P}^{\star}$ where diagrams from $\mathfrak{P}$ with an output wire typed C have their other wires "capped off" by $\circ\!-$ and $-\!\circ$.

T is a functor; $T(1_C) = 1_{T(C)}$ since $\{(\ulcorner\langle c|\urcorner, \ulcorner\langle c| ; 1_C\urcorner) \mid \ulcorner\langle c|\urcorner_{\in T(C)}\}$ is the identity relation on $T(C)$; $T(f) ;_{\mathbf{Rel}} T(g) = T(f ;_{\mathfrak{P}} g)$ since the relational composite is

$$T(f) ;_{\mathbf{Rel}} T(g) := \{(\ulcorner\langle c|\urcorner, \ulcorner\langle e|\urcorner) \mid c_{\in T(C)}\ ,\ \exists \ulcorner\langle d|\urcorner_{\in T(D)} : \ulcorner\langle c| ; f\urcorner = \ulcorner\langle d|\urcorner \& \ulcorner\langle d| ; g\urcorner = \ulcorner\langle e|\urcorner\}$$

We observe that $\ulcorner\langle d|\urcorner_{\in T(D)} : (\ulcorner\langle c| ; f\urcorner = \ulcorner\langle d|\urcorner \,\&\, \ulcorner\langle d| ; g\urcorner = \ulcorner\langle e|\urcorner)$ implies that $\ulcorner\langle c| ; f ; g\urcorner = \ulcorner\langle e|\urcorner$, so we have $T(f ;_{\mathfrak{P}} g \subseteq T(f) ;_{\mathbf{Rel}} T(g)$. For the other inclusion, we observe that $\ulcorner(\langle c| ; f) ; g\urcorner$ yields the state $\ulcorner\langle c| ; f\urcorner \in T(D)$ in the bracketed expression, thus satisfying the existential quantifier.

The unitors, associators, braidings, and coherences are tedious to write but conceptually trivial, so I will skip them. The tricky part of showing that T is monoidal is providing isomorphisms $T(C) \times T(D) \simeq T(C \otimes_{\mathfrak{P}} D)$. We present them first and comment later.

$$T(C) \times T(D) \to T(C \otimes_{\mathfrak{P}} D) := \{ \left(\begin{pmatrix} \ulcorner \langle c| \urcorner \\ \ulcorner \langle d| \urcorner \end{pmatrix}, \begin{cases} \ulcorner \langle x| \urcorner & \text{if } \exists \langle x|_{\in T(C \otimes_{\mathfrak{P}} D)} : \ulcorner \langle c| ; \multimap_C \urcorner = \ulcorner \langle x| ; (\multimap_C \otimes_{\mathfrak{P}} \multimap_D) \urcorner = \ulcorner \langle d| ; \multimap_D \urcorner \\ \ulcorner \langle c| \otimes_{\mathfrak{P}} \langle d| \urcorner & \text{otherwise} \end{cases} \right) \mid \ulcorner \langle c| \urcorner_{\in T(C)}, \ulcorner \langle d| \urcorner_{\in T(D)} \}$$

$$T(C \otimes_{\mathfrak{P}} D) \to T(C) \times T(D) := \{ \ulcorner \langle x| \urcorner, \begin{cases} \begin{pmatrix} \ulcorner \langle c| \urcorner \\ \ulcorner \langle d| \urcorner \end{pmatrix} & \text{if } \exists \ulcorner \langle c| \urcorner_{\in T(C)} \ulcorner \langle d| \urcorner_{\in T(D)} : \ulcorner \langle x| \urcorner = \ulcorner \langle c| \otimes_{\mathfrak{P}} \langle d| \urcorner \\ \begin{pmatrix} \ulcorner \langle x| ; (1_C \otimes_{\mathfrak{P}} \multimap_D) \urcorner \\ \ulcorner \langle x| ; (\multimap_C \otimes_{\mathfrak{P}} 1_D) \urcorner \end{pmatrix} & \text{otherwise} \end{cases} \mid \ulcorner \langle x| \urcorner_{\in T(C \otimes_{\mathfrak{P}} D)} \}$$

We walk through the construction case-by-case. For the first case top-to-bottom, if two states $\langle c|, \langle d|$ are obtainable by formally deleting the C and D outputs (using $\multimap$) of some state $\langle x|$ on $C \otimes_{\mathfrak{P}} D$, then we send the pair $(\ulcorner \langle c| \urcorner, \ulcorner \langle d| \urcorner)$ to $\ulcorner \langle x| \urcorner$. Note that in the first case, even if $\langle x|$ is tensor separable as $\langle c| \otimes_{\mathfrak{P}} \langle d|$, formal deletion creates new formal scalars which must also agree by the case guard. The total effect of the first case is to identify when $(\langle c|, \langle d|)$ arise as partial diagrams (where ends are truncated with $\multimap$) of some state $\langle x|$. The second case is where $(\langle c|, \langle d|)$ are not partial diagrams of some $\langle x|$ in this way, in which case we send the pair to their tensor product. The third case is the inverse of the second, sending all $\langle x|$ that are equivalent to a tensor-separable composite state $\langle c|$ and $\langle d|$ to that pair of states. The fourth case is the inverse of the first; if $\langle x|$ has no tensor-separable equivalent, then we project $\langle x|$ to states on C and D by formally deleting the appropriate outputs. The effect of taking equivalence classes makes the first relation as a whole an injective function, and likewise for the second relation.

All that remains is to show that T is a faithful model, i.e., for all lists of colours $[C], [D]$ and for all $f, g : [C] \to [D]$, $f =_{\mathfrak{P}} g \iff T(f) =_{\mathbf{Rel}} T(g)$. For the forward direction $\Rightarrow$, if $f =_{\mathfrak{P}} g$, then for any state $\langle x| : 0 \to [C]$, we have that $\langle x| ; f = \langle x| ; g$. Because $T(\langle x|) = \ulcorner \langle x| \urcorner \in T([C])$, we have $\ulcorner \langle x| ; f = \langle x| ; g \urcorner \in T([D])$, thus $T(f) =_{\mathbf{Rel}} T(g)$. For the converse direction $\Leftarrow$, if $f \neq_{\mathfrak{P}} g$, then by the Lindenbaum-Tarski construction and the relational-freeness of the generator $\multimap$, $\ulcorner \multimap ; f \urcorner \neq \ulcorner \multimap ; g \urcorner$, so $T(f)$ as a relation contains the pair $(\ulcorner \multimap_{[C]} \urcorner, \ulcorner \multimap_{[C]} ; f \urcorner)$ that is not present in $T(g)$, and symmetrically for $(\ulcorner \multimap_{[C]} \urcorner, \ulcorner \multimap_{[C]} ; g \urcorner)$, thus $T(f) \neq T(g)$. □

Corollary 5.2.11 (FinRel will work for bounded-size compositionality)**.** Just limit consideration to diagrams with upper bounds on the maximal number of generators among diagrams in their equivalence classes, or whatever reasonable finiteness constraint you want, I'm not a cop.

5.2.3 Text circuits of unbounded width in noncartesian settings

WHAT'S SPECIAL ABOUT **REL** THAT DISTINGUISHES IT FROM A CATEGORY OF DETERMINISTIC PROCESSES? For one, **Rel** is not cartesian monoidal; while the monoidal product in **Rel** is the categorical product of sets, and every object in **Rel** has a copy-delete comonoid, not every relation is a cohomomorphism with respect to this comonoid (in fact, only the functions are.) It turns out that this property is what allows the existence of tests that differ from deleting. In a cartesian monoidal category, every test is equal to deleting.

placeholder

In a noncartesian monoidal category such as **Rel** or **Stoch** – the category of stochastic maps between measureable sets, we may have tests corrresponding to postselection, which greatly increases our expressive capacity for composing constraints. Let's consider an example in **Stoch**, where states are probability distributions, for example a coinflip as a state on the space $\{H, T\}$. Observe that copying the outcome of a coinflip gives a probability distribution on $\{H, T\} \times \{H, T\}$ that is different from what we obtain by flipping two independent coins, as the first is perfectly correlated and the second is not. So **Stoch** is not cartesian monoidal, while it does have a copy-delete comonoid for every object.

placeholder

The scalars in **Stoch** correspond to probabilities in the range $[0, 1]$. Another way we could generate perfectly correlated coinflips is to flip two coins and throw away any outcomes that are not either $H \times H$ or $T \times T$; diagrammatically, this corresponds to copying the outcomes of the two coins and applying a test to one set of copies. Postselecting will yield outcomes will equal in distribution to copying a single coinflip, but only half of the time.

placeholder

Postselection gates are commutative, so in a circuit made up entirely of postselection gates, the only relevant size measure is the width of the circuit.

placeholder

Although it is a very stupid method, one way to achieve unbounded-width compositionality using postselection gates is to just continue sampling randomly until you find a sample that passes all the postselection tests. For example, suppose we have a group of people $\{\texttt{Alice}, \texttt{Bob}, \texttt{Claire}, \cdots\}$ standing in a queue modelled as a unit interval, and we have one binary linguistic relation $X\texttt{is-in-front-of}Y$, which postselects with condition $X > Y$ for $X, Y \in [0, 1]$. Assume every person's initial state is the uniform random distribution on $[0, 1]$. Then we can model the text `Alice is in front of Bob.  Claire is in front of Alice.` as:

placeholder

Any postselected sample in this case is a satisfying instance of our linguistic positions: an assignment of possible positions in the queue to people such that Alice really is in front of Bob and Claire is really in front of Alice. In this case, we can have any finite number of people and it will remain true that if we manage to obtain a postselected sample, it will be a configuration that satisfies our constraints. Moreover, no globally coordinating gate is required in this example. So by introducing random variables as initial states and postselection, we have shown that it is (in principle, and in some cases) possible to obtain approximations for text circuit PROPs using deterministic neural nets without global coordination gates and without the restriction of fixed width.

5.2.4 A value proposition for quantum machine learning

Since I am from the quantum group in my department, I should probably write something extolling quantum computers, so here is the token gesture. A practicality issue arises for the random sampling approach in the form of postselection, which is the procedure of throwing away samples that do not adhere to the constraints we have set and rerolling the dice. For very specific constraints that jointly independent samples are very unlikely to satisfy, we may have to reroll many times before we obtain a satisfying instance, which may take longer than we are willing to wait. While I am sure there are clever ways to mitigate the problem of postselection on a classical computer beyond my current knowledge, let me suggest a *possibility* to avoid postselection altogether; that is, every measurement will always return a satisfying instance. The main ingredient is a parameterised quantum circuit, which consists of unitary gates controlled by classical parameters; just as a neural net is a classically-parameterised process where the parameters determine weights and biases per neuron, classical parameters can determine the preparation of e.g. phase states within a unitary quantum gate. We only want to traffic in causal quantum processes so we never have to worry about postselection, i.e. the only test we ever want to apply is discarding. So without loss of generality we express each parameterised gate as the following composite by Stinespring dilation.

placeholder

That is, we deal with gates that consist of classically-parameterised unitaries and a classically-parameterised ansatz state on ancilla wires that we discard. Since the only test we ever apply is discarding, we do not have to postselect. **If** we can find classical parameters for each gate such that the equations of a text circuit PROP are satisfied, then we we would have a faithful model that would in principle keep the benefit of width-unbounded compositionality as in the probabilistic case, but without the drawbacks of postselection. To be balanced, I should qualify how big of an "if" we are dealing with. There are at least three obstacles that spring to mind. First is practical: the space of parameters for parameterised quantum circuits are not easily differentiable against a loss function unlike their classical counterparts and batching is difficult when dealing with ensembles of parameterised processes in different configurations, so training takes more time. Second is theoretical: there is more work to be done to understand what kinds of text circuit PROPs may be modelled faithfully in principle by quantum and classical gates, and in particular whether bounded-depth compositionality transfers from **FinRel** to **FdHilb** with quantum processes. Third is mystical-heuristical: we can approximate that finding a satisfying instance by random sampling on a classical computer takes something like $\mathcal{O}(e^N)$ samples where N is the number of gates, so if we believe in a complexity-theory analogue of "no free lunches", finding appropriate parameters for an ensemble of quantum or classical gates may be around that hard as well.

BUT WOULDN'T IT BE NICE?

5.3 Modelling metaphor

I will take a *metaphor* to be text where systematic language for one kind of concept is used to structure meanings for another kind of concept where literal meanings cannot apply. This may subsume some cases of what would otherwise be called *similes* or *analogies*.

THE IDEA: 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 Kelvin to wavelengths of light, and the restriction of this mapping to the visible spectrum gives the so-called "colour temperature" framework used by colourists.

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 "candle", "incandescent", "daylight", which obey both temperature-relations (e.g. candle is a lower temperature than incandescent) and colour-relations (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. 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 to reason about and calculate with metaphors such as "Time is Money".

THE MOTIVATION:

5.3.1 Orders, Temperature, Colour, Mood

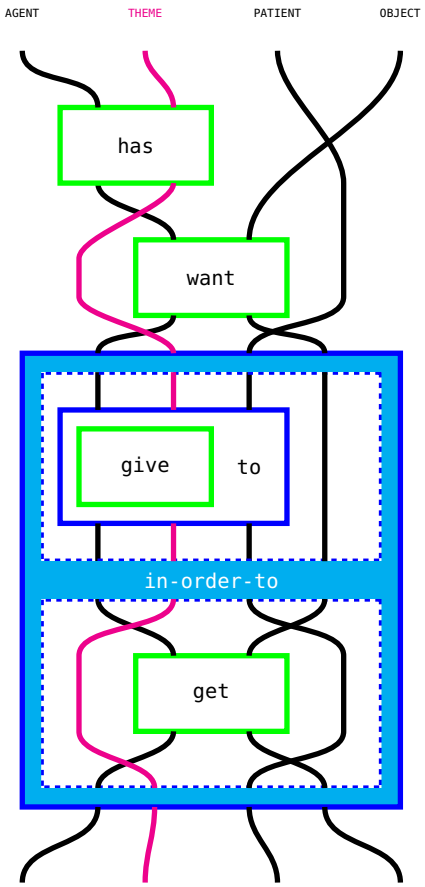
5.3.2 Complex conceptual structure

TIME IS MONEY

```
"Do you have time to look at this?"
"This is definitely worth the time!"
"What a waste of time."
```

I will work through an example of partially using the concept of Money to structure that of Time. Part of the concept of Money is that it can be *exchanged* for something. The concept of exchange can be glossed approximately as the following text, with variable noun-entries capitalised.

```
AGENT has THEME.
AGENT wants OBJECT.
AGENT gives THEME to PATIENT in-order-to get OBJECT.
```



The

5.3.3